Universität Rostock

Traditio et Innovatio

Datenbankgestützte Erkennung von Mustern in trainierten gefalteten neuronalen Netzen (CNN)

Database-driven Recognition of Patterns in trained Convolutional Neural Networks (CNN)

Name:	Florian Baldauf
Matrikelnummer:	217 203729
Abgabedatum:	19.08.2022
Betreuer und Gutachter:	Prof. Dr. Konrad Engel Universität Rostock Institut für Mathematik
Gutachter:	Prof. Dr. Andreas Heuer Universität Rostock Institut für Informatik

Masterarbeit am Institut für Mathematik
Mathematisch-Naturwissenschaftliche Fakultät

Zusammenfassung

Beim Maschinellen Lernen werden Algorithmen implementiert, um Muster und Korrelationen in Datensätzen zu erkennen und auf Basis dieser Analyse die besten Entscheidungen und Vorhersagen zu treffen. Anwendungen des Maschinellen Lernens werden umso genauer, je mehr Trainingsdaten für den Lernprozess zur Verfügung stehen. Das Verwalten dieser meist riesigen Datenmenge stellt eines der schwierigsten Probleme dar. Um in der Implementierung Performance-Probleme zu lösen, können die verwendete Hardware verbessert, der Code optimiert oder Parallelisierungstechniken wie das berühmte MapReduce-Verfahren benutzt werden. In dieser Arbeit wird die Verwendung einer transparenten Datenbankunterstützung für Big Data Analytics diskutiert. Es wird die Transformation von Algorithmen in SQL-Anfragen bei Künstlichen Neuronalen Netzen(KNN) und gefalteten neuronalen Netzen, kurz CNN, beschrieben. Im Kontext der Klassifikationsaufgabe wird ein CNN-Modell zur automatischen Ziffernerkennung trainiert und die datenbankgestützte Umsetzung der Erkennungsphase vorgestellt.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

List of Algorithms

Verzeichnis der Listings

Symbolverzeichnis

$f * g$	Faltung der Funktionen f und g
$f \circledast g$	zyklische Faltung der Funktionen f und g
$L^1(\mathbb{R}^n)$	Raum der Lebesgue-integrierbaren Funktionen auf $\mathbb{R}^n$
$A_{i,:}$	i-te Zeile einer Matrix A
$A_{:,j}$	j-te Spalte einer Matrix A
$A_{i,j}$	Eintrag der Matrix an der i-ten Zeile und j-ten Spalte
${}_1A$	Anzahl der Zeilen einer Matrix A
${}_2A$	Anzahl der Spalten einer Matrix A
$\theta \in \mathbb{R}$	reeller Schwellwert für ein Perzeptron
$w \in \mathbb{R}^n$	Stellungsvektor beim Perzeptron
Λ_l	FFN mit l Neuronenschichten
$b \in \mathbb{R}^n$	Biasvektor eines neuronalen Netzes
$W \in \mathbb{R}^{m \times n}$	Gewichtsmatrix eines neuronalen Netzes
$\mathcal{W}$	Modellparameter
$\mathcal{H}$	Hyperparameter
$\mathcal{M}$	Datenmenge
$\mathcal{C}$	Menge der Klassenlabel
$\mathcal{T}$	Trainingsmenge
$\mathcal{T}'$	Testmenge
$K \in \mathbb{R}^{k \times k}$	Kern eines gefalteten neuronalen Netzes
$K^{rot180} \in \mathbb{R}^{k \times k}$	ein um 180 Grad gedrehter Kern $K \in \mathbb{R}^{k \times k}$
X	Eingabebild bzw. Karte bei FNN/CNN
$X * K$	Matrixfaltung
I	Einheitsmatrix mit jeweils passender Dimension
$\mathbf{1}$	Einsvektor bzw. Einsmatrix mit jeweils passender Dimension
A^*	Adjungierte Matrix der Matrix $A \in \mathbb{C}^{m \times n}$
$u \otimes v \in \mathbb{R}^{m \times n}$	dyadisches Produkt zweier Vektoren $u \in \mathbb{R}^m$ und $v \in \mathbb{R}^n$
$A \odot B \in \mathbb{R}^{m \times n}$	Matrix, welche sich aus der elementweisen Multiplikation der Einträge in den Matrizen $A \in \mathbb{R}^{m \times n}$ und $B \in \mathbb{R}^{m \times n}$ ergibt
$\mathbf{val}[k]$	Arrayzugriff auf das Array **val** an k-ter Stelle
$[n]$	Menge $\{1, \ldots, n\}$
S_n	symmetrische Gruppe, die aus allen Permutationen einer n-elementigen Menge besteht

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 [64, 32, 49]. 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 Assistenzfunktionen 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* [67, 66]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* [22], kurz ML, genutzt werden. Die Analyse von Sensordaten für solche Assistenzsysteme ist daher Forschungsgegenstand in den Bereichen der Big Data Analytics [14] und Künstlichen Intelligenz.

Das PArADISE-Projekt [39] des Lehrstuhls für Datenbank- und Informationssysteme der Universität Rostock beschäftigt sich mit dem Designprozess von Assistenzsystemen 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 Performance-Probleme zu lösen, können die verwendete Hardware verbessert, der Code

optimiert oder Parallelisierungstechniken wie das berühmte *MapReduce*-Verfahren [9] benutzt werden. Ein andere Möglichkeit besteht in der Verwendung einer transparenten Datenbankunterstützung [40] 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 Performance von ML-Algorithmen zu verbessern. Besonders interessant ist hier die Umsetzung von Operationen der linearen Algebra, unter anderem motiviert durch Test-of Time-Award-Winner Dan Suciu [65]. Die Transformation von Matrixoperationen in parallele relationale Datenbanksysteme bei dichtbesetzten bzw. dünnbesetzten Problemen wird beispielsweise in Marten et.al. [43] beleuchtet.

- Konzepte wie *Query Decomposition* [12] und *Answering Queries using Views* [1, 37] werden genutzt, um bestimmte Auswertungen von Daten näher an den Sensoren durchzuführen und damit *Privacy*-Aspekte [2] der Benutzer zu berücksichtigen.

- *Data Provenance* [25, 10] 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. [41] wird am Beispiel eines Meeting-Szenarios der Einsatz von Hidden-Markov-Modelle untersucht. Es wird erläutert, wie die Erkennungsphase eines zuvor trainierten Modells datenbankgestützt in parallelen 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-Datenbanksystemen 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 25, in eine Folge von SQL-Anweisungen zu übersetzen. Der Speicher- und Zeitaufwand dieser Anweisungen ist in Abhängigkeit von der Dimension von X zu untersuchen.*

CNN werden zur Lösung von Aufgaben der Computergrafik [31, 34, 13] erfolgreich genutzt. Im Kontext der Mustererkennung soll die Klassifikation von digitalisierten Bildern als Anwendungsbeispiel durch ein vorher trainiertes CNN-Modell dienen. In dieser Arbeit werden dazu Grauwertbilder aus dem MNIST-Datensatz [35] 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 kann als Benchmark zur Klassifikation von Grauwertbildern genutzt werden. Um dabei erfolgreich zu sein, muss ein CNN vorher mithilfe von Trainingsdaten angelernt werden. Hier wird Backpropagation, etabliert von Rumelhart et. al. [55], als Lernalgorithmus basierend auf dem Gradientenverfahren genutzt.

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

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

Problem 3. *Angenommen, es wird ein trainiertes CNN als Modell zur Klassifikation von Grauwertbildern aus dem MNIST-Datensatz genutzt. Die Vorwärtsrechnung, vgl. Definition 30, ist 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 wird der Begriff der linearen 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 vorwärtsgerichtete 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 werden die Faltungsoperation und die CNN-Architektur 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. Im Abschnitt 5.2 wird hinsichtlich Problemstellung 3 eine (objekt-) relationale Umsetzung der Vorwärtsrechnung für ein trainiertes Modell zur Ziffernerkennung vorgestellt. 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 Definitionen 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 Objekten 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. [45]. Dabei ist mit Lernen gemeint, dass das Computerprogramm bezüglich der Aufgabe T sein Leistungsmaß P mit wachsender 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*

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

Trainingsmenge 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, welches 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 Trainingsmenge $\mathcal{T}$ 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 *performance* P des Modells durch die Minimierung solcher Fehlerfunktionen zu steigern.

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

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

durch die Anpassung der Modellparameter $\mathcal{W}$ gelöst werden. Dabei wird $\mathcal{E}$ Fehlerfunktion genannt. Diese sei in dieser Arbeit als stetig differenzierbar vorausgesetzt.

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 Datenmenge $\mathcal{M}' := \{x_i \ : \ 1 \leq i \leq m'\}$ und die Menge $\mathcal{C} := \{K_1, \ldots, K_s\}$ der Klassen gegeben. Weiter sei $f : \mathcal{M}' \to \mathcal{C}$ eine bekannte Klassifikationsfunktion. Dann wird mit*

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

eine Testmenge bezeichnet.

Auf Trainings- bzw. Testmengen lassen sich Fehlerraten als Perfomance-Maße 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 Trainingsmenge. 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 vertiefende Analyse des Overfittings und anderer Probleme sowie deren Behandlung sei auf Goodfellow [22] verwiesen. Fraglich bleibt zudem, unter welchen Voraussetzungen überhaupt eine sinnvolle Approximation möglich ist. Dazu wird im Folgenden das Problem der linearen Trennbarkeit von Punktmengen behandelt.

2.1.2. Trennbarkeit und ein erster Lernalgorithmus

Seien wieder Vektoren $x_1, \ldots, x_m \in \mathbb{R}^n$ aus Objekten $O_1, \ldots, O_m$ extrahiert. 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 [53], kurz PLA, kann genutzt werden, um den Parameter w schrittweise zu bestimmen. Dieser ist im Algorithmus 1 dargestellt. Verändert sich in einem Zyklus, also das einmalige Durchlaufen aller Punkte in $\mathcal{M}$, der Parameter w nicht, so wird der PLA abgebrochen.

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 kann in Rosenblatt [53] gelesen werden. □

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 $i = 1, \dots m$ **do**
      $d = \chi(x) w^T x$
      if $d \leq 0$ **then**
        $w = w + \chi(x) x$
      end if
    end for
  end while

Es lassen sich jedoch einfache Beispiele angeben, bei denen die Punktmengen nicht durch lineare Entscheidungsgrenzen getrennt werden können. Ein Beispiel ist das zweidimensionale XOR-Problem, bei dem die beiden Punktmengen $P_0 = \{(0,0),(1,1)\}$ und $P_1 = \{(0,1),(1,0)\}$ getrennt werden sollen. Hier scheitert der PLA, da die Punktmengen P_0 und P_1 nicht linear trennbar sind. Um solche Aufgaben zu lösen, ist es notwendig, komplexe Entscheidungsgrenzen zu ermitteln.

Perzeptronen und neuronale Netze, welche im Kapitel 3 vorgestellt werden, sind in der Lage, solche Entscheidungsgrenzen auch bei Multiklassenproblemen, also $s \geq 2$, widerzuspiegeln. Um das Problem (2.2) zu lösen, wird hier die numerische Minimierung von stetig differenzierbaren Funktionen benötigt. Dazu wird das allgemeine Abstiegs-Verfahren genutzt, welches in der Literatur auch Gradientenverfahren genannt wird.

Definition 8 (Abstiegsrichtung)**.** *Sei $f : \mathbb{R}^n \to \mathbb{R}$ stetig differenzierbar. Dann heißt $z \in \mathbb{R}^n$ Abstiegsrichtung für f im Punkt $x \in \mathbb{R}^n$ genau dann, wenn*

$$z^T \nabla f(x) < 0$$

gilt.

Ziel des Verfahrens ist, für eine Funktion f stationäre Punkte x^* mit $\nabla f(x^*) = 0$ näherungsweise zu bestimmen. Es werden also Punkte berechnet, welche die notwendige Bedingung einer lokalen Minimalstelle erfüllen. Aus folgendem Lemma ergibt sich das Gradientenverfahren, siehe Algorithmus 2.

Lemma 1. *Sei $f : \mathbb{R}^n \to \mathbb{R}$ stetig differenzierbar und z eine Abstiegsrichtung für f im Punkt x. Dann existiert ein λ_0, sodass*

$$f(x + \lambda z) < f(x), \quad \forall \lambda \in (0, \lambda_0]$$

gilt. Die reelle Zahl λ wird als Schrittweite bezeichnet.

Beweis. Ein Beweis wird in [48] vorgeführt. □

Algorithm 2 Das allgemeine Gradientenverfahren

Require: f stetig differenzierbar, Fehlertoleranz ε, Schrittweite λ
Ensure: stationärer Punkt x
  Wähle Startwert x beliebig
  while $||\nabla f(x)|| > \varepsilon$ **do**
    Bestimme Abstiegsrichtung z für f in x
    Setze $x = x + \lambda z$
  end while

Für eine vertiefende Analyse von Architekturen, Funktionalitäten und Implementierungsmöglichkeiten des Maschinellen Lernens sei auf das umfangreiche Buch von Goodfellow [22] verwiesen.

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. [26]. Zum Abfragen von Datenbeständen wird die Datenbanksprache SQL im Abschnitt 2.2.3 eingeführt und deren theoretische Grundlagen 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 nichtleerer 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 $w \in \mathrm{dom}(A)$ einen 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. [26])**.** *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 bestimmte Funktionen zur Verwaltung der Datenbank unter gewissen Anforderungen liefert. Typischerwiese sind die von Edgar F. Codd etablierten Anforderungspunkte, siehe [26], von einem Datenbankmanagementsystem umzusetzen. Eine der geforderten Funktionen 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. [27, 26] 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 Relationenalgebra 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
$$r_1 \cup r_2 := \{t \mid t \in r_1 \vee t \in r_2\}$$
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 $\text{dom}(A) = \text{dom}(B)$ 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 [26].

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 Funktionen, 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 [20] verwiesen, um die angepasste Definition des Operatorensystems ω und die zugrunde liegende Theorie nachzuvollziehen.

Der Gruppierungsoperator 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 sind. 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 1 gegeben. Die Relation

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

Tabelle 1.: Zu sehen ist die Beispielrelation **A** 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 Gruppierungsoperation

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

j	v
1	7
2	2

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

Der erweiterte Projektionsoperator

Des Weiteren soll der Projektionsoperator $\pi_F(r)$ auf tupelweise arithmetische Berechnungen bezüglich einer oder mehrerer 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 ein 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 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 Datenbanksprache 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 einzelnen Standards sei auf die entsprechende Literatur verwiesen. Ein guter Überblick ist in Heuer et. al. [26] 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 kartesischen Produkt kombiniert werden können. Ebenfalls optional ist die **WHERE**-Klausel, welche Selektionsbedingungen bezüglich Konstanten oder Attributen enthält.

Um den Anfragemechanismus nachzuvollziehen, seien im Folgenden zwei Relationen **Angestellte** und **Projekt** wie in den Tabellen 2 und 3 gegeben.

Angestellte				
ID	Name	Spezialisierung	Projektnummer	Gehalt
1	Martin	Elektrotechnik	3	2300
2	Lennardt	Informatik	1	1500
3	Johann	Informatik	3	1800
4	Jana	Maschinenbau	3	2100
5	Antonia	Buchhaltung	2	2000

Tabelle 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	Forschungsabteilung	40000	Schwerin	offen

Tabelle 3.: Abgebildet ist die Beispielrelation Projekt mit den Attributen Projektnummer, Projektname, Budget, Ort und Status.

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 offenen Projekten. Dementsprechend ist die Ergebnisrelation in Tabelle 4 dargestellt.

Ergebnis	
A.Name	P.Projektname
Martin	Forschungsabteilung
Johann	Forschungsabteilung
Jana	Forschungsabteilung
Antonia	Verwaltung

Tabelle 4.: Zu sehen ist die Ergebnisrelation der zuvor beschriebenen Anfrage (2.5).

Die Aliasnamen „A“ und „P“ dienen hier zur Übersicht. Die klassische 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 Attributwertkombinationen. 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
GROUP BY A.Spezialisierung (2.6)

verwendet. Diese berechnet die Angestelltenkosten und gruppiert sie nach den jeweiligen Spezialisierungen. Die Ergebnistabelle ist in Tabelle 5 dargestellt. Ein Beispiel einer geschalteten Anfrage zur Bestimmung des Angestellten mit dem höchsten Gehalt ist

Ergebnis	
A.Spezialisierung	Angestelltenkosten
Elektrotechnik	2300
Informatik	3300
Buchhaltung	2000
Maschinenbau	2100

Tabelle 5.: Es ist die Ergebnisrelation der zuvor beschriebenen Anfrage (2.6) abgebildet.

