Universität Rostock

Traditio et Innovatio

Titel der Arbeit, bei Bedarf auch zweizeilig

Untertitel der Arbeit, auch mehrzeilig oder ganz weglassen.

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

Masterarbeit am Institut für Mathematik
Mathematisch-Naturwissenschaftliche Fakultät

Zusammenfassung

Platz für eine kurze Zusammenfassung.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

List of Algorithms

Verzeichnis der Listings

Abkürzungsverzeichnis

Symbolverzeichnis

1. Einleitung

!!! eine kurze Einleitung, wird noch verbessert/erweitert, in Kontext CNN einordnen!!!
!!! ist die Einleitung von der Projektarbeit(FeedForwardNetze in SQL)

!!! Literaturnachweise in bibmanager eingügen !!!

Assistenzsysteme stellen Informationen und Hilfestellungen bei bestimmten Produkten bereit, um deren Bedienung zu erleichtern. Heutzutage spielen sie in fast allen Bereichen der Industrie und Forschung eine große Rolle und unterstützen Menschen bei ihren Tätigkeiten [**winner2014handbook**, **kurihata2005rainy**, **omerdic2011design**]. Meist werden solche Systeme intelligent genannt, denn sie sind in der Lage, durch die Verarbeitung von Sensordaten eigenständig auf bestimmte Szenarien zu reagieren beziehungsweise Vorhersagen über zukünftige Situationen zu treffen. Die Analyse solcher Assistenzsysteme ist daher Forschungsgegenstand in den Bereichen der Big Data Analytics und Künstlichen Intelligenz. Sensoren liefern hierbei im Zusammenhang des *Internet of Things* [**xia2012internet**, **wortmann2015internet**] meist große Datenmengen, die geschickt gespeichert und verarbeitet werden sollen.

Das PArADISE-Projekt [**paradise**] des Lerhstuhls für Datenbank- und Informationsysteme der Universität Rostock beschäftigt sich mit dem Designsprozess von Assisenzsystemen mit dem Ziel, die Entwickler von assistiven Systemen zu unterstützen. Durch eine hochparallele Analyse großer Mengen von Sensordaten soll hierbei die datengetriebene Entwicklung von Assisenzsystemen gelingen. In der Anwendung werden jene Systeme genutzt, um Vorhersagen über bestimmter Ereignisse durch Algorithmen (Maschinelles Lernen) zu treffen. Verfahren des überwachten maschinellen Lernens lassen sich typischerweise in 2 Schritten durchführen. In einer Trainingsphase werden mit Hilfe von Trainingsdaten per Annotation 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.

In der Arbeit von Marten [**marten2017machine**] wird am Beispiel eines Meeting Szenarios ein Maschinelles Lernverfahren namens Hidden-Markov-Modelle untersucht. Es wird erläutert, wie die Erkennungsphase eines zuvor trainierten Modells datenbankgestützt in parallelen SQL-Datenbanksystemen realisiert werden kann. Als Ergebnis wird festgehalten, dass die datenbankgestützte Umsetzung von ML-Algorithmen, hier speziell bei Hidden-Markov-Modellen, in bestimmten Szenarien gute Skalierungseigenschaften hinsichtlich der Datenmenge besitzt und verschiedenste Resultate der relationalen Datenbankforschung vorteilhaft genutzt werden können. Dies motiviert Analysen anderer Machine-Learning-Verfahren und deren Transformation in SQL. Diese Arbeit beschäftigt sich mit neuronalen Netzen als weitverbreitete Machine Learning Methode und deren Transformation in SQL-Datenbanksystemen unter Zuhilfenahme von Werkzeugen der linearen Algebra.

Motivation

In diesem Abschnitt wird die Nutzung paralleler Datanbanksysteme zur Trainings- und Erkennungsphase von Aktivitätsmodellen basierend auf Algorithmen des Machine Learnings [**anzai2012pattern**], kurz ML, motiviert. Die vielen verwendeten Sensoren liefern meist hoch frequente Daten und daher fallen in der Lernphase riesige Datenmengen an, die perfomant behandelt werden müssen. Hier sollen Techniken aus Datenbanksystemen eingebunden werden, um ML-Tools sinnvoll zu unterstützen, was sich als spannendes Forschungsgebiet der Informationssysteme herausstellt [**abiteboul2018research**]. Gelingt dies, so können verschiedenste Resultate der relationalen Datenbankforschung vorteilhaft genutzt werden.

- Techniken der parallelen Datenbanksysteme ermöglichen es, SQL Anfragen auf Rechnercluster zu verteilen, um so die Perfomance von ML Algorithmen zu verbessern.

- Konzepte wie *Query Decomposition* [**chirkova2011materialized**] und *Answering Queries using Views* [**afrati2019answering**, **levy1999answering**] werden genutzt, um bestimmte Auswertungen von Daten näher an den Sensoren durchzuführen und damit *Privacy* [**agrawal2000privacy**] Aspekte von Nutzern zu berücksichtigen.

- *Data Provenance* [**heuer2015metis**, **bruder2017konzepte**] 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.

Das Ziel dieser Arbeit ist, den Transformationsprozess von Machine-Learning-Algorithmen in SQL Anfragen zu beleuchten. Besonders interressant ist die Erweiterung von SQL um Konzepte der linearen Algebra, unter anderem motiviert durch Test-of Time-Award-Winner Dan Suciu [**interviewsuciu**] .

Problemstellung

Aufbau der Arbeit

`\input{<DateiName>}`

oder

`\include{<DateiName>}`

einfügt.

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

2. Grundlagen

2.1. Mathe/ ML Learning

2.2. Relationale Datenbanksysteme

2.2.1. Die Anfragesprache SQL

2.2.2. Lineare Algebra in SQL

In diesem Abschnitt wird eine Darstellungsform von Vektoren und Matrizen als Relationen vorgestellt. Weiter werden Ideen zur Umsetzung wichtiger Basisoperationen mit Vektoren und Matrizen in SQL beleuchtet, da diese mathematischen Objekte bei zahlreichen statistischen Analysen eingestzt werden.