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-Anfragen immer als Multimengen fungieren. Um eine Duplikateliminierung zur Verfügung zu haben, kann die **SELECT**-Klausel durch den **DISTINCT**-Operator erweitert werden. Die Repräsentation der Operatoren der Relationenalgebra durch SQL-Anfragen wird in Tabelle 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 6.

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 6.: Vergleich des Operatorensystems aus Abschnitt 2.2.2 mit dem SQL-Anfragekern, vgl. [26].

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 Repräsentation 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.

Dicht besetzte Matrizen

Im Folgenden wird das *Coordinate-Schema* [38] als Schema für die Darstellung dichtbesetzter Matrizen genutzt. Dieses Schema gestaltet sich als einfach und ist daher weit verbreitet [56]. Für eine weiterführende Diskussion anderer Darstellungsmöglichkeiten und deren Vor- und Nachteile sei auf Marten [38] 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 Indizes i und j sind jeweils als ganzzahlige Datentypen, engl. integer, dargestellt. Die Matrixeinträge werden in double precision [28] 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.

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 7.

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

Tabelle 7.: Das Coordinate-Schema zur Matrix A aus Beispiel 1.

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 [33] 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

$$\begin{aligned}&\textbf{SELECT } x.i \textbf{ AS } i,\ r * x.v + (s * y.v) \textbf{ AS } v\\&\textbf{FROM } x \textbf{ JOIN } y \textbf{ ON } x.i = y.i\end{aligned}$$

Ähnlich ergibt sich die Matrixaddition $rA + sB$ zu

$$\begin{aligned}&\textbf{SELECT } A.i \textbf{ AS } i,\ A.j \textbf{ AS } j, (r * A.v) + (s * B.v) \textbf{ AS } v\\&\textbf{FROM } A \textbf{ JOIN } B \textbf{ ON } A.i = B.i \textbf{ AND } A.j = B.j\end{aligned}$$

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 Operationen sind die Matrixvektor- und Matrixmatrixmultiplikation. Durch Kombination vorheriger Basisoperationen ergibt sich die entsprechenden Transformationen für die Matrixvektormultiplikation $Ax \in \mathbb{R}^m$ einer Matrix $A \in \mathbb{R}^{m \times n}$ und eines Vektors $x \in \mathbb{R}^n$ zu

$$\begin{aligned}&\textbf{SELECT } A.i \textbf{ AS } i,\ \textbf{SUM}(A.v * x.v) \textbf{ AS } v\\&\textbf{FROM } A \textbf{ JOIN } x \textbf{ ON } A.j = x.i\\&\textbf{GROUP BY } A.i.\end{aligned}$$

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* [17] 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. [38])**.** *Eine dünn besetzte Matrix $A \in \mathbb{R}^{m \times n}$ wird als Relation*

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

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

In dem Open-Source-Datenbanksystem PostgreSQL [46], 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, siehe Definition 14, benutzt, während für dünn besetzte Probleme das Spaltenkompression-Schema, vgl. Definition 15, genutzt wird. Beide Darstellungsformen erlauben kompakte SQL-Anfragen für die Matrixmatrix- bzw. Matrixvektormultiplikation. Für eine tiefere Analyse dieser und anderer Darstellungsmöglichkeiten und deren Performance hinsichtlich verschiedenster Basisoperationen sei auf Marten [38] verwiesen.

3. Grundlagen neuronaler Netze

In diesem Kapitel werden Künstliche Neuronale Netze [15], 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 eines 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 [50, 51, 61]. Sie eignen sich daher für viele typische Aufgaben des maschinellen Lernens, beispielsweise für die Klassifikation digitalisierter Objekte.

Im ersten Abschnitt wird das Perzeptron [53] als Grundeinheit eines neuronalen Netzes eingeführt. Im folgenden Abschnitt wird das Konzept der Multi-Layer-Perzeptronen [62] 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 [44] 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 linearen 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 1 dargestellt. In dieser Arbeit wird das Perzeptron oft einfach Neuron genannt. Die Begriffe sind austauschbar und

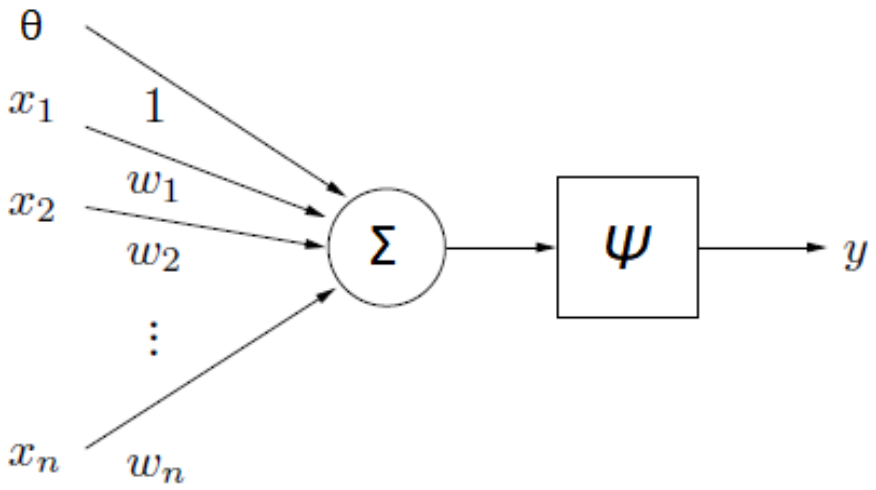

Abbildung 1.: Arbeitsweise eines Perzeptrons mit entsprechender Notation aus Definition 16.

meinen dasselbe. Bei der Wahl der Funktion ϕ gibt es mehrere Möglichkeiten. Wird wie in Minsky [44] die Heavyside-Funktion

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

genutzt, kann das Perzeptron als binärer Klassifikator wie in Abschnitt 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 [8, 7, 55]. Zunächst sind einige Definitionen notwendig, vgl. [23], 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 [52], 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 2. *(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 wie beim Perzeptron, siehe Definition 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 *Neuronen* Φ_i *mit der Beziehung* $y_i = \Phi_i(x) = \phi(W_{i,:}^T x + b_i)$ *für* $1 \leq i \leq m$.

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

Definition 20 (Multi-Layer-Perzeptron, vgl. [23])**.** *Für eine gegebene Anzahl* $1 < l \in \mathbb{N}$ *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 3 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 KNN, MLP und FFN sind austauschbar und meinen dasselbe Modell. Die Funktionsauswertung eines FNN wird im Algorithmus Vorwärtsrechnung 3 festgehalten. Das zuvor angesprochene XOR-Problem kann nun beispielsweise mithilfe eines FNN bestehend aus zwei Schichten gelöst werden [22]. Es lassen sich zwischen Modell- und Hyperparameter von KNN unterscheiden.

Definition 21 (Hyper- und Modellparameter)**.** *Sei für $l \in \mathbb{N}$ ein FNN Λ_l gegeben. Dann werden die Eingabe- und Ausgabedimension s_0, s_l, die Anzahl l der (verdeckten) Schichten sowie die verwendeten Aktivierungsfunktionen 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 Trainingsprozess angepasst werden, sodass die gegebene Aufgabe zufriedenstellend gelöst wird. Wie optimale Modellparameter in einem Trainingsprozess gefunden werden können, wird im folgenden Abschnitt 3.3 erläutert.

3.3. Optimale Parameterwahl bei neuronalen Netzen

In diesem Abschnitt steht das Klassifikationsproblem für digitalisierte Objekte im Mittelpunkt. Dazu werden FNN als Modelle $\tilde{f}$ zur Approximation von Klassifikationsfunktionen $f : \mathcal{M} \to \mathcal{C}$, vgl. Abschnitt 2.1.1, eingesetzt. Die Anzahl der Neuronen in der Ausgabeschicht entspricht der Anzahl der verschiedenen Klassen in $\mathcal{C}$, also $s_l = s$. Zur Vereinfachung der Notation sei mit $C := \{1, \dots, s\}$ die Menge der Klassen bezeichnet. Wird für eine Eingabe x die Vorwärtsrechnung $\Lambda(x)$ vorgenommen, so bezeichne $\tilde{f}(x)$ die approximierte Klassifikationsfunktion, welche als

$$\tilde{f}(x) := \underset{k}{\operatorname{argmax}} \ (\Lambda(x))_k$$

definiert ist. Dabei sei bemerkt, dass der Index l, welcher die Anzahl der Schichten von Λ kodiert, für die folgenden Betrachtungen entfällt. Weiter stehe eine Trainingsmenge

$$\mathcal{T} = \{(x_i, c) \in \mathcal{M} \times \mathcal{C} \ : \ 1 \leq i \leq m\}$$

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

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

Sind die Hyperparameter des FFN festgelegt, müssen die Modellparameter $\mathcal{W}$ optimal gewählt werden. In diesem Sinne bedeutet optimal, dass das Minimierungproblem

$$\mathcal{E}(\mathcal{T}, \mathcal{W}) \to \min \tag{3.1}$$

im Raum der Parameter $\mathcal{W}$ gelöst werden soll. Dabei sei bemerkt, dass dies unabhängig von den Hyperparametern geschieht. Im Folgenden wird der Backpropagationsalgorithmus als ein Lösungsansatz für die Optimierungsaufgabe (3.1) vorgestellt.

Backpropagationsalgorithmus

Bei FFN wird Backpropagation, etabliert von Rumelhart et. al. [55], als Lernalgorithmus genutzt, um eine gewählte Fehlerfunktion $\mathcal{E}$ zu minimieren. Dabei wird das Gradientenverfahren, vgl. Algorithmus 2, zur numerischen Minimierung von $\mathcal{E}$ im Raum der Modellparameter $\mathcal{W}$ genutzt. Es sei bemerkt, dass das Finden von globalen Minima durch den Backpropagationsalgorithmus keineswegs garantiert ist. In dieser Arbeit wird $\mathcal{E}$ immer als stetig differenzierbare Funktion vorausgesetzt, damit das Gradientenverfahren angewendet werden kann. Ebenso sollten alle verwendeten Aktivierungsfunktionen aus Definition 18 als stückweise stetig differenzierbare Funktion vorausgesetzt werden. Die Optimierung der Parameter geschieht iterativ, dargestellt mit dem Index n, und besteht aus zwei Schritten. Für festes n wird zuerst eine Abstiegsrichtung

$$\Delta_n := \nabla_{\mathcal{W}} \mathcal{E}(\mathcal{T}, \mathcal{W}) \tag{3.2}$$

berechnet und dann werden die Parameter

$$\mathcal{W}_{n+1} := \mathcal{W}_n + \lambda \Delta_n \tag{3.3}$$

aktualisiert. Es werden Gradienten der Fehlerfunktion bezüglich der Gewichtsmatrizen und Biasvektoren ermittelt und anschließend werden jene Parameter mit einer wählbaren Lernrate $\lambda \in \mathbb{R}$ in Gleichung (3.3) angepasst. In Gleichung (3.2) 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 2 (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*

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

Beweis. Ein Beweis kann in Forster [19] gelesen werden. □

Die effiziente Berechnung der Abstiegsrichtungen gehört zu den schwersten Aufgaben des Maschinellen Lernens, siehe [36]. Der Online-Backpropagationsalgorithmus, in der Literatur auch als *Stochastic Gradient Descent* bezeichnet, wird im Folgenden zur Optimierung der Modellparameter 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. Der Fehler in der Ausgabeschicht wird mithilfe einer Fehlerfunktion $\mathcal{E}$ berechnet.
- Bei der Rückwärtsrechnung wird für jedes Neuron ein lokaler Fehler bezüglich $\mathcal{E}$ berechnet, beginnend in der Ausgabeschicht über die verdeckten Schichten bis zur Eingabeschicht. Der Fehler wird über die Ausgabe- zur Eingabeschicht zurück propagiert. Dabei werden die Gewichtungen der Neuronenverbindungen und Schwellwerte abhängig von ihrem Einfluss auf den Fehler direkt bzw. indirekt geändert.

Als Fehlerfunktion $\mathcal{E}$ wird im weiteren Verlauf die mittlere quadratische Abweichung verwendet.

Definition 22. *Seien eine Trainingsmenge* $\mathcal{T}$ *und Modellparameter* $\mathcal{W}$ *eines FFN* Λ *gegeben. Dann wird die mittlere quadratische Abweichung als*

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

definiert, wobei $y = \Lambda(x)$ *der Ausgabevektor des FFN und* $t = t(x, c)$ *der Zielvektor des Datenpaars* (x, c) *ist.*

Beim Online-Verfahren wird der Fehler

$$E_x = \frac{1}{2}||y - t(x, c)||_2^2 = \frac{1}{2} \sum_{k=1}^{s_l} (y_k - t_k)^2 \tag{3.4}$$

für eine einzige Eingabe x berechnet. Es müssen die Abstiegsrichtungen

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

für $1 \leq m \leq l$ ermittelt werden. Die Ausgabeschicht eines l-schichtigen FFN Λ besitze s_l Neuronen. Im Folgenden wird die Notation wie in [18] 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 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. Zunächst wird nur eine verdeckte Schicht mit s_j Neuronen betrachtet. Mit i ist ein Eingabeneuron, j ein verdecktes Neuron und k ein Ausgabeneuron gemeint. Außerdem werden die Gradienten komponentenweise berechnet und daher die Modellparameter komponentenweise aktualisiert. Das mehrmalige Anwenden der Kettenregel, vgl. Definition 2, auf die Fehlerfunktion in Gleichung (3.4) liefert

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

mit

$$\delta_k := e_k \psi'(V_k) = \frac{\partial E_x}{\partial V_k}$$

als sogenannte lokale Fehler für $1 \leq k \leq s_l$. Für die Ausgabeschicht lässt sich der lokale Fehler direkt berechnen. Für die Schwellwerte gilt analog

$$\Delta b^l_k = \frac{\partial E_x}{\partial b^l_k} = \delta_k.$$

Die lokalen Fehler bezüglich E_x für die Parameter der verdeckten Schicht lassen sich indireket mithilfe der lokalen Fehler δ_k der darüberliegenden Ausgabeschicht berechnen.

Es gilt

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

mit x_i als Eingabeneuron und

$$\delta_j := \left(\sum_{k=1}^{s_l} \delta_k w_{k,j}^l \right) \psi'(V_j) = \frac{\partial E_x}{\partial V_j} \tag{3.5}$$

als lokale Fehler der verdeckten Neuronen für $1 \leq j \leq s_j$. Hier werden die lokalen Fehler bezüglich E_x von der darüberliegenden Schicht zurück propagiert. Für die Schwellwerte gilt wieder

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

Backpropagation für mehrere Neuronenschichten

Die Verallgemeinerung für beliebig viele verdeckte Schichten lässt sich in Abhängigkeit der lokalen Fehler beschreiben. Zur Vereinfachung der Notation beschreibe L die Anzahl der Neuronenschichten eines FFN Λ_L. Seien mit $w_{j,i}^{(l)}$ und $b_j^{(l)}$ Parameter einer beliebigen Neuronenschicht l bezeichnet. Dann gilt

$$\begin{aligned}
\Delta w_{j,i}^{(l)} &= \frac{\partial E_x}{\partial w_{j,i}^{(l)}} = \delta_j^{(l)} z_i^{(l-1)} \\
\Delta b_j^{(l)} &= \frac{\partial E_x}{\partial b_j^{(l)}} = \delta_j^{(l)}
\end{aligned}$$

mit

$$\delta_j^{(l)} = \begin{cases} e_j \psi'(V_j^{(L)}), & \text{wenn } j \text{ ein Ausgabeneuron ist,} \\ \left(\sum_k \delta_k^{(l+1)} w_{k,j}^{(l+1)} \right) \psi'(V_j^{(l)}), & \text{wenn } j \text{ ein verdecktes Neuron ist.} \end{cases}$$

Dabei ist $z_i^{(l-1)}$ die Aktivierung des Neurons i auf der Schicht $l-1$, $e_j = y_j - t_j$ der Fehler des j-ten Ausgabeneurons, $V_j^{(l)}$ die Netzeingabe des j-ten Neurons der Schicht l,

$\delta_k^{(l+1)} = \frac{\partial E_x}{\partial V_k^{(l+1)}}$ der lokale Fehler der Schicht $l+1$ und $w_{k,j}^{(l+1)}$ entsprechende Gewichte. Die Online-Backpropagation eines FFN Λ wird als komponentenweiser und iterativer Trainingsalgorithmus beschrieben. Daher taucht im Algorithmus 4 eine Schleife in Abhängigkeit von n auf. Als Abbruchbedingung kann eine maximale Anzahl N von Iterationen vorgegeben werden. Andere Abbruchbedingungen können mit der Norm der Abstiegsrichtungen oder abhängig von der Größe des Trainingsfehlers mit einer Fehlertoleranz $\varepsilon > 0$ formuliert werden.