2.2.3. Matrixdarstellunng

Im Folgenden wird das *Coordinate scheme* [**martendiss**] als Schema für die Darstellung von Matrizen und Vektoren genutzt. Dieses Schema gestaltet sich als einfach und ist daher weit verbreitet [**saad1990sparskit**]. Für eine weiterführende Diskussion anderer Darstellungsmöglichkeiten, sei an dieser Stelle auf Marten [**martendiss**] verwiesen. Das Coordinate Scheme beinhaltet 3 Arrays, welche den Zeilenindex, Spaltenindex und den Matrixeintrag als jeweilge Nicht-Null Werte strukturieren.

Beispiel 1. Für $x \in \mathbb{R}^n$ und $A \in \mathbb{R}^{n \times m}$ ergeben sich die Relationen

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

für den Vektor x und

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

für die Matrix A. 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 Abbildung 2.1.

Zeile i	1	1	2	2	3	3
Spalte j	1	2	1	2	1	2
Eintrag $a_{i,j}$	1	2	-5	2	0	7

Abbildung 2.1.: Das Coordinate Schema zur Matrix A aus Beispiel 1.

2.2.4. Basisoperationen

In diesem Abschnitt werden typische Operationen mit Objekten der linearen Algebra beschrieben. Einfache Operationen wie Summation und Multiplikation für reelle Zahlen sind bereits im SQL-Standard 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 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

$$\begin{aligned}&\textbf{select } A.i \textbf{ as } i,\ A.j \textbf{ as } j, r * A.v + (s * B.v) \textbf{ as } v\\&\textbf{from } A \textbf{ join } B \textbf{ on } A.i = B.i \textbf{ and } A.j = B.j\end{aligned}$$

Das Auftreten von **NULL**-Werten in den Relationen sei hierbei ausgeschlossen. Mit der Aggregation **SUM** können zudem Skalarprodukte und damit Längenbegriffe wie Normen und dadurch induzierte Abstandsbergiffe formuliert werden. Die entsprechenden SQL-Anfragen sind im Anhang **??** zu finden.
Weitere wichtige Oprationenen stellen die Matrixvektor- und Matrixmatrixmultiplikation dar. Durch Kombination vorheriger Basisoperationen ergeben sich entsprechende Transformationenen für die Matrixvektormultiplikation $Ax \in \mathbb{R}^m$ einer Matrix $A \in \mathbb{R}^{m \times n}$ und Vektors $x \in \mathbb{R}^n$ zu

$$\begin{aligned}&\textbf{select } A.i \textbf{ as } i,\ \textbf{sum}(A.v * x.v) \textbf{ as } v\\&\textbf{from } A \textbf{ join } x \textbf{ on } A.j = x.i\\&\textbf{group by } A.i\end{aligned}$$

Für die Matrix $C = AB \in \mathbb{R}^{m \times n}$ als Produkt zweier Matrizen $A \in \mathbb{R}^{m \times k}$ und $B \in \mathbb{R}^{k \times n}$ lautet die Anfrage

$$\begin{aligned}&\textbf{select } A.i \textbf{ as } i,\ B.j \textbf{ as } j,\ \textbf{sum}(A.v * B.v) \textbf{ as } v\\&\textbf{from } A \textbf{ join } B \textbf{ on } A.j = B.i\\&\textbf{group by } A.i,\ B.j\end{aligned}$$

Schließlich kann auch die Transponierte A^T einer Matrix A einfach berechnet werden,

siehe dazu Anhang **??**.

Zusammenfassend stellt sich heraus, dass wesentliche Objekte der linearen Algebra und damit verbundene fundamentale Operationen im SQL-Kern umgesetzt werden können. Im folgenden Abschnitt werden diese Resultate genutzt und im Zusammenhang mit einem Machine- Learning-Verfahren eingesetzt. Problemstellung(Einleitung)

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

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

SQL

Relationen, Tensoren
Marizen/Vektoren als Relationen
Basisoperationen

3. Grundlagen neuronaler Netze

In diesem Kapitel werden Künstliche Neuronale Netze[8], kurz KNN, als Forschungsgegenstand der Informatik eingeführt und deren mathematische Grundlagen präzisiert. Sie stellen informationsverarbeitende Systeme nach dem Vorbild von tierischen beziehungsweise menschlichen Gehirnen dar und bestehen aus Neuronen in gewissen Zuständen und Schichten, die über gewichtete Verbindungen miteinander gekoppelt sind. Jene Gewichte sind als freie Parameter des neuronalen Netzes zu verstehen und können während des Trainingsprozesses so angepasst werden, um eine entsprechende Aufgabe zu lösen. Gelingt dies, so können neuronale Netze genutzt werden, um bestimmte Muster in Daten, typischerweise in Bildern, Audio oder Stromdaten, zu erkennen[24, 25, 35]. Sie eignen sich daher für viele typische Aufgaben des maschinellen Lernens, beispielsweise für die Klassifikation digitalisierter Objekte.

Im ersten Abschnitt wird das Perzeptron[27] als Grundeinheit eines neuronalen Netzes eingeführt. Im folgenden Abschnitt wird das Konzept der Multi-Layer-Perzeptronen[37] durch die Kopplung mehrerer Perzeptronen mit bestimmten Übertragungs- und Aktivierungsfunktion in einem Netz erläutert. Diese Repräsentierung eines KNN wird im weiteren Verlauf dieser Arbeit genutzt. Weiter wird das Training neuronaler Netze hinsichtlich der Klassifikationsaufgabe im Abschnitt 3.3 erläutert und schließlich eine kurze Zusammenfassung 3.4 gegeben.