Algorithm 4 Online-Backpropagation für ein FFN Λ_L

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_L
  Berechne für die Eingabe (x, c) den Zielvektor $t = t(x, c)$
  $n = 0$
  $E = \inf$
  while $E > \varepsilon$ **or** $n < N$ **do** ▷ Abbruchbedingung, siehe Text
    for $(x, c) \in \mathcal{T}$ **do**
      Vorwärtsrechnung $y = \Lambda(x)$
      for $k = 1, \ldots, s_L$ **do**
        $\delta_k^{(L)} = (y_k - t_k)\psi'(V_k^{(L)})$
      end for
      Backpropagation
      for Schichten $l = L-1, \ldots 1$ **do**
        for $j = 1, \ldots, s_l$ **do**
          $\delta_j^{(l)} = \left(\sum_{k=1}^{s_{l+1}} \delta_k^{(l+1)} w_{k,j}^{(l+1)}\right) \psi'(V_j^{(l)})$
        end for
      end for
      Berechne Gradienten und aktualisiere Gewichte
      for $l = 1, \ldots, L$ **do**
        for $j = 1, \ldots, s_l$ **do**
          for $i = 1, \ldots s_{j-1}$ **do**
            $\Delta w_{j,i}^{(l)} = \delta_j^{(l)} z_i^{(l-1)}$
            $w_{j,i}^{(l)} = w_{j,i}^{(l)} + \lambda \Delta w_{j,i}^{(l)}$
          end for
          $\Delta b_j^{(l)} = \delta_j^{(l)}$
          $b_j^{(l)} = b_j^{(l)} + \lambda \Delta b_j^{(l)}$
        end for
      end for
      $n = n + 1$
    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 oft adaptive Verfahren genutzt, welche

vorangegangene Gradienten berücksichtigen und die Lernrate so anpassen. Bekannte Verfahren sind *Nesterov accelerated gradient* [59], *AdaGrad* [16], *RMSProp* [60] sowie *Adam* [30]. So sollen Probleme wie des *vanishing gradients* oder des *exploding gradients* vermieden werden, siehe dazu [24]. Für eine tiefere Analyse des Gradientenverfahrens und dessen Varianten sei auf die jeweiligen Arbeiten beziehungsweise als Zusammenfassung auf Ruder [54] verwiesen. Darüber hinaus gibt es andere Techniken wie die Regularisierung, um den 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 Feed-Forward-Netze als Modelle eingeführt, um Klassifikationsaufgaben 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 des Optimierungsproblems (3.1) kann wegen der Komplexität allgmemeiner Fehlerfunktionen $\mathcal{E}$ bzw. der großen Menge von Parametern $\mathcal{W}$ selten direkt angegeben werden [6]. Daher wird das Gradientenverfahren als iterativer Ansatz zur numerischen Minimierung der Fehlerfunktion genutzt. Dabei sind partielle Ableitungen der Fehlerfunktion bezüglich der Parameter der Neuronenschichten nötig, welche im Backpropagationsalgorithmus direket bzw. indirekt 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 10 Ausgabeneuronen genutzt und jedes Pixel eines Bildes mit den Abmessungen 1000×1000 als Merkmal genutzt, so ergeben sich bereits $10^7 + 10$ freie Parameter. Stehen nur relativ wenige Trainingsdaten zur Verfügung, ist die Struktur des FFN zu komplex und dies kann zur Überanpassung führen [11, 5]. Außerdem steigt der Zeitaufwand der Trainingsphase mit wachsender Parameteranzahl. Diese Anzahl muss also deutlich reduziert werden. Konzepte wie *Parameter Sharing* und spärliche Konnektivität, engl. *sparse connectivity*, erlauben die Reduktion der Parameteranzahl und werden ausführlich in Goodfellow [22] beschrieben.

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, vgl. Goodfellow [22], 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 und spärliche Konnektivität, um die Anzahl der freien Parameter drastisch zu reduzieren. So gelingt es besonders bei Aufgaben der Computergrafik [31, 34, 13] die Generalisierungsrate gegenüber klassischen FFN zu erhöhen. An dieser Stelle sei auf Goodfellow [22] verwiesen, um nachzuvollziehen, wie diese Konzepte im Detail von CNN-Modellen umgesetzt werden.

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 4 für CNN verallgemeinert. Im letzten Abschnitt 4.4 wird der Backpropagationsalgorithmus für ein konkretes Modell zur Klassifikation von Ziffern angewendet. An dieser Stelle wird der MNIST-Datensatz hinsichtlich Problemstellung 3 vorgestellt.

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

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

Beweis. Ein Beweis dieser Rechenregeln kann in Werner [63] 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 3 gelten analog.

Definition 24 (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.

Im 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 25 (Matrixfaltung, vgl. [23])**.** *Seien die Matrizen $X \in \mathbb{R}^{h \times b}$ und $K \in \mathbb{R}^{k \times k}$ gegeben. Sei $l = \lfloor k/2 \rfloor$. Die Matrixfaltung $Y = X * K \in \mathbb{R}^{h \times w}$ 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 dieser Definition besitzt das Ergebnis Y der Faltung die gleichen Abmessungen wie X.

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*

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

beziehungsweise der Kreuzkorrelation mit

$$Y_{i,j} = \sum_{u=-l}^{l} \sum_{v=-l}^{l} X_{i+u,j+v} K_{u,v} \tag{4.3}$$

und $X_{i,j} = 0$ für $i \notin [h]$ und $j \notin [b]$ angegeben. Das Auffüllen mit Nullen am Rand von X wird mit zero padding bezeichnet. Die Kreuzkorrelation in Gleichung (4.3), in der Fachliteratur oft auch als Faltung bezeichnet, stellt die Faltung in Gleichung (4.2) mit gedrehtem Kern K^{rot180} dar. Dabei ist

$$K^{rot180}_{-u,-v} := K_{u,v}.$$

Hier werden die Kerne speziell indiziert. Ist beispielsweise $k = 3$, so ist $K \in \mathbb{R}^{3\times 3}$ durch

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

und K^{rot180} durch

$$K^{rot180} = \begin{pmatrix} K_{1,1} & K_{1,0} & K_{1,1} \\ K_{0,1} & K_{0,0} & K_{0,-1} \\ K_{-1,1} & K_{-1,0} & K_{-1,-1} \end{pmatrix}$$

gegeben. In dieser Arbeit wird die Operation in Gleichung (4.3) als Matrixfaltung bezeichnet.

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

Bemerkung 3. *Für Schrittweiten $s_h, s_b \in \mathbb{N}$ ist die reduzierte Matrixfaltung $Y = X * K$ als*

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

definiert. Für $s_h = s_b = 1$ ergibt sich die Standardvariante wie in in Gleichung (4.1).

4.2. CNN-Architektur

Beim maschinellen Lernen sind Eingabedaten oft als mehrdimensionale Arrays dargestellt, welche eine oder mehrere Achsen repräsentieren, wobei die Ordnung dieser eine Rolle spielt. Bei digitalisierten Bildern sind das beispielsweise 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 [23].

Definition 26 (Gefaltete Übertragungsfunktion)**.** *Sei ein vierdimensionales Array, in Zukunft oft Kern genannt, $K \in \mathbb{R}^{z_{out} \times z_{in} \times k \times k}$ 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

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

wird gefaltete Übertragungsfunktion bezeichnet. Mit $*$ ist die Matrixfaltung wie in Definition 25 gemeint und mit $\cdot$ werden beliebige Raumachsen bezeichnet. Das Ergebnis der Übertragungsfunktion wird Netzeingabe genannt. Die Skalare α_{qp} können als lernbare Parameter genutzt werden. In dieser Arbeit gelte $\alpha_{qp} = 1$ für alle $q \in [z_{out}]$ und $p \in [z_{in}]$.

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 27 (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

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

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 28 (Pooling-Schicht, vgl. Grüning [23])**.** *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

$$\Psi_{pool,T}^{p_h,p_b}(X)_{i,j,l} := \underset{\substack{(i-1)\cdot p_h < i' \leq \min\{i\cdot p_h, {}_1X\} \\ (j-1)\cdot p_w < j' \leq \min\{j\cdot p_w, {}_2X\}}}{T} (X_{i',j',l})$$

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 Informationen 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-Funktionen sei auf Yu et. al. [68] 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-Funktion definiert.

Definition 29. *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*

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

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

Dies erlaubt die Definition des CNN-Modells als Kombination aus Faltungsschichten, Pooling-Schichten und Neuronenschichten eines FFN.

Definition 30. *(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

$$y = \Psi^{W^{(l)},b^{(l)},\psi_l} \circ \ldots \circ \Psi^{W^{(1)},b^{(1)},\psi_1} \circ T_f \circ \Psi_{conv}^{K^{(c)},b^{(c)},\psi_c} \circ \ldots \circ \Psi_{conv}^{K^{(1)},b^{(1)},\psi_l}(X) \tag{4.4}$$

darstellen. Dabei seien wieder die Dimensionen der Parameter passend gewählt, sodass die Komposition wohldefiniert ist. In Gleichung (4.4) können zwischen den Faltungsschichten Pooling-Schichten $\Psi_{pool,T}^{p_h,p_b}(X)$ geschaltet werden.

Auch CNN bestehen aus Hyper- und Modellparametern.

Definition 31 (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 4 soll nun für das CNN-Modell verallgemeinert werden. Als Fehlerfunktion wird wieder die mittlere quadratische Abweichung, vgl. Gleichung (3.4), 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.

Zur Vereinfachung werden die trainierbaren Parameter der Kerne und Gewichtsmatrizen als Gewichte $w_{j,i}^{(l)}$ bezeichnet. Gemeint ist also das Gewicht zwischen Neuron i und Neuron j in der (Faltungs)-Schicht l. 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, welcher im Folgenden verallgemeinert wird. Die Funktion ψ repräsentiere eine Aktivierungsfunktion einer (Faltungs)-Schicht und Pooling-Funktionen werden mit T sowie Flatten-Funktion mit T_f bezeichnet.

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 Bemerkung 2, komponentenweise als

$$y_j^{(l)}(x,y) = \psi\left(\sum_{i=1}^{z_{in}} \sum_{(u,v)\in F} y_i^{(l)}(x+u, y+u)\, w_{j,i}^{(l)}(u,v) + b_j^{(l)}\right) \tag{4.5}$$

schreiben. Es ist zu beachten, dass (x, y) hier ein Pixel der Karte j meint und nicht mit Eingabe- bzw. Ausgabevektoren zu verwechseln ist. 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 von $w_{j,i}^{(l)}$ ergibt sich als Summe über alle beteiligten Pixel (x, y) der Merkmalskarte j, also

$$\Delta w_{j,i}^{(l)}(u, v) = \sum_{(x,y)} \left(\delta_j^{(l)}(x, y) y_i^{(l-1)}(x + u, y + v) \right). \tag{4.6}$$

Der Zusammenhang in Gleichung (4.6) ist in Abbildung 2 grafisch dargestellt. Analog ergibt sich für den Schwellwert

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

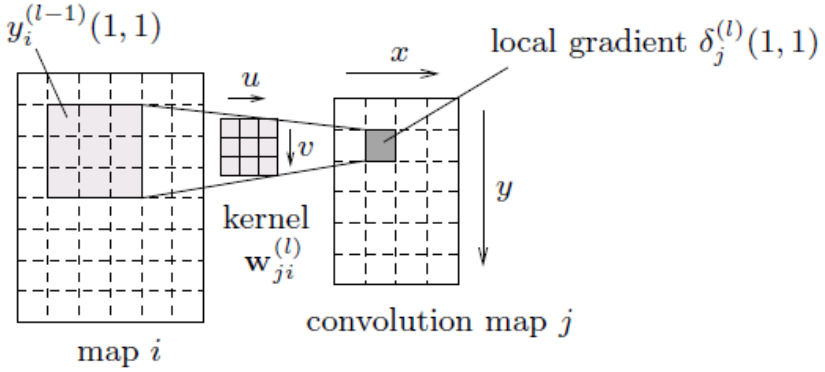

Abbildung 2.: Es ist die Rückwärtsrechnung bei Faltungsschichten dargestellt. Die Abbildung wurde aus [18] entnommen.

Die Berechnung des lokalen Fehlers $\delta_j^{(l)}$ in Gleichung (4.6) der Schicht l ist abhängig von der Schicht $l + 1$. Auch hier wird wieder komponentenweise vorgegangen. Im Folgenden bezeichne $V_j^{(l)}$ die Netzeingabe des Neurons j auf der Schicht l. Ist die Schicht $l + 1$ eine Neuronenschicht mit s_k Neuronen, wird der lokale Fehler in Gleichung (3.5) zu

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

verallgemeinert. Hier ist $\delta_j^{(l+1)}$ der von der Neuronenschicht $l + 1$ zurück propagierte Fehler des Neurons j bezüglich der Fehlerfunktion E und $\psi(V_j^{(l)})$ die entsprechende Aktivierung des Neurons j. Ist die nachfolgende Schicht eine Faltungsschicht, so ergibt sich der lokale Fehler der Schicht l zu

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

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

gilt

$$\delta_j^{(l)}(x,y) = \delta_j^{(l+1)}(\lceil x/p_h \rceil, \lceil y/p_b \rceil)\nabla T. \tag{4.7}$$

Dabei wird der Gradient ∇T jeweils an der zu (x,y) korrespondierenden Stelle ausgewertet und mit dem Fehler der darüberliegenden Schicht multipliziert. Der lokale Fehler wird je nach Wahl der Pooling-Funktion zurück propagiert. Beim Mittelwert-Pooling wird der Fehler gleichmäßig auf alle beteiligten Pixel aufgeteilt. Beim Maximum-Pooling wird der Fehler lediglich auf die Position des Neurons zurück propagiert, welches die größte Aktivierung besitzt. Ist $l+1$ eine Flatten-Schicht mit einer Flatten-Funktion T_f, so gilt

$$\delta_j^{(l)} = T_f^{-1}(\delta_j^{(l+1)}). \tag{4.8}$$

Die Berechnung in Gleichung (4.7) und Gleichung (4.8) wird Upsampling genannt. Die obigen Resultate sind im Algorithmus 5 zusammengefasst. Hier beschreibe L die Anzahl aller Faltungsschichten, Pooling-Schichten und Neuronenschichten zusammen.

Algorithm 5 Online-Backpropagation für gefaltete neuronale Netze, vgl. [18]

Require: Trainingsmenge $\mathcal{T}$, Modellparameter $\mathcal{W}$ und $\mathcal{W}_{conv}$, Fehlerfunktion E, Lernrate λ
Ensure: optimierte Modellparameter $\mathcal{W}, \mathcal{W}_{conv}$

```
Initialisiere zufällig alle Gewichte und Schwellwerte des CNN
Berechne für die Eingabe (x, c) den Zielvektor t = t(x, c)
n = 0
E = inf
while E > ε or n < N do                                   ▷ Abbruchbedingung wie bei Algorithmus 4
    for (x, c) ∈ T do
        Vorwärtsrechnung y
        for k = 1, ..., s_L do
            δ_k^(L) = (y_k − t_k) ψ'(V_k^(L))
        end for
        for Schichten l = L − 1, ... 1 do
            for all Karten/Neuronen j in Schicht l do
                for all Pixel (x, y) do
                    if Schicht l + 1 ist Neuronenschicht then
                        δ_j^(l)(x, y) = Σ_{k=1}^{s_{l+1}} ( Σ_(x,y) δ_k^(l+1) w_{k,j}^(l+1)(x, y) ) ψ'(V_j^(l))
                    end if
                    if Schicht l + 1 ist Faltungsschicht then
                        δ_j^(l)(x, y) = Σ_{k=1}^{z_out} ( Σ_{(u,v)∈F} δ_k^(l+1)(x, y) w_{k,j}^(l+1)(u, v) ) ψ'(V_j^(l))
                    end if
                    if Schicht l + 1 ist Pooling-Schicht then
                        δ_j^(l)(x, y) = δ_j^(l+1)(⌈x/p_h⌉, ⌈y/p_b⌉) ∇T
                    end if
                    if Schicht l + 1 ist Flatten-Schicht then
                        δ_j^(l) = T_f^{-1}(δ_j^(l+1))
                    end if
                end for
            end for
        end for
        Berechne Gradienten und aktualisiere Gewichte
        for l = 1, ..., L do
            if Schicht l ist Neuronenschicht then
                for j = 1, ..., s_l do
                    for i = 1, ... s_{j−1} do
                        Δw_{j,i}^(l) = δ_j^(l) y_i^(l−1)
                        w_{j,i}^(l) = w_{j,i}^(l) + λ Δw_{j,i}^(l)
                    end for
                    Δb_j^(l) = δ_j^(l)
                    b_j^(l) = b_j^(l) + λ Δb_j^(l)
                end for
            end if
            if Schicht l ist Faltungsschicht then
                for j = 1, ..., z_out do
                    for i = 1, ... z_in do
                        for (u, v) ∈ F do
                            Δw_{j,i}^(l)(u, v) = Σ_(x,y) ( δ_j^(l)(x, y) y_i^(l−1)(x + u, y + v) )
                            w_{j,i}^(l) = w_{j,i}^(l) + λ Δw_{j,i}^(l)
                        end for
                    end for
                    Δb_j^(l) = Σ_(x,y) δ_j^(l)(x, y)
                    b_j^(l) = b_j^(l) + λ Δb_j^(l)
                end for
            end if
        end for
        n = n + 1
    end for
    E = (1/2) Σ_{(x,c)∈T} ||y − t||_2^2
end while
```

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 [34] 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 3. 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.

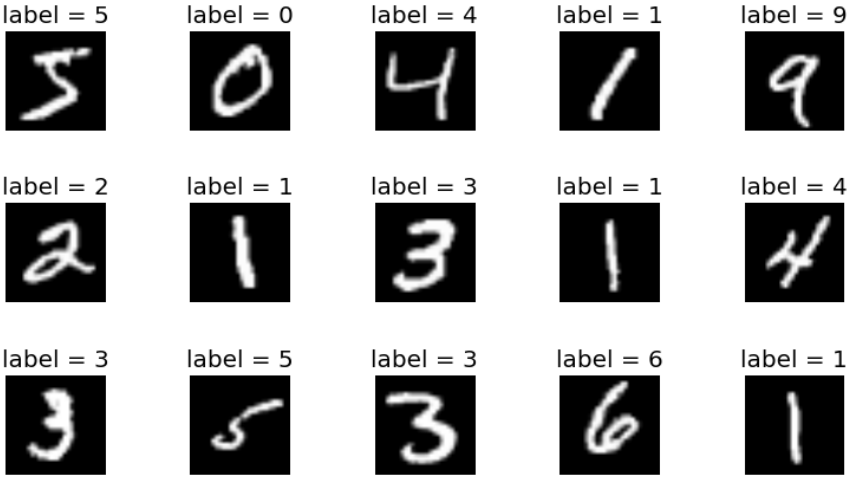

Abbildung 3.: Beispielbilder, vgl. [34] aus der öffentlichen MNIST-Datenbank. Zu sehen sind handgeschriebene Ziffern und zugehörige Annotationen.

Es wird ein CNN mit zwei Faltungsschichten C^1 und C^2 mit der logistischen Aktivierungsfunktion ψ, auch *sigmoid* genannt, 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 [34], ist in Abbildung 4 dargestellt. Dabei sind

- X ein Grauwert-Bild der Größe $h = b = 28$,
- $K^1 \in \mathbb{R}^{6\times 1\times 5\times 5}$ der Kern der Schicht C^1,
- $b^1 \in \mathbb{R}^6$ der Biasvektor der Schicht C^1,
- $K^2 \in \mathbb{R}^{12\times 6\times 5\times 5}$ der Kern der Schicht C^2,
- $b^2 \in \mathbb{R}^{12}$ der Biasvektor der Schicht C^2,

- $W \in \mathbb{R}^{192\times 10}$ die Gewichtsmatrix der Neuronenschicht,
- $b \in \mathbb{R}^{10}$ der Biasvektor der Neuronenschicht,
- $y \in \mathbb{R}^{10}$ die Ausgabe des CNN.

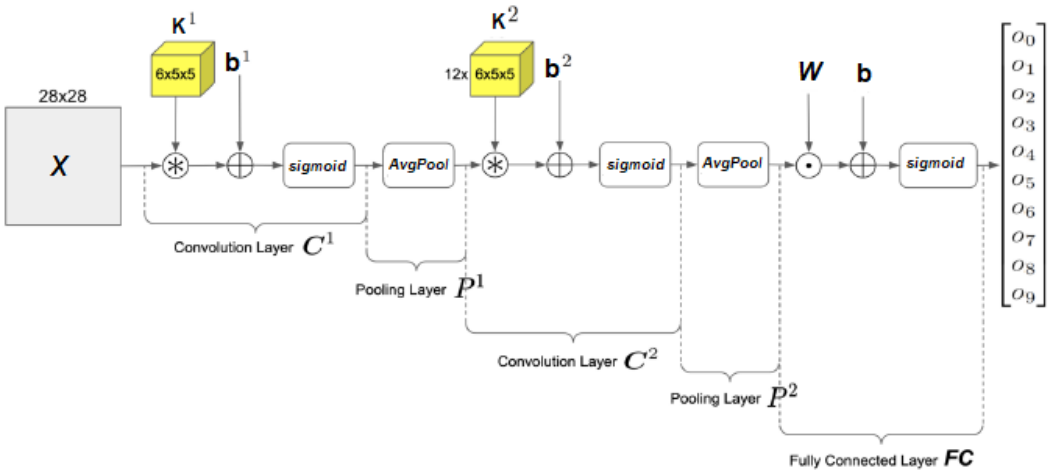

Abbildung 4.: Das CNN-Modell zur Klassifikation von handgeschriebenen Ziffern ist dargestellt. Die Abbildung wurde [47] entnommenen und angepasst.

Im Folgenden wird der Online-Backpropagationsalgorithmus 5 verwendet, um das CNN zu trainieren. Dieser besteht zunächst aus der Initialisierung und der Vorwärtsrechnung. Zur Übersicht wird im Folgenden eine komponentenweise Notation genutzt, die sich an Zhang [69] orientiert.

Vorwärtsrechnung

Alle Biasvektoren werden als Nullvektor initialisiert. Die Initialisierung der Gewichte wird ähnlich der Xavier-Initialisierung [21] vorgenommen, also

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

Die Parameteranzahl ist damit

$$\underbrace{(5\cdot 5+1)\cdot 6}_{\text{Gewichte in } C^1} + \underbrace{(5\cdot 5\cdot 6+1)\cdot 12}_{\text{Gewichte in } C^2} + \underbrace{(192+1\cdot 10)}_{\text{Gewichte im FFN}} = 3898.$$

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

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

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 werden also nur die Teilergebnisse genutzt, bei denen die Kerne vollständig in der Eingabekarte liegen. Diese Methode wird gültige Faltung, engl. *valid convolution*, genannt. Dann gilt $C^1 \in \mathbb{R}^{6\times 24\times 24}$. Anschließend folgen das Mittelwert-Pooling P^1 mit

$$P_p^1(i,j) = \frac{1}{4}\sum_{u=0}^{1}\sum_{v=0}^{1} C_p^1(2i-u, 2j-v),\ 1 \leq i,j \leq 12$$

für $1 \leq p \leq 6$. Es gilt $P^1 \in \mathbb{R}^{6\times 12\times 12}$. Es folgt die zweite Faltungsschicht C^2, welche zwölf Merkmalskarten

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

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

$$P_q^2(i,j) = \frac{1}{4}\sum_{u=0}^{1}\sum_{v=0}^{1} C_p^1(2i-u, 2j-v),\ 1 \leq i,j \leq 4$$

für $1 \leq q \leq 12$. Dann gilt $P^2 \in \mathbb{R}^{12\times 4\times 4}$. Diese zwölf 4×4- Matrizen werden durch die Flatten-Funktion T_f aus Definition 29 in einen Vektor $f \in \mathbb{R}^{12\cdot 4\cdot 4}$ umgewandelt. Dies sei durch

$$f = T_f(P^2)$$

beschrieben. Die Umkehrung wird mit

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

bezeichnet. Die Ausgabe des FFN ergibt sich schließlich zu

$$y = \psi(W^T f + b).$$

und der mittlere quadratische Fehler lautet

$$E = \frac{1}{2} \sum_{k=1}^{10} (y(k) - t(k))^2 ,$$

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

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

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

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

darstellen. Analog gilt

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

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

$$\begin{aligned}
\delta^f(j) &= \frac{\partial E}{\partial f} \\
&= \sum_{k=1}^{10} \frac{\partial E}{\partial y(k)} \cdot \frac{\partial y(k)}{\partial f(j)} \\
&= (y(k) - t(k)) \cdot \frac{\partial}{\partial f(j)} \psi \left(\sum_{j=1}^{192} W(k,j) f(j) + b(k) \right) \\
&= -\sum_{k=1}^{10} (y(k) - t(k)) \cdot y(k)(1 - y(k)) \cdot W(k,j) \\
&= \sum_{k=1}^{10} \delta^{\mathrm{FFN}}(k) \cdot W(k,j) \\
\Rightarrow \delta^f &= W \delta^{\mathrm{FFN}}.
\end{aligned} \tag{4.9}$$

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 T und $\nabla T = \frac{1}{4}\mathbf{1}$ das Upsampling

$$\delta_q^{C^2}(i,j) = \delta_q^{P^2}(\lceil i/2 \rceil, \lceil j/2 \rceil) \cdot \frac{1}{4}, \;\; 1 \leq i,j \leq, 1 \leq q \leq 12.$$

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

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

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

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

den lokalen Fehler $\delta^{C^2}_{q,\psi}$ der Faltungsschicht C^2. Damit ergibt sich wegen der Ableitung der logistischen Funktion

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

Es gilt

$$\begin{aligned} \Delta K^2_{q,p}(u,v) &= \sum_{i=1}^{8}\sum_{j=1}^{8} P^1_p(u+i, v+j)\cdot \delta^{C^2}_{q,\psi}(i,j) \\ \Rightarrow \Delta K^2_{q,p} &= P^1_p * \delta^{C^2}_{q,\psi}, \quad 1 \le q \le 12, 1 \le p \le 6. \end{aligned} \tag{4.10}$$

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 performant implementiert werden, da sie sowohl bei der Vorwärtsrechnung als auch Rückwärtsrechnung genutzt wird. Analog ergibt sich

$$\Delta b^2_q = \sum_{i=1}^{8}\sum_{j=1}^{8} \delta^{C^2}_{q,\psi}(i,j), \quad 1 \le q \le 12.$$

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

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

für alle $1 \le p \le 6$. Mit $K^2_{q,p,rot180}(-u,-v) := K^2_{q,p}(u,v)$ ergibt sich

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

Es gilt $\delta^{P^1} \in \mathbb{R}^{6\times 12\times 12}$. Mit dem Upsampling

$$\delta_p^{C^1}(i,j) = \delta_p^{P^1}\left(\lceil i/2\rceil, \lceil j/2\rceil\right) \cdot \frac{1}{4}, \quad 1 \le i,j \le 24$$

für $1 \le p \le 6$ kann nun der Gradient $\Delta K_{p,1}^1$ an einer Stelle (u,v) mit

$$\begin{aligned}
\Delta K_{p,1}^1(u,v) &= \frac{\partial E}{\partial K_{p,1}^1(u,v)} \\
&= \sum_{i=1}^{24}\sum_{j=1}^{24} \frac{\partial E}{\partial C_p^1(i,j)} \cdot \frac{\partial C_p^1(i,j)}{\partial K_{p,1}^1(u,v)} \\
&= \sum_{i=1}^{24}\sum_{j=1}^{24} \delta_p^{C^1}(i,j) \cdot \frac{\partial}{K_{p,1}^1(u,v)} \psi\left(\sum_{u=-2}^{2}\sum_{v=-2}^{2} X(i+u,j+v)\cdot K_{p,1}^1(u,v) + b_p^1\right) \\
&= \sum_{i=1}^{24}\sum_{j=1}^{24} \delta_p^{C^1}(i,j) \cdot C_p^1(i,j)\left(1 - C_p^1(i,j)\right) \cdot X(i+u,j+v).
\end{aligned}$$

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

$$\begin{aligned}
\delta_{p,\psi}^{C^1}(i,j) :&= \delta_p^{C^1}(i,j) \cdot C_p^1(i,j)\left(1 - C_p^1(i,j)\right) \\
&= \delta_p^{C^1}(i,j) \cdot \psi'(C_p^1(i,j)), \quad 1 \le i,j \le 8, 1 \le p \le 6.
\end{aligned}$$

Dann gilt schließlich

$$\begin{aligned}
\Delta K_{p,1}^1(u,v) &= \sum_{i=1}^{24}\sum_{j=1}^{24} X(u+i,v+j)\cdot \delta_{p,\psi}^{C^1}(i,j) \\
\Rightarrow \Delta K_{p,1}^1 &= X * \delta_{p,\psi}^{C^1}, \quad 1 \le p \le 6
\end{aligned}$$

und

$$\Delta b_p^1 = \sum_{i=1}^{24}\sum_{j=1}^{24} \delta_{p,\psi}^{C^1}(i,j), \quad 1 \le p \le 6.$$

Die Aktualisierung der Parameter mit der Lernrate $\lambda \in \mathbb{R}$ efolgt durch

$$\begin{aligned}
K_{p,1}^1 &= K_{p,1}^1 + \lambda \Delta K_{p,1}^1 \\
b_p^1 &= b_p^1 \quad + \lambda \Delta b_p^1 \\
K_{q,p}^2 &= K_{q,p}^2 + \lambda \Delta K_{q,p}^2 \\
b_q^2 &= b_q^2 \quad + \lambda \Delta b_q^2 \\
W &= W \quad + \lambda \Delta W \\
b &= b \quad + \lambda \Delta b.
\end{aligned}$$

Diese Schritte sind solange zu wiederholen, bis die gewünschte Erkennungsrate bzw. Generalisierungsrate erreicht wird. Für die effiziente Implementierung des Trainingsprozesses sind die Darstellungen als Matrixvektorprodukte, z.B. in Gleichung (4.9), beziehungsweise Faltungsoperationen, vgl. Gleichung (4.10), zu nutzen. Diese Operationen sind performant umzusetzen.

In diesem Kapitel wird die Backpropagation bei gefalteten neuronalen Netzen beschrieben. Bei der Darstellung des Trainingsprozesses wird darauf Wert gelegt, die Algorithmen möglichst allgemein zu beschreiben, um die meisten verwendeten Architekturen abzudecken. Es können also beliebig viele Faltungsschichten, Pooling-Schichten und Neuronenschichten untereinander verknüpft werden. Im folgenden Kapitel 5 wird ausgehend von optimierten Modellparametern eines CNN-Modells die datenbankgestützte Umsetzung in SQL diskutiert. Die Problemstellungen 1 und 3 werden bearbeitet.

5. Datenbankgestützte Implementierung von CNN

In diesem Kapitel werden Ideen zur Umsetzung der datenbankgestützten Mustererkennung durch trainierte 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 Definition 25, in SQL beleuchtet. Dabei stellt sich heraus, dass es sich lohnt, die diskrete Faltung mit Operationen der linearen Algebra mithilfe der in Kapitel 2 vorgestellten Basisoperationen umzusetzen. So gelingt es, die Problemstellung 1 zu lösen und auch für relativ große Grauwertbilder akzeptable Laufzeiten zu erreichen, die jedoch in Zukunft noch verbessert werden müssen.

Im Abschnitt 5.2 wird hinsichtlich Problem 3 eine (objekt-) relationale Umsetzung der Vorwärtsrechnung für das Modell 4 zur Ziffernerkennung vorgestellt. Dieses Modell sei mithilfe des Backpropagationsalgorithmus 5 bereits trainiert, siehe Abschnitt 4.4.

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 gedrehten 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 jedes 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 Umgebungen 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 jedes Pixel von X zu berechnen. In SQL kann dies mithilfe des kartesischen Produkts implementiert werden. Dazu bezeichnen **X** und **K** die Relation zur Darstellung der Matrizen $X \in \mathbb{R}^{h\times b}$ und $K \in \mathbb{R}^{k\times k}$ im Coordinate-Schema. Eine rein relationale Umsetzung von Gleichung (5.2) ist durch die SQL-Anfrage 5.1 gegeben. Zur Übersicht sind die SQL-Operationen blau gekennzeichnet. Als Datenbankmanagementsystem wird PostgreSQL mit der voreingestellten Parameterkonfiguration genutzt. Bestimmte Parameter sind der Tabelle 8 zu entnehmen. Für eine detailliertere Beschreibung dieser Parameter und deren Bedeutung sei auf die PostgreSQL-Dokumentation verwiesen.

Parameter	Value
shared_buffers	128MB
temp_buffers	8MB
effective_cache_size	4GB
work_mem	4MB
maintenance_work_mem	64MB
min_wal_size	80MB
max_wal_size	1GB
wal_buffers	4MB

Tabelle 8.: Es ist die verwendete Parameterkonfiguration von PostgreSQL 14 dargestellt.