3.1. Das Perzeptron

Zunächst wird das *Perzeptron* ähnlich wie in Minsky [22] als fundamentaler Baustein eines neuronalen Netzes eingeführt. Das Perzeptron wird oft als Basis moderner KNN angeführt und kann mithilfe des Perzeptron-Lernalgorithmus[27] trainiert werden, um das Problem der linearen Trennbarkeit **??**von Punktmengen zu lösen.

Definition 2 (Perzeptron)**.** *Für eine gegebene Funktion $\psi : \mathbb{R} \to \mathbb{R}$, einen Vektor $w \in \mathbb{R}^n$ und ein Skalar $\theta \in \mathbb{R}$ wird die Funktion*

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

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

Die Funktionsweise eines Perzeptrons ist in Abbildung 3.1 dargestellt. In dieser Arbeit wird das Perzeptron oft einfach Neuron geannt. Die Begriffe sind austauschbar und

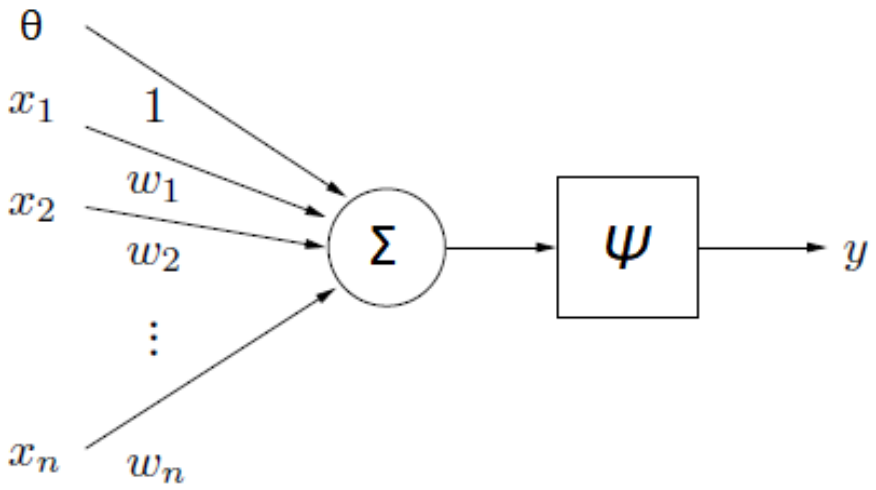

Abbildung 3.1.: Arbeitsweise eines Perzeptrons mit entsprechender Notation aus Definition 2.

meinen dasselbe. Bei der Wahl der Funktion ψ gibt es mehrere Möglichkeiten. Wird wie in Minsky[22] die Heavyside-Funktion

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

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

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

3.2. Multi-Layer-Perzeptron

In dieser Arbeit wird ein Künstliches Neuronales Netz als eine Menge von Perzeptronen, die in gewissen Schichten partitioniert und miteinander verbunden sind, notiert. Diese sogenannten *Multi-Layer-Perzeptronen*, kurz MLP, gelten als erste tiefe neuronale Netze und sind seit den späten 1980er-Jahren Gegenstand der Forschung[5, 4, 29]. Zunächst sind einige Definition notwendig, um eine lesbare Notation des MLPs zu geben.

Definition 3 (Übertragungsfunktion)**.** *Für eine gegebene Matrix* $W \in \mathbb{R}^{n \times m}$ *und einen*

Vektor $b \in \mathbb{R}^m$ ist

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

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

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

Definition 4 (Aktivierungsfunktion)**.** *Eine stetige, monoton steigende und nicht notwendigerweise lineare Funktion $\psi : \mathbb{R} \to \mathbb{R}$ wird als Aktivierungsfunktion bezeichnet.*

Es sei erwähnt, dass auch nicht monotone Aktivierungsfunktionen genutzt werden können, beispielsweise radiale Basisfunktionen[26], welche jedoch in dieser Arbeit nicht weiter von Interesse sind. Typische Aktivierungsfunktionen, weclhe heutzutage verwendet werden, sind die:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

für alle $x \in \mathbb{R}$.

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

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

für alle $x \in \mathbb{R}$.

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

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

Definition 6 (Neuronenschicht)**.** *Ist $\Psi^{W,b}$ eine Übertragungsfunktion mit den Parametern $W \in \mathbb{R}^{n\times m}$ 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 2. Eine Schicht besteht also aus m Perzeptronen $\tilde{\Psi}_i$ mit $y_i = \tilde{\Psi}(x_i) = \psi(W_{i,:}^T x + b_i)$ für $1 \leq i \leq m$.

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

Definition 7 (Multi-Layer-Perzeptron, vgl. gruening)**.** *Für eine gegebene Anzahl $l \in \mathbb{N}$, $l > 1$ von Schichten $\Psi^{W^{(1)},b^{(1)},\psi_1}, \ldots, \Psi^{W^{(l)},b^{(l)},\psi_l}$ bezeichne $s_l \in \mathbb{N}$ die Anzahl der*

Neuronen in Schicht l. Für eine Eingabe $x \in \mathbb{R}^{s_0}$ lässt sich die Ausgabe $y \in \mathbb{R}^{s_l}$ eines Multi-Layer-Perzeptron $\Lambda_l : \mathbb{R}^{s_0} \to \mathbb{R}^{s_l},\ x \mapsto y$ mit l Schichten durch

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

berechnen. Dabei gelten für die Gewichtsmatrizen die Dimensionsbedingungen

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

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

Algorithm 1 Vorwärtsrechnung

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

Das MLP-Modell wird im weiteren Verlauf dieser Arbeit repräsentativ als Künstliches Neuronales Netz bezeichnet. Die Begriffe MLP und FFN sind austauschbar. Die Funktionsauswertung eines FNN wird im Algorithmus Vorwärtsrechnung 1 festgehalten. Das zuvor angesprochene XOR-Problem kann nun beispielsweise mithilfe eines KNN bestehend aus zwei Schichten gelöst werden[12]. Es lassen sich zwischen Modell- und Hyperparameter von KNN unterscheiden.