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

```
1 SELECT kreuz.i AS i,
2        kreuz.j  AS j,
3        SUM(kreuz.vstrich * K.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
15 JOIN K ON K.i = kreuz.istrich − kreuz.i + (l+1)
16 AND K.j = kreuz.jstrich − kreuz.j + (l+1)
17 GROUP BY kreuz.i,
18          kreuz.j
```

In der Anfrage (Zeile 4-14) werden die Nachbarschaften aller Pixel in der temporären Relation **kreuz** 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 Indizes der Matrix K 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 5, bei der die Laufzeiten der SQL-Anfrage 5.1 in Abhängigkeit von der Dimension n für allgemeine Matrizen $X \in \mathbb{R}^{n \times n}$ dargestellt ist.

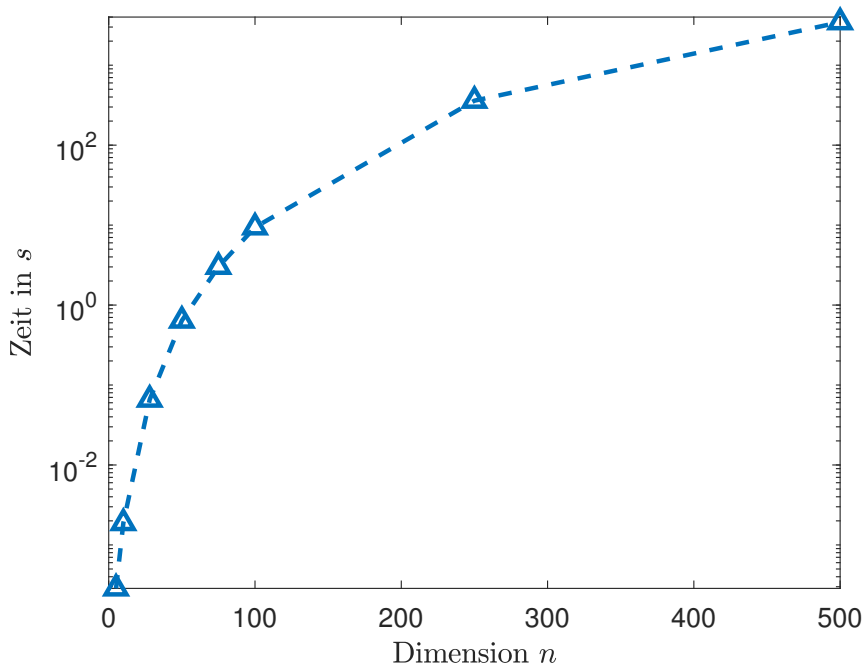

Abbildung 5.: Es sind die Laufzeiten der SQL-Anfrage 5.1 abhängig von der Größe der Matrix $X \in \mathbb{R}^{n \times n}$ in Sekunden dargestellt. Dabei wurde PostgreSQL mit der Konfiguration in Tabelle 8 genutzt.

Eine Verminderung 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 Vorwärtsrechnung 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 jedes Pixel (i, j) die Nachbarpixel $(i', j') \in U$ mit einer entsprechenden Nummer id zur Identifikation hinterlegt. So wird die zeitintensive Berechnung der Nachbarschaften mit dem kartesischen Produkt in der Anfrage 5.1 umgangen. Die Laufzeiten der verbesserten Anfrage 5.2 sind in Abhängigkeit von der Dimension n in Abbildung 6 dargestellt. Dabei ist der erhöhte Speicheraufwand für die Umsetzungstabellen für einen Kern $K \in \mathbb{R}^{3\times3}$ sichtbar. Kerne mit diesen Abmessungen werden oft in der Praxis verwendet.

Listing 5.2: SQL-Code zur Umsetzung der Faltung mit Umsetzungstabellen

```
1  SELECT temp.i,
2         temp.j,
3         SUM(temp.v*K.v)
4  FROM
5    (SELECT U.i AS i,
6            U.j  AS j,
7            I.v AS v,
8            U.istrich  AS istrich,
9            U.jstrich  AS jstrich
10     FROM X
11     JOIN U ON X.i=U.istrich
12     AND X.j =U.jstrich) temp
13 JOIN K ON K.i=temp.istrich-temp.i+(l+1)
14 AND F.j=temp.jstrich-temp.j+(l+1)
15 GROUP BY temp.i,
16          temp.j
```

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 zyklische Faltung dieser eindimensionalen Signale wird im Folgenden erklärt.

Definition 32 (Zyklische Faltung)**.** *Die zyklische Faltung* $y = f \circledast g \in \mathbb{R}^n$ *zweier*

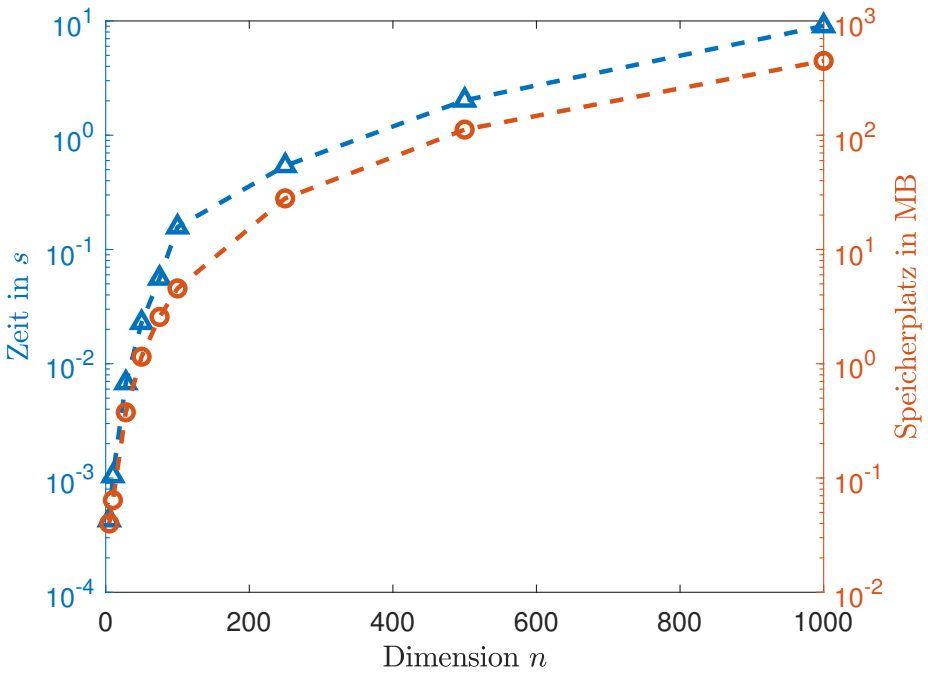

Abbildung 6.: Es ist der Zeit- und Speicheraufwand der SQL-Anfrage 5.2 in Abhängigkeit von der Größe der Matrix $X \in \mathbb{R}^{n \times n}$ dargestellt. Die blaue Kurve spiegelt die Laufzeiten in Sekunden wider. Außerdem ist der benötigte Speicherbedarf der Umsetzungstabellen für die Nachbarschaften in PostgreSQL für einen Kern $K \in \mathbb{R}^{3 \times 3}$ in MB abgebildet.

zeitdiskreter Signale $f \in \mathbb{R}^n$ *und* $g \in \mathbb{R}^n$ *ist durch*

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

definiert. Dabei werden Indizes außerhalb von $0, \ldots, n-1$ *durch Modulo-Rechnung* (mod n) *in den gültigen Indexbereich abgebildet.*

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

Definition 33 (Toeplitz-Matrix, Zyklische Matrix)**.** *Eine diagonalkonstante Matrix* $A \in \mathbb{R}^{n \times n}$ *der Gestalt*

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

wird Toeplitz[1]-Matrix genannt. Es gilt

$$A_{i,j} = A_{i+1,j+1} = a_{i-j} \tag{5.3}$$

für alle Indizes $0 \leq i, j \leq n-1$*. 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* $0 \leq i \leq n-1$ *wird* A *zyklische Matrix genannt. Zyklische Matrizen sind damit eindeutig durch einen Vektor* $a \in \mathbb{R}^n$ *charakterisiert.*

Für zeitdiskrete Signale $f, g \in \mathbb{R}^n$ kann die zyklische Faltung $y = f \circledast g$ als Matrixvektorprodukt mit einer zyklischen Matrix dargestellt werden. Es gilt

$$y = \underbrace{\begin{pmatrix} g_0 & g_{n-1} & g_{n-2} & \cdots & g_1 \\ g_1 & g_0 & g_{n-1} & \cdots & g_2 \\ g_2 & g_1 & g_0 & \ddots & g_3 \\ & \ddots & \ddots & \ddots & \\ g_{n-1} & g_{n-2} & g_{n-3} & \cdots & g_0 \end{pmatrix}}_{=:\mathrm{zyk}(g)} \begin{pmatrix} f_0 \\ f_1 \\ f_2 \\ \vdots \\ \vdots \\ f_{n-1} \end{pmatrix}$$

Die Faltung zweidimensionaler Signale kann mithilfe von doppelt zyklischen Blockmatrizen dargestellt werden.

Definition 34 (Doppelt zyklische Blockmatrix)**.** *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 Definition 33 sind. Ist zusätzlich* A *eine zyklische Matrix, so wird* A *doppelt zyklische Blockmatrix genannt.*

Die Konstruktion solcher zyklischen Blockmatrizen soll im Folgenden beleuchtet werden. Dazu seien die Matrizen $X \in \mathbb{R}^{n \times n}$ und $K \in \mathbb{R}^{k \times k}$ gegeben. Zunächst wird der Kern K in eine $n \times n$-Matrix eingebettet, die wieder mit $K \in \mathbb{R}^{n \times n}$ bezeichnet wird. Dazu wird der Kern von unten und von rechts mit Nullen aufgefüllt, siehe Beispiel 3. Weiter bezeichne $\mathrm{vec}(X)$ 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, vgl. Definition 29. Das folgende Lemma liefert die Darstellung der Matrixfaltung, siehe Bemerkung 2, als Matrixvektorprodukt mit dünnbesetzter Matrix.

Lemma 4 (vgl. Jain [29], [57])**.** *Für einen gedrehten Kern* $K \in \mathbb{R}^{n \times n}$ *wird die zyklische Blockmatrix* $A \in \mathbb{R}^{n^2 \times n^2}$ *als*

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

[1] Otto Toeplitz 1881-1940

konstruiert. Sei die zweidimensionale Faltung von X und K als

$$Y_{i,j} = \sum_{u=0}^{n-1} \sum_{v=0}^{n-1} X_{i+u,j+v} K_{u,v}$$

mit $X_{i,j} = 0$ für $i \notin \{0, \ldots, n-1\}$ und $j \notin \{0, \ldots, n-1\}$ gegeben. Dann gilt der Zusammenhang $\mathrm{vec}(Y) = A\,\mathrm{vec}(X)$. *Die Matrix A ist dünnbesetzt.*

Beweis. Ein Beweis ist von Jain [29] gegeben. Das Auffüllen von $K \in \mathbb{R}^{k \times k}$ zu $K \in \mathbb{R}^{n \times n}$ mit Nullen führt zur dünnbesetzten Struktur der Matrix A. □

Beispiel 3. *Seien die Matrizen*

$$X = \begin{pmatrix} x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 \\ x_7 & x_8 & x_9 \end{pmatrix}, \quad K = \begin{pmatrix} k_1 & k_2 & 0 \\ k_3 & k_4 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

gegeben. Mit

$$A\,\mathrm{vec}(X) = \begin{pmatrix} k_1 & k_2 & 0 & k_3 & k_4 & 0 & 0 & 0 & 0 \\ 0 & k_1 & k_2 & 0 & k_3 & k_4 & 0 & 0 & 0 \\ 0 & 0 & 0 & k_1 & k_2 & 0 & k_3 & k_4 & 0 \\ 0 & 0 & 0 & 0 & k_1 & k_2 & 0 & k_3 & k_4 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \end{pmatrix}$$

$$= \begin{pmatrix} k_1x_1 + k_2x_2 + k_3x_4 + k_4x_5 \\ k_1x_2 + k_2x_3 + k_3x_5 + k_4x_6 \\ k_1x_4 + k_2x_5 + k_3x_7 + k_4x_8 \\ k_1x_5 + k_2x_6 + k_3x_8 + k_4x_9 \end{pmatrix} = \mathrm{vec}\,(Y)$$

kann die Matrixfaltung $Y = X * K$ ohne zero padding, welche im Modell 4 benötigt wird, berechnet werden. Dazu muss lediglich der Vektor $\mathrm{vec}(X)$ *in eine entsprechende Matrix transformiert werden.*

Im Beispiel 3 ist zu beachten, dass A keine doppelt zyklische Blockmatrix ist, da sich die Abmessungen der Ergebnismatrix $Y \in \mathbb{R}^{2 \times 2}$ mit der gültigen Faltung verkleinern. Es fehlen also gewisse Zeilen in A, sodass A nicht zyklisch ist. Die genaue Konstruktion doppelt zyklischer Blockmatrizen für die Faltungsoperation mit bzw. ohne zero padding ist ein technisches Detail und wird in dieser Arbeit nicht weiter beleuchtet. Für eine mögliche Implementierung sei auf den MATLAB-Code A.1 im Anhang A.2verwiesen.

Die datenbankgestützte Umsetzung der Matrixvektormultiplikation mit dünnbesetzter Matrix wurde bereits im Abschnitt 2.2.4 erläutert. Bei CNN lassen sich die dünnbesetzten zyklischen Blockmatrizen für die Kerne bereits vor der Vorwärtsrechnung in Relationen speichern. Seien $X \in \mathbb{R}^{n \times n}$ und $K \in \mathbb{R}^{k \times k}$ sowie $\mathbf{vecX}$ und $\mathbf{K}$ entsprechende

Relationen des Vektors vec(X) bzw. der eingebetteten $n \times n$-Matrix K im Coordinate-Schema. Die Matrixfaltung $Y = X * K$ lässt sich mit den obigen Resultaten in der SQL-Anfrage 5.3 berechnen. In der Abbildung 7 sind numerische Resultate hinsichtlich des Speicher- und Zeitaufwands der SQL-Anfrage 5.3 für einen Kern $K \in \mathbb{R}^{3\times3}$ dargestellt.

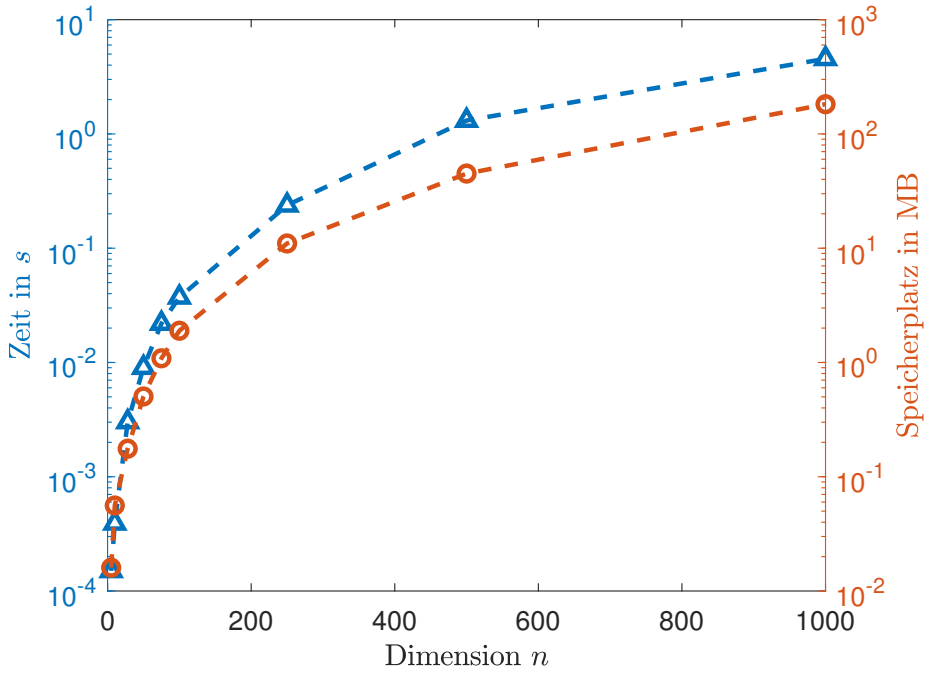

Abbildung 7.: Es ist der Zeit- und Speicheraufwand der SQL-Anfrage 5.3 in Abhängigkeit von der Größe der Matrix $X \in \mathbb{R}^{n\times n}$ dargestellt. Die blaue Kurve spiegelt die Laufzeiten in Sekunden wider. Außerdem ist der benötigte Speicherbedarf der dünnbesetzten Matrizen im Spaltenkompression-Schema für einen Kern $K \in \mathbb{R}^{3\times3}$ in MB abgebildet.

Listing 5.3: SQL-Code zur Umsetzung der Matrixfaltung als Matrixvektorprodukt

```
1 SELECT temp.i AS i,
2        SUM(v) AS v
3 FROM
4   (SELECT UNNEST(K.i) AS i,
5           UNNEST(K.v)*vecX.v AS v
6    FROM K
7    JOIN vecX ON K.j=vecX.i) temp
8 GROUP BY i
```

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

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

für $0 \leq k \leq n-1$ definiert. Das komplexe Signal $\hat{x}$ ist dem Frequenzbereich zugeordnet.

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.

Definition 36 (Inverse diskrete Fourier-Transformation)**.** *Sei $\hat{x} = (\hat{x}_0, \ldots, \hat{x}_{n-1}) \in \mathbb{C}^n$. Mit den Koeffizienten*

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

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

Die diskrete Fourier-Transformation aus Definition 35 lässt sich in ein Matrixvektorprodukt $\hat{x} = Fx$ überführen, wobei $F \in \mathbb{C}^{n \times n}$ eine symmetrische Matrix der Gestalt

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

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 5. *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 $x = \frac{1}{n} \bar{F} \hat{x}$.*

[2]Jean Baptiste Joseph Fourier 1786-1830

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 $\hat{x} = Fx \Leftrightarrow \frac{1}{n}\bar{F}\hat{x} = x$. □

Die inverse diskrete Fourier-Transformation lässt sich mithilfe der diskreten Fourier-Transformation berechnen. Dieser Zusammenhang wird insbesondere für die spätere datenbankgestützte Implementierung der Fourier-Transformationen genutzt.

Lemma 6. *Sei das Fourier-Paar*

$$\begin{aligned} \mathrm{DFT}(x): &\quad \hat{x}_k = \sum_{j=0}^{n-1} \mathrm{e}^{-\frac{2\pi i}{n}jk} \cdot x_j, \\ \mathrm{iDFT}(\hat{x}): &\quad x_j = \frac{1}{n}\sum_{k=0}^{n-1} \mathrm{e}^{\frac{2\pi i}{n}jk} \cdot \hat{x}_k \end{aligned}$$

für $x = (x_0, \ldots, x_{n-1})^T \in \mathbb{R}^n$ gegeben. Dann gilt $x = \frac{1}{n}(DFT(\hat{x}^*))^*$. Hierbei ist mit * die komplexe Konjugation gemeint.

Beweis. Für alle $0 \leq j \leq n-1$ gilt

$$\begin{aligned} x_j^* &= \frac{1}{n}\sum_{k=0}^{n-1} \mathrm{e}^{-\frac{2\pi i}{n}jk} \cdot \hat{x}_k^* \\ &= \frac{1}{n}\mathrm{DFT}(\hat{x}^*)_j. \end{aligned}$$

Die Konjugation auf beiden Seiten liefert die Aussage. □

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 37. *Die zweidimensionale diskrete Fourier-Transformation für $X \in \mathbb{R}^{n\times n}$, kurz $\hat{X} = \mathrm{2DFT}(X)$, ist als*

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

definiert.

Die zweidimensionale diskrete Fourier-Transformation ist als Hintereinanderausführung von zwei eindimensionalen Fourier-Transformationen, vgl. Definition 35, 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 Gleichung (5.4) darstellen.

Zwischen der zyklischen Faltung aus Definition 32 und der diskreten Fourier-Transformation wie in Definition 35 besteht ein fundamentaler Zusammenhang, und zwar das Faltungstheorem. Eine Version davon wird im weiteren Verlauf dieser Arbeit genutzt, um die Matrixfaltung, vgl. Definition 25, mithilfe von Fourier-Transformationen zu berechnen.

Satz 3 (Zyklisches Faltungstheorem)**.** *Seien Vektoren $f, g \in \mathbb{R}^n$ gegeben und $y = f \circledast g$ 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 [58] 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 der Kern K gedreht und $l = \lfloor k/2 \rfloor$. Die Matrixfaltung $Y = X * K$ ist durch*

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

erklärt, vgl. Bemerkung 2. Der Kern K wird in eine $n \times n$-Matrix ähnlich wie in Beispiel 3 eingebettet und diese wird wieder mit $K \in \mathbb{R}^{n \times n}$ bezeichnet wird. In der Matrix K werden zusätzlich bestimmte Zeilen zyklisch verschoben, sodass die zyklischen Randbedingungen der Faltung im Zeitbereich eingehalten werden. Stehen die zweidimensionalen diskreten Fourier-Transformationen $\hat{X} = \mathrm{2DFT}(X)$ und $\hat{K} = \mathrm{2DFT}(K)$ zur Verfügung, so gilt mit dem Faltungstheorem, siehe Satz 3, der Zusammenhang

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

Für eine detailliertere Beschreibung der Konstruktion der eingebetteten Matrizen $K \in \mathbb{R}^{n \times n}$ sei aufgrund deren Umfangs auf Jain [29] verwiesen. Eine Implementierungsmöglichkeit A.3 in MATLAB ist im Anhang A.2 gegeben. 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 $\hat{X}$ und $\hat{K}$ für die Eingabe X und den gedrehten Kern 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.

Für Schritt 1 und Schritt 3 kann wegen Lemma 5 und Lemma 6 die Fourier-Matrix F benutzt werden. Die Aufgabe besteht nun darin, die Berechnungen in Algorithmus 6 datenbankgestützt umzusetzen.

Algorithm 6 Matrixfaltung mit diskreten Fourier-Transformationen

Require: Eingabematrix $X \in \mathbb{R}^{n \times n}$, eingebetteter Kern $K \in \mathbb{R}^{n \times n}$, Fourier-Matrix $F \in \mathbb{R}^{n \times n}$
Ensure: Matrixfaltung $Y = X * K$
Berechne 2DFT(X)
$\hat{X} = FXF^T$
$\hat{K} = FKF^T$
Berechne das Produkt mit der elementweisen Multiplikation
$\hat{Y} = \hat{X} \odot \hat{K}$
Bestimme i2DFT(Y)
$Z = \hat{Y}^*$
$\hat{Z} = FZF^T$
$Y = \frac{1}{n^2}\hat{Z}^*$

Dies gelingt, da ausschließlich Basisoperationen wie die Matrixmatrixmultiplikation sowie das Adjungieren von Matrizen benötigt werden. Die elementweise Multiplikation und die Konjugation können als einfache skalare Funktionen implementiert werden. Da die Matrix F nur von den Dimensionen der beteiligten Matrizen abhängt und diese durch die Hyperparameter des verwendeten CNN festgelegt sind, kann F vor der Vorwärtsrechnung in einer Relation gespeichert werden. Die Matrix F besitzt komplexwertige Einträge und daher wird das Attribut **v** im Coordinate-Schema in die Attribute **re** und **im** aufgeteilt, um den Real- und Imaginärteil getrennt zu speichern. Die Multiplikation zweier komplexer Zahlen z_1 und z_2 ist durch

$$\begin{aligned} z_1 \cdot z_2 &= (\Re(z_1) + i\Im(z_1))(\Re(z_2) + i\Im(z_2)) \\ &= (\Re(z_1) \cdot \Re(z_2) - \Im(z_1) \cdot \Im(z_2)) + i(\Re(z_1) \cdot \Im(z_2) + \Im(z_1) \cdot \Re(z_2)) \end{aligned}$$

erklärt. Darüber hinaus können auch die Fourier-Transformierten $\hat{K}$ für alle beteiligten Kerne K bereits vor der Erkennungsphase bestimmt werden. So kann die Berechnungszeit verkürzt werden.

Seien $X \in \mathbb{R}^{n \times n}$ und **X** sowie **F** Relationen zur Darstellung der Matrizen X und F im Coordinate-Schema. Im ersten Berechnungsschritt ist lediglich die Matrix 2DFT(X) zu bestimmen. Die entsprechende SQL-Anfrage 5.4 lässt sich formulieren. Dabei werden *Common-Table-Expressions*, zu Erkennen am Schlagwort **WITH**, welche als temporäre Tabellen zu verstehen sind, verwendet. So ist es möglich, zum einen Zwischenergebnisse übersichtlich darzustellen und zum anderen jene Ergebnisse weiterzuverwenden. Damit gelingt die iterative Berechnung der Matrixmatrixprodukte. Die erste temporäre Tabelle **FT** in den Zeilen 1-6 ermittelt die Matrix F^T. In den Zeilen 7-15 wird die Matrix

XF^T und anschließend in den Zeilen 16-24 das Produkt FXF^T in der temporären Relation **FXFT** berechnet. Schließlich folgt in den letzten Zeilen die Ausgabe der zweidimensionalen diskreten Fourier-Transformation. Numerische Resultate zum Zeit- und Speicheraufwand der Anfrage 5.4 für einen Kern $K \in \mathbb{R}^{3\times 3}$ sind der Abbildung 8 zu entnehmen.

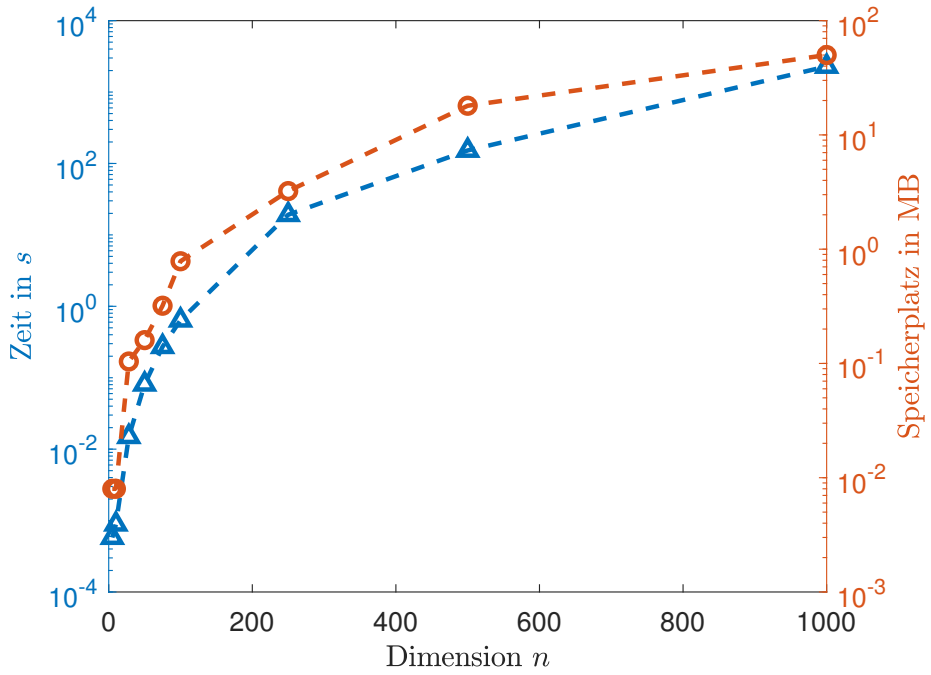

Abbildung 8.: Es ist der Zeit- und Speicheraufwand der SQL-Anfrage 5.4 in Abhängigkeit von der Größe der Matrix $X \in \mathbb{R}^{n\times n}$ dargestellt. Die blaue Kurve spiegelt die Laufzeiten in Sekunden wider. Außerdem ist der benötigte Speicherbedarf der Fourier-Matrizen in PostgreSQL in MB abgebildet.

Listing 5.4: SQL-Code zur Umsetzung der 2DFT

```
1  WITH FT AS
2    (SELECT F.j AS i,
3            F.i  AS j,
4            F.re  AS re,
5            F.im AS im
6     FROM F),
7       XFT AS
8    (SELECT X.i AS i,
9            FT.j AS j,
10           SUM(X.v*FT.re) AS re,
11           SUM(X.v*FT.im) AS im
12    FROM X
13    JOIN FT ON X.j=FT.i
14    GROUP BY X.i,
15             FT.j),
16      FXFT AS
```

```
17   (SELECT F.i AS i,
18           XFT.j AS j,
19           SUM(F.re*XFT.re-F.im*XFT.im) AS re,
20           SUM(F.re*XFT.im+F.im*XFT.re) AS im
21    FROM F
22    JOIN XFT ON F.j=XFT.i
23    GROUP BY F.i,
24             XFT.j)
25 SELECT FXFT.i AS i,
26        FXFT.j AS j,
27        FXFT.re AS re,
28        FXFT.im AS im
29 FROM FXFT
```

Im zweiten Berechnungsschritt ist eine elementweise Multiplikation auszuführen. Angenommen die Matrizen $\hat{X}$ und $\hat{K}$ sind als Relationen **X** und **K** im Coordinate-Schema hinterlegt. Dann wird $\hat{X} \odot \hat{K}$ in der SQL-Anfrage 5.5 implementiert. Diese Berechnung ist im Vergleich zur Faltungsoperation nicht zeitkritisch. Der dritte Berechnungsschritt lässt sich analog zum ersten Schritt darstellen. Dabei ist an zwei Stellen lediglich das Adjungieren einer komplexen Matrix, siehe Anhang A.1, notwendig.

Listing 5.5: SQL-Code zur Umsetzung der elementweisen Multiplikation

```
1 SELECT X.i AS i,
2        X.j AS j,
3        X.re*K.re-X.im*K.im AS re,
4        X.re*K.im+X.im*K.re AS im
5 FROM X
6 JOIN K ON X.i=K.i
7 AND X.j = K.j)
```

Wird zur Berechnung der DFT die sogenannte schnelle Fourier-Transformation (engl. *Fast-Fourier-Transformation*, kurz: FFT) genutzt, können die Zeitkosten von $\mathcal{O}(n^2)$ auf $\mathcal{O}(n \log n)$ vermindert werden. In dieser Arbeit wird ausschließlich die diskrete zweidimensionale Fourier-Transformation behandelt.

5.1.4. Zusammenfassung

In den vorherigen Abschnitten werden drei Ansätze zur datenbankgestützten Umsetzung der Matrixfaltung in SQL beschrieben. Es ist festzuhalten, dass der Nachbarschaft-Ansatz nur mit zugehörigen Umsetzungstabellen für die Umgebungen sinnvoll einsetzbar ist. So wird eine Verminderung der Laufzeit erreicht, aber die benötigten Umsetzungstabellen werden für wachsende Dimensionen schnell größer. Für einen Kern $K \in \mathbb{R}^{3\times3}$ und eine Matrix $X \in \mathbb{R}^{n\times n}$ ergeben sich

$$\underbrace{16}_{\text{Eckpunkte}} + \underbrace{(4 \cdot (n-2))}_{\text{Randpunkte}} \cdot 6 + \underbrace{((n-2)(n-2))}_{\text{innere Punkte}} \cdot 9$$

Tupel in der Umsetzungstabelle.

Der zweite Ansatz nutzt die Darstellung der Faltungsoperation als Matrixvektorprodukt mit dünnbesetzter Matrix. Hinsichtlich der Laufzeiten ist dieser Ansatz am

besten geeignet. Jedoch ist auch hier der Speicherbedarf für die Matrizen groß, da die Dimension n bei der Konstruktion der zyklischen Blockmatrizen quadratisch eingeht.

Im vorherigen Abschnitt 5.1.3 wird die Nutzung der zweidimensionalen diskreten Fourier-Transformation diskutiert. Die datenbankgestützte Umsetzung der zweidimensionalen Fourier-Transformation gelingt mithilfe der Matrixmatrixmultiplikation. Dabei werden Fourier-Matrizen genutzt, die vor der Vorwärtsrechnung in Relationen gespeichert werden können. Mit dem zyklischen Faltungstheorem, vgl. Satz 3, ist die Matrixfaltung mit der 2DFT darstellbar. Hinsichtlich der Laufzeiten schneidet das Verfahren jedoch am schlechtesten ab. Hier könnte die Implementierung der schnellen Fourier-Transformation eine deutliche Verbesserung bewirken. Der Vergleich der Zeit- und Speicherkosten der drei Verfahren zur Berechnung der Matrixfaltung ist in den Abbildungen 9 bzw. 10 dargestellt. Je nach Anwendungsszenario und Rahmenbedingungen ist eine entsprechende Vorgehensweise zu wählen.