Definition 8 (Hyper- und Modellparameter)**.** *Sei für $l \in \mathbb{N}$ ein KNN Λ_l gegeben. Dann werden die Eingabe- und Ausgabedimension s_0, s_l, die Anzahl l der (verdeckten) Schichten sowie die verwendeten Aktivierungsfunktion ψ_l Hyperparameter des neuronalen Netzes genannt. Die Gewichtsmatrizen und Biasvektoren mit den entsprechend passenden Abmessungen stellen die Modellparameter $\mathcal{W} := \{(W^{(i)}, b^{(i)}) :\ i = 1, \ldots, l\}$ des neuronales Netzes dar.*

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

3.3. Training neuronaler Netze

Künstliche Neuronale Netze gehören zu den typischen Vertretern von maschinellen Lernalgorithmen, welche hinsichtlich einer bestimmten Aufgabe, engl. *task* T, und einem Leistungsmaß, engl. *perfomance* P an der Erfahrung, engl. *experience* E lernen[12]. Dabei ist mit Lernen gemeint, dass das Computerprogramm bezüglich der Aufgabe T sein Leistungsmaß P mit wachsener Erfahrung E schrittweise steigert. Wie in Kapitel **??** erläutert, gibt es viele verschiedene Aufgaben, wie die Regression, Klassifikation oder Clusterung bestimmter Objekte.

In den folgenden Abschnitten wird das Klassifikationsproblem als *task* T im Mittelpunkt stehen. Weiter werden KNNs als Modellschätzer aus der Wahrscheinlichkeitstheorie interpretiert und fundamentale Aussagen wie das *Universal-Approximation-Theroem*[16] gegeben. Schließlich wird bezüglich der Klassifikationsaufgabe das Training neuronaler Netze erläutert.

3.3.1. Neuronale Netze als universelle Schätzer

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

Definition 9. *Seien die Mengen $D \subset \mathbb{R}^n$ und $\mathcal{C} = \{c_1, \dots, c_m\}$ gegeben. Eine Funktion $f : D \to \mathcal{C}$, welche ein Element aus D einer Klasse $c_i \in \mathcal{C}$ zuordnet, wird Klassifikationsfunktion genannt. Hier gibt es $m \in \mathbb{N}$ verschiedene Klassenlabels.*

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

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

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

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

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

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

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

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

Satz 1 (Universal-Approximation-Theroem[**gruen**]). *Sei ψ_1 eine nichtkonstante, beschränkkte Aktivierungsfunktion und id : $\mathbb{R} \to \mathbb{R}$ die Identität sowie $D \subset \mathbb{R}^n$ kompakt. Dann existieren für alle $\varepsilon > 0$ und stetigen Funktionen $f : D \to \mathbb{R}$ Parameter $N \in \mathbb{N}, W^{(1)} \in \mathbb{R}^{n \times N}, b^{(1)} \in \mathbb{R}^N$ sowie $W^{(2)} \in \mathbb{R}^{N \times 1}$, sodass*

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

gilt.

Beweis. Ein Beweis kann in Hornik[15] nachgelesen werden. □

Das Universal-Approximation-Theroem kann ebenfalls auf unbeschränkte und nichtkonstante Funktion $f : D \to \mathbb{R}^m$ erweitert werden. Heutzutage wird oft die ReLU-Funktion als Aktivierungsfunktion verwendet[31, 21].

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

Beweis. Siehe Sonoda et. al.[32]. □

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

Korollar 2. *Ein MLP mit zwei Schichten, wobei ψ_2 die Softmax-Funktion ist, kann genutzt werden, um stetige Funktionen $f : K \to [0, 1]^m$, welche von einem Kompaktum $K \subset \mathbb{R}^n$ in eine (Wahrscheinlichkeits)-Verteilung über die Klassen $\mathcal{C}$ abbilden, beliebig genau zu approximieren.*

Beweis. Siehe [**strauss**]. □

Die Aussage kann auf das MLP mit beliebig vielen Schichten erweitert werden. In dieser Arbeit umfasst die Menge D aus Definition 9 digitalisierte Objekte als Vektoren $x \in \mathbb{R}^n$ und ist endlich und damit kompakt. Daher kann wegen Korollar 2 das MLP als Modell genutzt werden, um stetige Funktionen P_{Daten} sinnvoll zu approximieren.

3.3.2. Optimale Parameterwahl bei neuronalen Netzen

Wird ein künstliches neuronales Netz als probabilistisches Modell genutzt und sind die Hyperparameter festgelegt, müssen die Modellparameter $\mathcal{W}$ gewählt werden. Um die Approximationsgüte, also die *perfomance P*, bezüglich des Klassifikationsproblems messbar zu machen, werden Fehlerfunktionen eingeführt. Mit Trainingsdaten als *experience E* und dem Gradientenverfahren[23] sollen optimale Parameter $\mathcal{W}$ gefunden werden, sodass die gewählte Fehlerfunktion minimiert wird. Im folgenden steht ein

MLP $\Lambda(\,\cdot\,;\mathcal{W}) : D \to [0,1]^m$ mit der Softmax-Funktion als Aktivierungsfunktion im Mittlepunkt, welches als parametrisiertes Modell f_{Modell} wie in 3.3 genutzt wird. Die Trainingsdaten werden in Trainingsmengen und Testmengen aufgeteilt.

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

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

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

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

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

wie in Bishop[2] berechnen. Für eine Trainingsmenge $\mathcal{T}$ soll das Produkt über alle Wahrscheinlichkeiten der korrekten Klassenzugehörigkeiten c gegeben der Eingaben x maximiert werden. Dieser Ansatz wird *Maximum Likelihood-Methode*[30] genannt und eine Parameterwahl ist durch eine Lösung des Optimierungsproblems

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

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

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

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

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

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

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

definiert.

Das Minimieren der negative log likelihood ist äquivalent zur Maximierung der Likelihood aus 3.5, denn es gilt

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

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

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

wobei $\hat{c} = f_{Modell}(x)$ und $t(x,c)$ der Zielvektor des Datenpaars (x,c) ist, siehe Goodfellow[12]. Das Problem 3.6 wird nun allgmemein mit Fehlerfunktionen definiert.