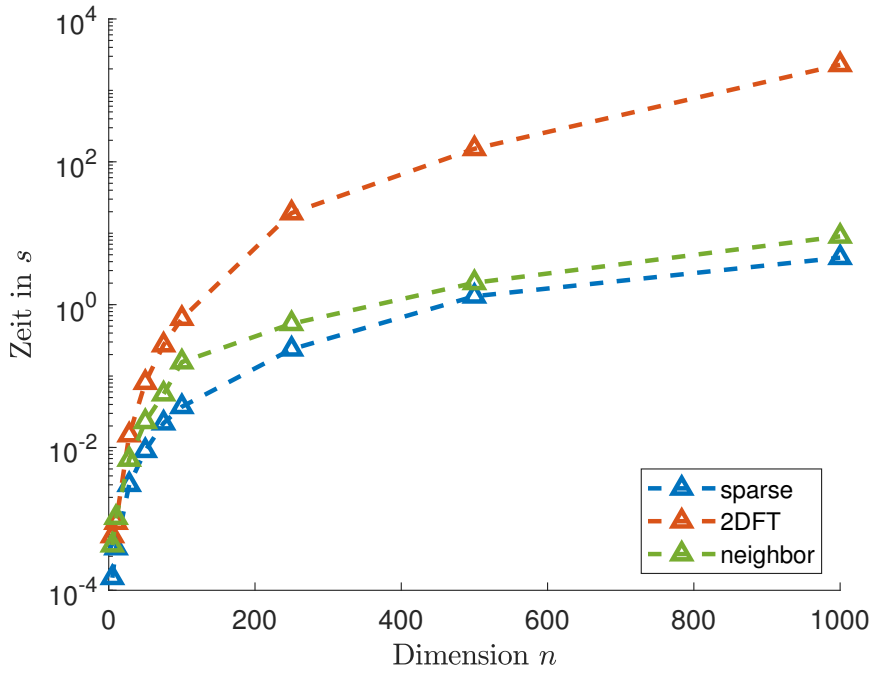

Abbildung 9.: Es sind die Zeitkosten der vorgestellten Verfahren in Abhängigkeit von der Dimension n abgebildet. Die blaue Kurve spiegelt den Matrixvektorprodukt-Ansatz mit dünnbesetzter Struktur, engl. *sparse*, wider. Die rote Kurve beschreibt die Laufzeiten für die Matrixfaltung mithilfe der 2DFT. Mit Grün sind die Laufzeiten des Nachbarschaft-Ansatzes in Sekunden dargestellt.

5.2. Datenbankgestützte Vorwärtsrechnung für CNN

In diesem Abschnitt steht das Problem 3 im Vordergrund und es wird eine relationale Umsetzung der Vorwärtsrechnung für ein trainiertes CNN-Modell diskutiert. In Ab-

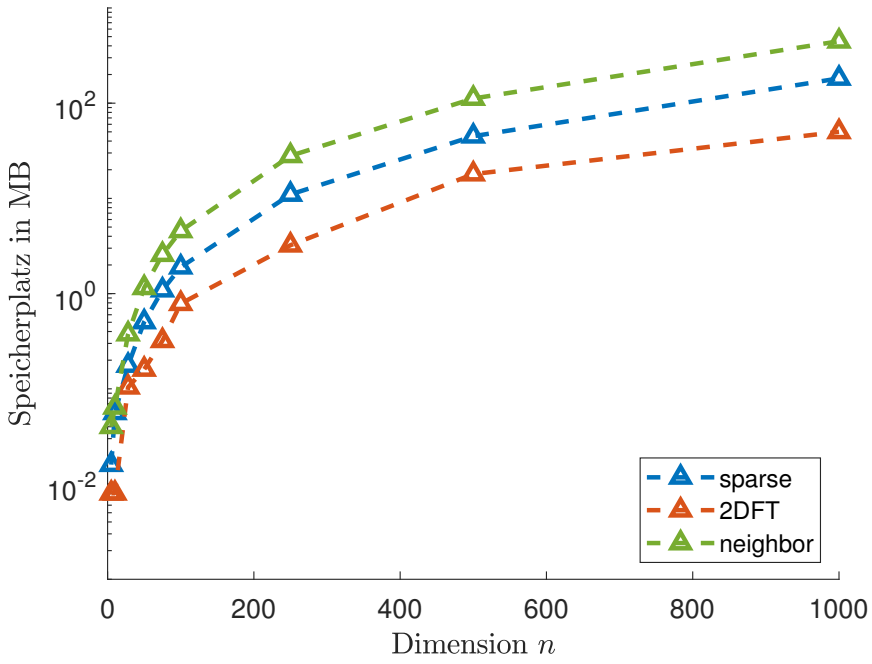

Abbildung 10.: Es ist der zusätzliche Speicherbedarf der vorgestellten Verfahren in Abhängigkeit von der Dimension n abgebildet. Die blaue Kurve spiegelt den Matrixvektorprodukt-Ansatz mit dünnbesetzter Struktur wider. Die rote Kurve beschreibt den Speicherbedarf für die Matrixfaltung mithilfe der 2DFT. Mit Grün ist der Speicherbedarf des Nachbarschaft-Ansatzes in MB dargestellt.

hängigkeit von den Hyperparameter des verwendeten Modells und dem Anwendungsszenario sind die im Abschnitt 5.1 vorgestellten Resultate zu nutzen. Die konkreten Implementierungen in SQL beziehen sich auf das im Abschnitt 4.4 vorgestellte Modell 4, welches als laufendes Beispielmodell dienen soll. In diesem Abschnitt tauchen immer wieder Modellparameter auf, welche als trainiert und damit optimal im Sinne der gestellten Klassifikationsaufgabe vorausgesetzt werden. Nur dann ist eine sinnvolle datenbankgestützte Mustererkennung möglich.

Gefaltete Übertragungsfunktion

Bei ML-Verfahren wird oft mit mehrdimensionalen Arrays gearbeitet. Diese sind problemlos mit dem Relationenmodell vereinbar. Das Coordinate-Schema wird einfach mit entsprechenden Attributen, welche die Achsen darstellen, erweitert. Für einen Kern $K \in \mathbb{R}^{q \times p \times k \times k}$ ergibt sich die Relation $\mathbf{K}$ zu

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

wobei q und p die neuen Achsen präsentieren. In diesem Kontext wird von einem erweiterten Coordinate-Schema gesprochen.

Zur Berechnung der Merkmalskarten durch die gefaltete Übertragungsfunktion, vgl. Definition 26, sind (gewichtete) Summen über Matrixfaltungen zu bestimmen. Um dies kompakt darzustellen, werden rekursive Anfragen benutzt, welche seit 1999 in SQL standardisiert sind. An dieser Stelle sei bemerkt, dass je nach Wahl des Datenbankmanagementsystems diese Anfragen unterstützt beziehungsweise nicht unterstützt werden. Ein Beispiel zur rekursiven Berechnung von

$$\sum_{n=1}^{100} n \tag{5.6}$$

ist in Anfrage 5.6 gegeben.

Listing 5.6: Ein Beispiel einer rekursiven SQL-Anfrage

```
1 WITH RECURSIVE t(n) AS
2   (
3    VALUES (1)
4    UNION ALL SELECT n+1
5    FROM t
6    WHERE n < 100 )
7 SELECT sum(n)
8 FROM t;
```

Eine rekursive Anfrage besteht immer aus einem nicht-rekursiven Term (im Beispiel Zeile 3), der Vereinigung **UNION** bzw. **UNION ALL** und einem rekursiven Term (Zeile 4-6), welcher den Selbstverweis auf die entsprechende Relation beinhaltet. Der **UNION ALL**-Operator entfernt im Gegensatz zum **UNION**-Operator keine Duplikate. Die erste **SELECT**-Klausel definiert die Rekursionsbasis und darf sich nicht auf die zu definierende Relation, im Beispiel **t**, beziehen. Durch diese Anweisung werden zudem die Datentypen der Attribute der zu definierenden Relation festgelegt. In den Zeilen 7 und 8 wird die Summe in Gleichung (5.6) über die Partialsummen in der Relation **t** berechnet.

Bei der gefalteten Übertragungsfunktion, siehe Definition 26, mit einer Aktivierungsfunktion ψ ist eine Summe der Form

$$\psi\left(\sum_{p=1}^{z_{in}} \alpha_{qp} A_p + b_q \mathbf{1}\right) \tag{5.7}$$

zu berechnen. Dabei sind die Matrizen A_p Ergebnisse von Matrixfaltungen, $\alpha_{qp} \in \mathbb{R}$ beliebige Gewichte sowie $b \in \mathbb{R}^{z_{out}}$ ein Biasvektor. Zukünftig werden Aktivierungsfunktionen im SQL-Code mit T bezeichnet. Zur Berechnung von der Summe in Gleichung (5.7) wird eine rekursive SQL-Anfrage 5.7 genutzt. Die entsprechenden Relationen liegen jeweils im erweiterten Coordinate-Schema vor. In der Anfrage gilt für alle Gewichte $\alpha_{qp} = 1$. Die Erweiterung für beliebige Gewichte kann mit einer zusätzlichen Relation für die Gewichtsparameter gelingen.

Listing 5.7: Die rekursive SQL-Anfrage zur Berechnung der gefalteten Übertragungsfunktion

```
1  WITH RECURSIVE Arec(n, q, p, i, j, summe) AS
2    (SELECT 1,
3      A.q AS q,
4           A.p AS p,
5           A.i AS i,
6           A.j AS j,
7           A.v AS v
8     FROM A
9     UNION ALL SELECT n+1,
10         Arec.q,
11                    Arec.p,
12                    Arec.i,
13                    Arec.j,
14                    summe+A.v
15    FROM Arec
16    JOIN A ON Arec.q=A.q
17    AND Arec.p=A.p
18    AND Arec.i=A.i
19    AND Arec.j=A.j
20    WHERE n<zin )
21 SELECT Arec.q AS q,
22        Arec.p AS p,
23        Arec.i AS i,
24        Arec.j AS j,
25        T(Arec.summe+b.v) AS v
26 FROM Arec
27 JOIN B ON Arec.n=B.q
28 WHERE n=zin
```

Die datenbankgestützte Matrixfaltung ist bereits im vorherigen Abschnitt 5.1 beleuchtet worden. Zur Veranschaulichung wird die SQL-Implementierung 5.8 zur Berechnung der sechs Merkmalskarten in der Faltungschicht C^1 des Modells 4 dargestellt. Dabei wurde der im Abschnitt 5.1.1 diskutierte verbesserte Nachbarschafts-Ansatz mit Umgebungstabellen und das erweiterte Coordinate-Schema benutzt. Hier ist im Gegensatz zur Schicht C^2 keine rekursive Anfrage notwendig. Es kommen wieder Common-Table-Expressions zum Einsatz. In den Zeilen 1-19 werden die Matrixfaltungen $X * K_p^1$ mit zero-padding mithilfe einer Umgebungstabelle **U** für $1 \leq p \leq 6$ berechnet. In den Zeilen 20 bis 27 wird nur das Teilergebnis behalten, welches mit der gültigen Faltung bestimmt wurde. Schließlich folgt in den Zeilen 28-34 die Manipulation mit den Schwellwerten b_p und die Berechnung der Aktivierung mit der logistischen Funktion.

Listing 5.8: Die SQL-Anfrage zur Berechnung von C^1

```
WITH C1same AS
  (SELECT K1.p AS p,
          temp.i,
          temp.j,
          SUM(temp.v*K1.v) AS v
   FROM
     (SELECT U.i AS i,
             U.j AS j,
             X.v AS v,
             U.istrich AS istrich,
             U.jstrich AS jstrich
      FROM X
      JOIN U ON X.i=U.istrich
      AND X.j =U.jstrich) temp
   JOIN K1 ON K1.i=temp.istrich-temp.i+2
   AND K1.j=temp.jstrich-temp.j+2
   GROUP BY K1.p,
            temp.i,
            temp.j),
     C1valid AS
  (SELECT C1same.p AS p,
          C1same.i-2 AS i,
          C1same.j-2 AS j,
          C1same.v AS v
   FROM C1same
   WHERE C1same.i BETWEEN 3 AND 26
     AND C1same.j BETWEEN 3 AND 26),
     C1 AS
  (SELECT C1valid.p AS p,
          C1valid.i AS i,
          C1valid.j AS j,
          1/(1+EXP(-(C1valid.v+B1.v))) AS v
   FROM C1valid
   JOIN B1 ON C1valid.p=B1.i)
```

Pooling

Die datenbankgestützte Umsetzung von Pooling-Schichten kann direkt durch die Nutzung von Nachbarschaften wie in Abschnitt 5.1.1 gelingen. Da die Dimensionen aller vorkommenden Merkmalskarten und Kerne eines CNN durch die Wahl der Hyperparameter festgelegt werden, müssen die Nachbarschaften beim Pooling nur einmalig berechnet werden. Des Weiteren werden Funktionen wie **MAX** und **MEAN** im SQL-Standard unterstützt, sodass Maximum-Pooling und Mittelwert-Pooling implementiert werden können.

Eine weitere Möglichkeit besteht darin, das Mittelwert-Pooling mit den Schrittweiten $p := p_h = p_b$ als Faltungsoperation mit dem sogenannten Mittelwert-Kern $\frac{1}{p^2}\mathbf{1} \in \mathbb{R}^{p\times p}$ zu implementieren. An dieser Stelle können Resultate aus dem vorherigen Abschnitt 5.1 genutzt werden, um abzuschätzen, welcher Ansatz für das Anwendungsszenario besser geeignet ist. Schließlich ist auch die Flatten-Funktion, vgl. Definition 29, leicht in SQL

umsetzbar.

Eine Kombination aus Mittelwert-Pooling mit Schrittweite $p = 2$ und der Flatten-Funktion T_f ist für das konkrete Modell 4 in der SQL-Anfrage 5.9 gegeben. Alle beteiligten Relationen liegen entsprechend ihrer Namen im erweitertem Coordinate-Schema vor. Es wird der oben diskutierte Nachbarschaft-Ansatz gewählt.

Listing 5.9: SQL-Code zur Umsetzung des Mittelwert-Poolings und anschließender Flatten-Operation

```
1  WITH P2 AS
2    (SELECT C2.q AS q,
3            U.i AS i,
4            U.j AS j,
5            AVG(C2.v) AS v
6     FROM C2
7     JOIN U ON C2.i=U.istrich
8     AND C2.j =U.jstrich
9     GROUP BY C2.q,
10              U.i,
11              U.j),
12      P2stride AS
13    (SELECT P2.q AS q,
14            ceil(P2.i/2)::INTEGER AS i,
15            ceil(P2.j/2)::INTEGER AS j,
16            P2.v AS v
17     FROM P2
18     WHERE mod(P2.i, 2)=0
19       AND mod(P2.j, 2)=0 ),
20       Flatten AS
21    (SELECT (P2stride.q-1)*(4*4)+(P2stride.i-1)*4+P2stride.j AS i,
22            1 AS j,
23            P2stride.v AS v
24     FROM P2stride)
25  SELECT *
26  FROM Flatten
```

In den Zeilen 1 bis 11 wird das Mittelwert-Pooling mithilfe der Aggregatfunktion **AVG** und der Nachbarschaften in der Relation **U** bestimmt. Dieses Ergebnis wird mit den Schrittweiten $s_h = s_b = 2$, vgl. Bemerkung 3, in den Zeilen 12 bis 19 in der Dimension verkleinert. Dabei werden Funktionen wie das Aufrunden **CEIL** und Modulo-Rechnen **MOD** zur passenden Selektierung der Indizes genutzt. Die Doppelpunkte sorgen dafür, dass die Indizes zu natürlichen Zahlen umgewandelt werden. Im Englischen wird diese Typumwandlung als *type-casting* bezeichnet. Alle verwendeten Operationen werden im SQL-Standard unterstützt. Schließlich wird der Vektor $f = T_f(P^2) \in \mathbb{R}^{192}$ der Flatten-Funktion in den Zeilen 20 bis 26 berechnet.

Vorwärtsgerichtete neuronale Netze

Die letzte Schicht eines CNN besteht meistens aus einem FFN mit einer oder mehreren verdeckten Neuronenschichten. Die Theorie von FFN wird in Kapitel 3 vorgestellt. Dabei wird die Ausgabe eines neuronalen Netzes mittels Vorwärtsrechnung, vgl.

Algorithmus 3, berechnet. Diese lässt sich ausschließlich mit bereits vorgestellten Basisoperationen wie der Matrixvektormultiplikation bzw. Vektoraddition und einfachen Funktionsauswertung darstellen. Wird das Modell 4 zur Klassifikation von Ziffern genutzt, so ist die Vorwärtsrechnung lediglich für eine Neuronenschicht datenbankgestützt umzusetzen. Die Verallgemeinerung für beliebig viele Schichten wurde bereits in einer Projektarbeit der Universität Rostock untersucht [4].

Seien die Gewichtsmatrix $W \in \mathbb{R}^{192\times 10}$ sowie der Biasvektor $b \in \mathbb{R}^{10}$ wie in Abschnitt 4.4 gegeben. Die entsprechenden Relationen **W** und **B** werden gemäß dem Coordinate-Schema erstellt. Weiter sei T eine beliebige Aktivierungsfunktion. Beim Modell 4 ist T die logistische Funktion. Schließlich sei für eine Eingabekarte X in der Relation **FLATTEN** der Vektor $f = T_f(X)$ der Flatten-Schicht gespeichert, welcher als Eingabe des FFN dient. Die Ausgabe des FFN und damit des gesamten CNN lässt sich durch die SQL-Anfrage 5.10 berechnen. In den Zeilen 1-9 wird das Matrixvektorprodukt Wf berechnet, welches anschließend mit dem Biasvektor b manipuliert wird. Schließlich werden die Aktivierungen der Ausgabeschicht mit der Funktion T ermittelt. In den Zeilen 10-15 wird der Index mit der maximalen Aktivierung selektiert, welcher, vermindert um Eins, die Klassifikation der Ziffer darstellt.

Listing 5.10: SQL-Code zur Umsetzung der Vorwärtsrechnung des einschichtigen FFN aus Modell 4

```
1  WITH OUT AS (
2  SELECT B.i AS i,
3         T(B . v + sub . v) AS v
4  FROM
5    (SELECT W. i AS i,
6            SUM(W. v * FLAT. v) AS v
7     FROM W
8     JOIN FLAT ON W . j = FLAT . i
9     GROUP BY W . i) sub
10 JOIN B ON B.i=sub.i
11 SELECT out.i-1 AS label,
12        out.v
13 FROM OUT
14 WHERE v=
15     (SELECT MAX(v)
16      FROM OUT)
```

Die vollständige datenbankgestützte Vorwärtsrechnung für das trainierte Modell 4 kann durch die Kombination der obigen Resultate durchgeführt werden. Für die Klassifikation von 28×28- Grauwertbildern kann wegen der Dimension der Nachbarschaft-Ansatz genutzt werden. Für andere Modelle sind je nach Anwendungsszenario und Wahl der Hyperparameter die entsprechenden Faltungs- und Pooling-Operationen mit den in Abschnitten 5.1 und 5.2 vorgestellten Methoden hinsichtlich deren Zeit-und Speicheraufwand umzusetzen.

6. Zusammenfassung und Ausblick

In dieser Arbeit wird gezeigt, wie die Mustererkennung in neuronalen Netzen und in gefalteten neuronalen Netzen datenbankgestützt implementiert werden kann. Dazu wird die Vorwärtsrechnung der jeweiligen Modelle in eine Folge von SQL-Anfragen übersetzt. Für die datenbankgestützte Umsetzung der Faltungsoperation werden in dieser Arbeit drei Ansätze vorgestellt, die anschließend hinsichtlich ihrer Einsetzbarkeit in Anwendungen diskutiert werden. Es stellt sich als vorteilhaft heraus, Basisoperationen der linearen Algebra mit dem SQL-Kern zu vereinen, um zufriedenstellende Resultate hinsichtlich der Problemstellung 1 zu erhalten. Daher wurden in Kapitel 2 Methoden und Implementierungen vorgestellt, um Objekte der linearen Algebra als Relationen darzustellen und damit verbundene Matrixvektoroperationen, insbesondere die dünnbesetzte Matrixvektormultiplikation, datenbankgestützt umzusetzen.

In der Erkennungsphase werden trainierte Modelle genutzt, um erkannte Muster in einer Zieldatenmenge effizient ableiten zu können. Die Trainingsphase neuronaler Netze wird in dieser Arbeit mithilfe der Backpropagation realisiert. Dieser Algorithmus wird für vorwärtgerichtete neuronale Netze im Kapitel 3 und für gefaltete neuronale Netze im Kapitel 4 verallgemeinert. In der Literatur wird die Rückwärtsrechnung bei CNN meist nur für spezielle Modelle erläutert beziehungsweise auf die Backpropagation bei neuronalen Netzen verwiesen. In dieser Arbeit wird sowohl die Backpropagation bei FFN sowie CNN algorithmisch beschrieben, um dem Leser die Möglichkeit zu geben, selbst eigene Modelle zu trainieren und zu verwenden. Bei der Darstellung der Trainingsprozesse wird darauf Wert gelegt, die Algorithmen möglichst allgemein zu beschreiben, um die meisten verwendeten Architekturen abzudecken. Auch hier ist zu beobachten, dass der Trainingsprozess mit Matrixvektoroperationen bzw. Faltungsoperationen umgesetzt werden kann. Die performante Implementierung dieser Funktionen ist im Hinblick auf Problem 2 zu gewährleisten.

In dieser Arbeit wird das Modell 4 genutzt, um handgeschriebene Ziffern aus dem MNIST-Datensatz zu klassifizieren. Die Vorwärts- und Rückwärtsrechnung dieses Modells ist im Abschnitt 4.4 ausführlich beschrieben. Zusammen mit den Resultaten aus Kapitel 5 gelingt in dieser Arbeit die datenbankgestützte Vorwärtsrechnung für dieses trainierte Modell. Damit wird Problemstellung 3 gelöst. Schließlich soll noch ein Ausblick auf weitere Forschungsthemen gegeben werden.

Datenbankgestützte Trainingsphase

In den Abschnitten 3.3 und 4.3 wird der Online-Backpropagationsalgorithmus erläutert. Dabei müssen bestimmte Abstiegsrichtungen bestimmt werden. Die effiziente Berechnung dieser Abstiegsrichtungen gehört wohl zu den schwersten Aufgaben des Maschi-

nellen Lernens. Die datenbankgestützte Umsetzung der Trainingsphase in (parallelen) Datenbankmanagementsystemen könnte dabei Abhilfe schaffen und ist zu untersuchen.

Schnelle Fourier-Transformation in SQL

Im Abschnitt 5.1 wird die Faltung mithilfe der diskreten Fourier-Transformation beleuchtet und eine datenbankgestützte Implementierung in SQL vorgestellt. Dabei wird die Transformation als Matrixmatrixprodukt FXF^T berechnet, was insbesondere für große Matrizen zu langsamen Laufzeiten führt. Wird zur Berechnung der DFT die sogenannte schnelle Fourier-Transformation genutzt, können die Zeitkosten von $\mathcal{O}(n^2)$ auf $\mathcal{O}(n \log n)$ vermindert werden. Verschiedene Möglichkeiten, die eindimensionale FFT in SQL zu implementieren, werden in Marten et. al.[42] präsentiert. Die Verallgemeinerung für die zweidimensionale FFT und deren Nutzung zur Berechnung von Faltungen ist zu diskutieren.

Darstellung der Kerne

Bei CNN werden trainierbare Kerne genutzt, mit denen die Faltungen durchgeführt werden. Im Abschnitt 5.1 wird ein Ansatz erläutert, bei dem Kerne in Blockmatrizen überführt werden. Diese Matrizen weisen eine gewisse Bandstruktur auf, welche in Relationen überführt werden kann. So sollte es möglich sein, hinsichtlich des Zeit- und Speicheraufwands bessere Resultate zu erzielen. Darüber hinaus sind bestimmte Kerne seperabel [3]. Auch diese Eigenschaft ist in zukünftiger Forschung zu diskutieren.

Systemanpassung

Während der gesamten Arbeit wurden die Laufzeiten von SQL-Anfragen im Datenbankmanagementsystem PostgreSQL gemessen. Die verwendete Parameterkonfiguration von PostgreSQL ist im Abschnitt 5.1 angegeben. Eine Rekonfiguration kann zu besseren Laufzeiten für die in Kapitel 5 vorgeführten SQL-Anfagen führen und ist untersuchen. Schließlich sollten Faltungsoperationen somit schneller berechnet werden können.

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] Lin Bai, Yiming Zhao und Xinming Huang. „A CNN Accelerator on FPGA Using Depthwise Separable Convolution“. In: *IEEE Trans. Circuits Syst. II Express Briefs* 65-II.10 (2018), S. 1415–1419.

[4] Florian Baldauf. *Lineare Algebra in relationalen Datenbanbksystemen: Transformation von Machine-Learning-Methoden in SQL*. Projektarbeit, Universität Rostock. 2016.

[5] 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.

[6] 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.

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

[8] 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.

[9] 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.

[10] 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.

[11] 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.

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

[13] 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.

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

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

[16] 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).

[17] 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.

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

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

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

[21] Xavier Glorot und Yoshua Bengio. „Understanding the difficulty of training deep feedforward neural networks“. In: *AISTATS*. Bd. 9. JMLR Proceedings. JMLR.org, 2010, S. 249–256.

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

[23] Tobias Grüning. „Neural text line extraction in historical documents : a two-stage clustering approach“. Diss. Universität Rostock, 2018.

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

[25] 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.

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

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

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

[29] Anil K Jain. *Fundamentals of digital image processing*. Prentice-Hall, Inc., 1989.

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

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

[32] 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.

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

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

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

[36] 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.

[37] 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.

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

[39] 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.

[40] 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.

[41] 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.

[42] Dennis Marten, Holger Meyer und Andreas Heuer. „Calculating Fourier Transforms in SQL“. In: *ADBIS*. Bd. 11695. Lecture Notes in Computer Science. Springer, 2019, S. 151–166.

[43] 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.

[44] 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.

[45] Tom M Mitchell und Tom M Mitchell. *Machine learning*. Bd. 1. 9. McGraw-hill New York, 1997.

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

[47] Son Nguyen. *A gentle explanation of Backpropagation in Convolutional Neural Network (CNN).* `https://medium.com/@ngocson2vn/a-gentle-explanation-of-backpropagation-in-convolutional-neural-network-cnn-1a70abff508b`. 2020.

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

[49] 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.

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

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

[52] 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`.

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

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

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

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

[57] Hanie Sedghi, Vineet Gupta und Philip M. Long. „The Singular Values of Convolutional Layers". In: *CoRR* abs/1805.10408 (2018).

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

[59] 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.

[60] 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.

[61] 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.

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

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

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

[65] 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.

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

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

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

[69] Zhifei Zhang. *Derivation of Backpropagation in Convolutional Neural Network (CNN)*. `https://medium.com/@ngocson2vn/a-gentle-explanation-of-backpropagation-in-convolutional-neural-network-cnn-1a70abff508b`. 2016.

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.

Sei $A \in \mathbb{C}^{m\times n}$ eine Matrix im erweitertem Coordinate-Schema. Die adjungierte Ma-

trix $A^* \in \mathbb{C}^{n \times m}$ kann durch die SQL-Anfrage

$$\textbf{SELECT } A.j \textbf{ AS } i,\ A.i \textbf{ AS } j,\ A.re \textbf{ AS } re,\ -(A.im) \textbf{ AS } im$$
$$\textbf{FROM } A$$

berechnet werden.

A.2. MATLAB-Implementierungen

In diesem Abschnitt werden zwei MATLAB-Implementierungen gesammelt. Die Implementierungen Code A.1 und Code A.2 können zur Konstruktion der zyklischen Blockmatrizen aus Abschnitt 5.1 genutzt werden. Der Code wurde von Royi Avital im Zuge der digitalen Bildverarbeitung entwickelt. Mehr Informationenen sind im ausführlichen GitHub Repository https://github.com/RoyiAvital/StackExchangeCodes (zuletzt aufgerufen am 17.08.2022) zu finden.

Listing A.1: Code 1 zur Konstruktion zyklischer Blockmatrizen

```
function [ mK,cBlockMtx ] = Create2DKernelConvMtxSparse( mH, numRows, numCols,
    convShape )
CONVOLUTION_SHAPE_FULL = 1;
CONVOLUTION_SHAPE_SAME = 2;
CONVOLUTION_SHAPE_VALID = 3;
numColsKernel = size(mH, 2);
numBlockMtx = numColsKernel;
cBlockMtx = cell(numBlockMtx, 1);
for ii = 1:numBlockMtx
    cBlockMtx{ii} = full(CreateConvMtxSparse(mH(:, ii), numRows, convShape));

end
switch(convShape)
    case(CONVOLUTION_SHAPE_FULL)
        diagIdx = 0;
        numRowsKron = numCols + numColsKernel - 1;
    case(CONVOLUTION_SHAPE_SAME)
        diagIdx = floor(numColsKernel / 2);
        numRowsKron = numCols;
    case(CONVOLUTION_SHAPE_VALID)
        diagIdx = numColsKernel - 1;
        numRowsKron = numCols - numColsKernel + 1;
end
vI = ones(min(numRowsKron, numCols), 1);
mK = kron(spdiags(vI, diagIdx, numRowsKron, numCols), cBlockMtx{1});
for ii = 2:numBlockMtx
    diagIdx = diagIdx - 1;
    mK = mK + kron(spdiags(vI, diagIdx, numRowsKron, numCols), cBlockMtx{ii});
end
end
```

Listing A.2: Code 2 zur Konstruktion zyklischer Blockmatrizen

```
function [ mK ] = CreateConvMtxSparse( vK, numElements, convShape )
CONVOLUTION_SHAPE_FULL = 1;
CONVOLUTION_SHAPE_SAME = 2;
CONVOLUTION_SHAPE_VALID = 3;
kernelLength    = length(vK);
switch(convShape)
    case(CONVOLUTION_SHAPE_FULL)
        rowIdxFirst = 1;
        rowIdxLast  = numElements + kernelLength - 1;
        outputSize  = numElements + kernelLength - 1;
    case(CONVOLUTION_SHAPE_SAME)
        rowIdxFirst = 1 + floor(kernelLength / 2);
        rowIdxLast  = rowIdxFirst + numElements - 1;
        outputSize  = numElements;
    case(CONVOLUTION_SHAPE_VALID)
        rowIdxFirst = kernelLength;
        rowIdxLast  = (numElements + kernelLength - 1) - kernelLength + 1;
        outputSize  = numElements - kernelLength + 1;
end
mtxIdx = 0;
% The sparse matrix constructor ignores valus of zero yet the Row / Column
% indices must be valid indices (Positive integers). Hence 'vI' and 'vJ'
% are initialized to 1 yet for invalid indices 'vV' will be 0 hence it has
% no effect.
vI = ones(numElements * kernelLength, 1);
vJ = ones(numElements * kernelLength, 1);
vV = zeros(numElements * kernelLength, 1);
for jj = 1:numElements
    for ii = 1:kernelLength
        if (( ii + jj - 1 >= rowIdxFirst) && (ii + jj - 1 <= rowIdxLast))
            % Valid otuput matrix row index
            mtxIdx = mtxIdx + 1;
            vI(mtxIdx) = ii + jj - rowIdxFirst;
            vJ(mtxIdx) = jj;
            vV(mtxIdx) = vK(ii);
        end
    end
end
mK = sparse(vI, vJ, vV, outputSize, numElements);
end
```

Die Implementierungen Code A.3 und Code A.4 dienen hinsichtlich der $2DFT$ zur Einbettung der Kerne in entsprechende $n \times n$- Matrizen sowie das Extrahieren dieser im Coordinate-Schema.

Listing A.3: Code zur Einbettung der Kerne in $n \times n$-Matrizen

```
function [ mHC ] = CircularExtension2D( mH, numRows, numCols )

kernelRadiusV = floor(size(mH, 1) / 2); %<! Vertical Radius
kernelRadiusH = floor(size(mH, 2) / 2); %<! Horizontal Radius

mHC = mH;
mHC(numRows, numCols) = 0;
```

```
8  mHC = circshift(mHC, [-kernelRadiusV, -kernelRadiusH]);
9
10
11 end
```

Listing A.4: Code zum Konstruieren und zum Extrahieren des Coordinate-Schemas

```
1  function [ImatFFT2,prod,Conv_mat_my] = write_twiddle_data(I,K,string1)
2
3
4  W=fft(eye(size(I)));
5  n=size(I,1);
6  m=size(I,2);
7  radius=floor(size(K,1)/2);
8
9  %[x,m,fsize]=padarrays(I,K,'valid');
10
11 Ipad=padarray(I,[0 0],0,'both');
12
13 mm=CircularExtension2D(rot90(K,2),size(Ipad,1),size(Ipad,2));
14 W_f=fft(eye(size(mm)));
15
16 ImatFFT2=W*Ipad*W.';
17 FmatFFT2=W_f*mm*W_f.';
18
19
20 prod=ImatFFT2.*FmatFFT2;
21
22 prod=prod'; %(adjungierte transpose+conj)
23
24 Conv_mat_my=1/n^2*(W*prod*W.')';
25 Conv_mat_my=Conv_mat_my(1+radius:n-radius,1+radius:n-radius);
26 %FFT2=ifft2(fft2(Ipad).*fft2(mm),'symmetric');
27 %FFT2=FFT2(1+radius:n-radius,1+radius:n-radius);
28 %FFT=ifft2(fft2(x).*fft2(m));
29 %FFT=FFT(4:27,4:27);
30 matconv=conv2(I,rot90(K,2),'valid');
31 %norm(FFT-matconv,'fro');
32 %norm(FFT2-matconv,'fro');
33 norm(Conv_mat_my-matconv,'fro');
34
35 Filter_sql=zeros(n^2,4);
36 k=1;
37 for i=1:n
38
39     for j=1:n
40
41         Filter_sql(k,:)=[i j real(W_f(i,j)) imag(W_f(i,j))];
42         k=k+1;
43     end
44
45 end
46
47 writematrix(Filter_sql,string1);
```

```
48 end
```

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 19.08.2022

_____
Florian Baldauf