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

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

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

Backpropagationsalgorithmus

Bei FFN wird Backpropagation, etabliert von Rumelhart et. al. [29], als Lernalgorithmus genutzt, um eine gewählte Fehlerfunktion E zu minimieren. Dazu wird das Gradientenverfahren zur numerischen Minimierung von E im Raum der Modellparameter $\mathcal{W}$ genutzt. Dabei sei bemerkt, dass das Finden von globalen Minima durch den Backpropagationsalgorithmus nicht garantiert ist. In dieser Arbeit wird E immer als stückweise stetig differenzierbare Funktion gewählt, damit das Gradientenverfahren angewendet werden kann. Sowohl die negative log likelihood L_{NNL} als auch die mittlere quadratische Abweichung L_{MSE} sind als Fehlerfunktion geeignet. Weiter sollten auch die Aktivierungsfunktionen aus Definition 4 als stückweise stetig differenzierbare Funktion vorausgesetzt werden. Die Optimierung der Parameter geschieht iterativ und besteht jeweils aus zwei Schritten. Zuerst wird eine Abstiegsrichtung

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

berechnet und dann die Parameter

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

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

Die Abstiegsrichtungen werden 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 partielle Ableitungen mittels Kettenregel zu*

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

Beweis. siehe [11] □

Im Bereich des tiefen Lernens zählt die effiziente Berechnung dieser Gradienten zu den schwersten Aufgaben[20]. Im Folgenden wird der Online-Backpropagationsalgorithmus, im Englischen auch als *Stochastic Gradient Descent* bezeichnet, erläutert. Grundsätzlich lässt sich das Verfahren in zwei Schritte einteilen.

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

Als Fehlerfunktion wird im weiteren Verlauf die mittlere quadratische Abweichung

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

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

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

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

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

für $1 \leq k \leq s_l$ ermittelt werden. Dazu wird im Folgenden die Notation [**du_diss**]

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

genutzt. Es wird nur eine verdeckte Schicht mit s_j Neuronen betrachtet. Die Verallgemeinerung für beliebig viele Schichten ist analog. Mit i ist ein Eingabeneuron, j eine verstecktes Neuron und k eine Ausgabeneuron gemeint. Außerdem werden die Gradienten komponentenweise berechnet und daher die Paramter komponentenweise aktualisiert. Die mehrmalige Anwendung der Kettenregel 2 auf die Fehlerfunktion (3.12) liefert

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

mit

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

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

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

Die Gradienten für die Parameter der verdeckten Schicht lassen sich mithilfe der lokalen

Fehler δ_k der darüberliegenden Ausgabeschicht berechnen, also

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

mit x_i als Eingabeneuron und

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

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

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

Die Backpropagation wird als iterativer Algorithmus beschrieben. Daher tauchen im Algorithmus 2 Variablen 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 Traingsfehlers mit einer Fehlertoleranz $\varepsilon > 0$ formuliert werden.

Bei der Wahl der Lernrate werden heutzutage werden oft adaptive Verfahren genutzt, welche vorangegangene Gradienten berücksichtigen und die Lernrate so anpassen. Bekannte Verfahren sind *Nesterov accelerated gradient*[33], *AdaGrad*[10], *RMSProp*[34] sowie *Adam*[17]. So sollen Probleme wie des *vanishing gradients* oder des *exploding gradients* vermieden werden, siehe dazu[14]. Für eine tiefere Analyse des Gradientenverfahrens und dessen Varianten sei auf die jeweilgen Arbeiten beziehungsweise als Zusammenfassung auf Ruder[28] verwiesen. Darüber hinaus gibt es andere Techniken wie die Regularisierung, um die oben genannten Probleme zu entgehen. Für die Problemstellung in dieser Arbeit genügt es, das Online-Verfahren mit konstanter Lernrate zu nutzen.

3.4. Zusammenfassung

In diesem Kapitel wurden KNN als Feed-Forward-Netze eingeführt. Es stellt sich heraus, dass diese Netze als probabilistische Modelle genutzt werden können, um Klassi-

Algorithm 2 Online-Backpropagation für ein FFN, vgl. [**du_diss**]

Require: Trainingsmenge $\mathcal{T}$, Modellparameter $\mathcal{W}_0$, Fehlerfunktion E, Lernrate λ
Ensure: optimierte Modellparameter $\mathcal{W}$

```
Initialisiere zufällig alle Gewichte und Schwellwerte des FFN Λ
n = 0
while E > ε or n < N do                          ▷ Abbruchbedingung, siehe Text
    for (x, c) ∈ T do
        y(n) = Λ(x(n))
        e_k(n) = y_k(n) − t_k(n)
        Berechne die Gradienten bezüglich der Ausgabeschicht
        δ_k(n) = e_k(n)ψ'(V_k(n))
        Δw^l_{k,j}(n) = −λ δ_k(n) z_j(n)
        Δb^l_k(n) = −λ δ_k(n)
        Berechne die Gradienten bezüglich der verdeckten Schicht
        δ_j(n) = (Σ_{k=1}^{s_l} δ_k w^l_{k,j}) ψ'(V_j)
        Δw^h_{j,i}(n) = −λ δ_j(n) x_i(n)
        Δb^h_j(n) = −λ δ_j(n)
        Aktualisiere Gewichte
        W_{n+1} = W_n + ΔW_n
        n = n + 1
    end for
    E = (1/2) Σ_{(x,c)∈T} ||y − t||_2^2
end while
```

fikationsaufgaben hinreichend gut zu lösen. Dabei ist es wichtig die Hyper- und Modellparameter je nach Anwendung und Leistungsmaß optimal zu wählen. Dazu werden Trainingsdaten genutzt, um während eines Lernprozesses eine Fehlerfunktion zu Minimieren. Eine Lösung von 3.8 kann wegen der Komplexität der Fehlerfunktion E bzw. der großen Menge von Parametern $\mathcal{W}$ selten direkt angegeben werden[3]. Daher wird das Gradientenverfahren als iterativer Ansatz zur numerischen Mnimierung der Fehlerfunktion genutzt. Dabei sind partielle Ableitungen der Fehlerfunktion bezüglich der Parameter nötig, welche im Backpropagationsalgorithmus mithilfe der mehrdimensionalen Kettenregel berechnet werden können.

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

4. Gefaltete neuronale Netze

Feed-Forward-Netze gelten als leistungsstarke maschinelle Lernmethoden, da sie so trainiert werden können, um beliebige komplexe Funktionen abhängig von einer vektorwertigen Eingabe zu approximieren. Ist die Dimension der Eingabeschicht jedoch zu groß, treten bei klassischen FFN Probleme hinsichtlich der Paramteranzahl auf. Die Problemstellung **??** dieser Arbeit besteht in der Klassifikation digitalisierter Bilder. Wird ein FFN mit 100 Ausgabeneuronen genutzt und jeder Pixel eines Bildes mit den Abmessungen 1000×1000 als Merkmal genutzt, so ergeben sich bereits $10^8 + 100$ freie Parameter. Stehen nur relativ wenige Traingsdaten zur Verfügung, ist die Struktur des FFN zu komplex und dies kann zur Überanpassung führen[6, 1]. Die Parameteranzahl muss also deutlich reduziert werden. Konzepte wie *Parameter Sharing* und spärliche Konnektivität, engl. *sparse connectivity* erlauben diese Reduktion, vgl. Goodfellow[12] und werden in den folgenden Abschnitten erläutert.

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

In diesem Kapitel wird erläutert, wie gefaltete neuronale Netze die erwähnten Nachteile von FFN umgehen. CNN sind in der Lage, lokale Muster zu erkennen, sind äquivariant gegenüber Translationen und realisieren Konzepte wie Parameter Sharing, um die Anzahl der freien Parameter drastisch zu reduzieren. So gelingt es, besonders bei Aufgaben der Computergrafik[18, 19, 7] die Generalisierungsrate gegenüber klassichen FFN zu erhöhen.

Gefaltete neuronale Netze unterscheiden sich von FFN bei der Berechnung der Übertragungsfunktion. Dazu wird die gefaltete Übertragungsfunktion definiert, welche das Konzept der diskreten Faltung nutzt. Im folgenden Abschnitt 4.1wird zunächst die Faltung als mathematische Operation eingeführt und deren Zusammenhang zur Fourier-Transformation[38] erläutert. Anschließend wird im Abschnitt **??** das CNN-Modell definiert und schließlich im Abschnitt 4.3 der Backpropagationsalgorithmus 2 auf CNN verallgemeinert.

4.1. Die Faltungsoperation

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

Definition 13 (Faltung)**.** *Für zwei Funktionen $f, g : \mathbb{R}^n \to \mathbb{C}$ ist die Faltung als*

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

definiert, wobei gefordert wird, dass das Integral für fast alle x wohldefiniert ist. Für $f, g \in L^1(\mathbb{R}^n)$ ist dies der Fall.

Für die Faltung gelten einige Rechenregeln.

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

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

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

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

Definition 14 (Diskrete Faltung)**.** *Für zwei Funktionen $f, g : D \to \mathbb{C}$ mit einem diskreten Definitionsbereich $D \subseteq \mathbb{Z}^n$ ist die diskrete Faltung als*

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

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

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

Definition 15 (Zyklische Matrix, vgl. Gray[13])**.** *Eine quadratische Matrix heißt zyklisch im Vektor $a = (a_0, \ldots, a_{n-1})^T \in \mathbb{R}^n$, wenn sie die Gesatlt*

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

besitzt.

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

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

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

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

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

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

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

Bemerkung 3. Oft wird eine kompakte Schreibweise der Faltung von $X \in \mathbb{R}^{h \times b}$ und $K \in \mathbb{R}^{k \times k}$ mit

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

und $X_{i,j} = 0$ für $i \notin [h]$ und $j \notin [b]$ angegeben. Dabei wird der Kern negative indiziert. Ist beispielsweise $k = 5$, so ist mit $K_{-2,-2}$ das Matrixelement $k_{1,1}$ und mit $K_{0,0}$ der Wert im Mittelpunkt $k_{3,3}$ gemeint.

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

Bemerkung 4. Für Schrittweiten $s_h, s_b \in \mathbb{N}$ ergibt sich die reduzierte zweidimensionale Faltung $Y = X * K$ zu

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

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

4.2. CNN Architektur

Beim maschinellen Lernen sind Eingabedaten oft als mehrdimensionale Arrays abgelegt, welche eine oder mehrere Achsen repräsentieren, wobei die Ordnung dieser eine Rolle spielt. Bei digitalisierten Bildern sind das bespielsweise die Höhe und Breite des Bildes, welche als Raumachsen bezeichnet werden. Hinzu kommen Kanalachsen, zum Beispiel besitzen Grauwert-Bilder einen Farbkanal, während RGB-Farbbilder drei Kanäle der Farben rot, grün und blau besitzen. Dementsprechend werden Grauwert-Bilder wie in Definition 1 nun als dreidimensionale Arrays $X \in [0,1]^{h \times b \times 1}$ dargestellt. Dies erlaubt die Definition der gefalteten Übertragungsfunktion, wie in Gruening[**gruening**]

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

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

mit

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

wird gefaltete Übertragungsfunktion bezeichnet. Mit $*$ *ist die Matrixfaltung wie in Definition 16 gemeint und mit* $\cdot$ *werden beliebige Raumachsen bezeichnet. Die Skalare* α_p *können als lernbare Parameter genutzt werden. In dieser Arbeit gelte* $\alpha_p = 1$ *für alle* $p \in [z_{in}]$*.*

Bemerkung 5. Ist $\psi : \mathbb{R} \to \mathbb{R}$ eine Aktivierungsfunktion wie in Definition 4, so wird für $X \in \mathbb{R}^{\cdot \times \cdot \times z}$ mit

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

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

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

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

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

gegeben. Die Matrizen $Y_{:,:,p}$ *werden für* $1 \leq p \leq z_{out}$ *Merkmalskarten genannt. Weiter bezeichne* $\Psi_{conv}^{K,b,\psi}$ *die Faltungsschicht* $\mathcal{S}_{conv}$ *mit* $\Psi_{conv}^{K,b,\psi}(X) := \psi\left(\Psi_{conv}^{K,b}(X)\right)$*.*

Bei CNN werden sogenannte *Pooling*-Schichten verwendet, um die Dimensionen der Raumachsen neben dem zero padding weiter zu verkleinern und das Modell robuster gegenüber Überanpassung zu machen. Dazu werden Pooling-Funktionen eingesetzt,

welche unabhängig voneinander auf Merkmalskarten operieren und so die Rechenkomplexität des Modells reduzieren. Es ist sinnvoll, symmetrische Pooling-Funktionen T zu wählen, für die die Bedingung

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

gilt. Mögliche Pooling-Funktionen sind

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

Heutzutage wird oft die von Weng et. al.[36] eingeführte Max-Pooling-Schicht benutzt. 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 19 (Pooling-Schicht, vgl. Grüning [**gruening**])**.** *Seien $p_h, p_b \in \mathbb{N}$ und T eine Pooling-Funktion. Die Funktion*

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

mit

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

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 Information von Eingabedaten, welche sich lokal in Fenstern der Größe $p_h \times p_b$ befinden und reduzieren so die Raumdimension. In dieser Arbeit wird das Maximum-Pooling oder Mittelwert-Pooling benutzt. Für andere Pooling-Funktion sei auf Yu et. al.[39] verwiesen. Die Idee bei CNN besteht nun darin, Faltungsschichten mit Pooling-Schichten zu kombinieren und schließlich ein FFN wie in Definition 7 anzuknüpfen. Dazu wird für allgemeine mehrdimensionale Arrays die Flatten-Schicht definiert.

Definition 20. *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 multidimensionale Eingabe wird also in einen Vektor umgewandelt.

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

Definition 21. *(Gefaltetes Neuronales Netz) Seien $h, w, z_{in}, z_{out}, l, c \in \mathbb{N}$ und Schichten $\Psi^{W^{(1)},b^{(1)},\psi_1}, \ldots, \Psi^{W^{(l)},b^{(l)},\psi_l}$ sowie Faltungsschichten $\Psi_{conv}^{K^{(1)},b^{(1)},\psi_1}, \ldots, \Psi_{conv}^{K^{(l)},b^{(l)},\psi_l}$ gegeben. Die Vorwärtsrechnung eines CNN lässt sich als Komposition*

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

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

Auch CNN bestehen aus Hyper- und Modellparametern.

Definition 22 (Hyper- und Modellparameter von CNN)**.** *Die Eingabe- und Ausgabedimensionen h, b, z_{in}, z_{out}, die Anzahl c der Faltunsschichten, dei 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 2 soll nun für das CNN-Modell verallgemeinert werden. Als Fehlerfunktion wird wieder die mittlere quadratische Abweichung (3.11) genutzt. Die Abstiegsrichtungen für die Schichten des FFN wurden bereits im vorherigen Kapitel 3 hergeleitet. Es müssen nun Gradienten innerhalb der Faltungsschichten beziehungsweise Pooling-Schichten berechnet werden. Die Einträge der Eingabe- und Merkmalskarten, in Zukunft einfach Karten genannt, können als Neuronenaktivierungen interpretiert werden. Weiter werden zur Vereinfachung die trainierbaren Gewichte der Kerne mit $w_{j,i}^{(l)}$ bezeichnet. Gemeint ist also das Gewicht zwischen Neuron i auf Schicht $l-1$ und Neuron j der Schicht l bezeichnet. Weiter sei $y_i^{(l-1)}$ die Aktivierung des Neurons i der Schicht $l-1$ und $\delta_j^{(l)}$ sei der lokale Fehler des Neurons j der Schicht l. Die Funktion ψ repräsentiere eine Aktivierungsfunktion, Pooling-Funktion oder Flatten-Funktion, je nachdem auf welcher Schicht sie benutzt wird.

Die Gradienten werden wieder komponentenweise berechnet. Angenommen, l ist eine Faltungsschicht. Die Aktivierung einer Merkmalskarte j an der Stelle (x, y) lässt sich mit der Matrixfaltung, siehe 16, 17, als

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

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

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

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

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

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

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

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

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

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

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

Dieses Vorgehen wird auch Upsampling genannt.

!!!!!!!

!!! noch als algorithmus aufschreiben und mit dem folgenden Beispiel vergleichen, ob Indizies/lokale Fehler usw. richtig sind!!!

!!!!!!

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[19] bietet 60.000 Trainingsbilder handgeschriebener Ziffern und 10.000 Testbilder, welche jeweils durch menschliches Wissen annotiert sind. Dieser Datensatz gilt als typischer Benchmark zur Klassifikation von Ziffern und wird im weiteren Verlauf dieser Arbeit genutzt. Jedes einzelne Bild besteht aus 28×28 Pixeln, welche jeweils einen Grauwert zwischen 0 und 1 annehmen, vgl. Abbildung 4.1. Ein Bild wird als $X \in [0, 1]^{28 \times 28 \times 1}$ und ein Trainingspaare als (X, c)

mit $c \in \{0, \ldots, 9\}$ bezeichnet.

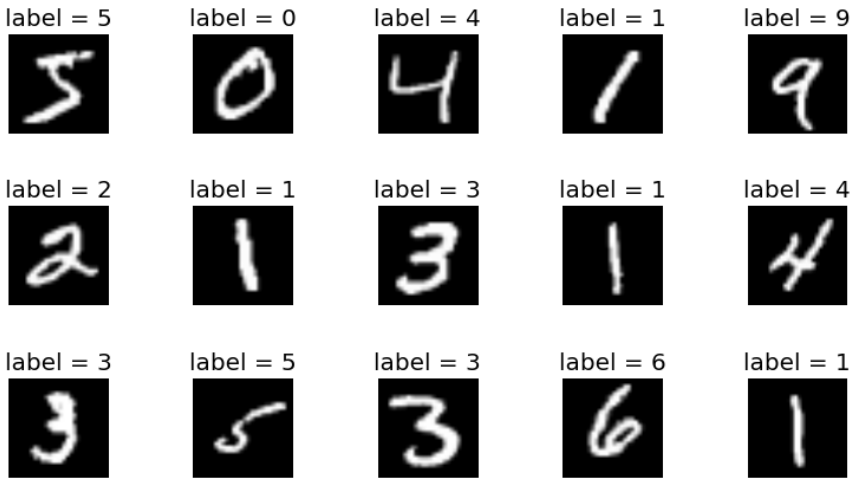

Abbildung 4.1.: Beispielbilder, vgl. [19] aus der öffentlichen MNIST-Datenbank. Zu sehen sind handgeschriebene Ziffern und zugehörige Annotationen.

Es wird ein CNN mit zwei Faltungsschichten C^1 und C^2 mit der logistischen Aktivierungsfunktion ψ genutzt. Die Ausgabe der Schicht C^1 umfasst sechs Merkmalskarten und die Ausgabe der Schicht C^2 zwölf Merkmalskarten. Zur Faltung werden quadratische 5×5 -Kerne verwendet. Außerdem werden zwei Pooling-Schichten P^1 und P^2mit 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[19], ist in Abbildung **??** dargestellt. Dabei sind

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

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

Vorwärtsrechnung

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

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

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

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

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

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

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

$$\begin{aligned} C^2_q &= \psi\left(\sum_{p=1}^{6} P^1_p * K^2_{q,p} + \mathbf{1} b^2_q\right) \\ C^2_q(i,j) &= \psi\left(\sum_{p=1}^{6}\sum_{u=-2}^{2}\sum_{v=-2}^{2} P^1_p(i+u, j+v)\cdot K^2_{q,p}(u,v) + b^2_q\right) \end{aligned} \tag{4.8}$$

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

Mittelwert-Pooling P^2 mit

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

für $1 \leq q \leq 12$. Es gilt $P^2 \in \mathbb{R}^{4\times4\times12}$. Diese zwölf 4×4- Matrizen werden durch die Flatten-Funktion T_f aus Definition 20 in einen Vektor $f \in \mathbb{R}^{4\cdot4\cdot12}$ umgewandelt. Dies sei durch

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

beschrieben. Die Umkehrung wird mit

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

bezeichnet. Die Ausgabe des FFN ergibt sich schließlich zu

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

und der mittlere quadratische Fehler lautet

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

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

Backpropagation

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

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

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

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

darstellen. Analog gilt

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

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

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

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

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

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

$$\begin{aligned}
\delta_q^{C^2}(i,j) &= \delta_q^{P^2}(\lceil i/2 \rceil, \lceil j/2 \rceil) \cdot \nabla \phi(\lceil i/2 \rceil) \\
&= \delta_q^{P^2}(\lceil i/2 \rceil, \lceil j/2 \rceil) \cdot \frac{1}{4} \quad \forall i,j \in [8], \forall q \in [12].
\end{aligned}$$

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

$$\begin{aligned}
\Delta K^2_{q,p}(u,v) &= -\lambda \frac{\partial E}{\partial K^2_{q,p}(u,v)} \\
&= -\lambda \sum_{i=1}^{8} \sum_{j=1}^{8} \frac{\partial E}{\partial C^2_q(i,j)} \cdot \frac{\partial C^2_q(i,j)}{\partial K^2_{q,p}(u,v)} \\
&= -\lambda \sum_{i=1}^{8} \sum_{j=1}^{8} \delta_q^{C^2}(i,j) \cdot \frac{\partial}{K^2_{q,p}(u,v)} \psi \left(\sum_{p=1}^{6} \sum_{u=0}^{4} \sum_{v=0}^{4} P^1_p(i+u, j+v) \cdot K^2_{q,p}(u,v) + b^2_q \right) \\
&= -\lambda \sum_{i=1}^{8} \sum_{j=1}^{8} \delta_q^{C^2}(i,j) \cdot C^2_q(i,j) \left(1 - C^2_q(i,j)\right) \cdot P^1_p(i+u, j+v)
\end{aligned}$$

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

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

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

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

Es gilt

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

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

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

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

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

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

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

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

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

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

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

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

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

Dann gilt schließlich

$$\begin{aligned}\Delta K^1_{p,1}(u,v) &= -\lambda\sum_{i=1}^{24}\sum_{j=1}^{24}X_(u+i,v+j)\cdot\delta^{C^1}_{p,\psi}(i,j)\\ \Rightarrow \Delta K^1_{p,1} &= -\lambda\, X * \delta^{C^2}_{q,\psi},\quad \forall p\in[6]\end{aligned}$$

und

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

4.5. Motivation der Faltung

Weiter machen! Sie nutzt wichtige Konzepte zur Optimierung von Machine-Learning-Verfahren wie spärliche Konnektivität (engl. *sparse connectivity*), *Parameter Sharing*

und *äquivariante Repräsentation*, vgl. [**goodfellow**]. Spärliche Konnektivität bedeutet, dass Neuronen auf einer Schicht $\int_{l+1}$ nur durch wenige Neuronen der Schicht $\mathcal{S}_l$ beeinflusst wird. Dies ist bei CNNs typisch, da meist die verwendeten Filter viel kleiner als die Eingabe ist. Noch mehr erklären + Abbildung

Mit Parameter Sharing ist die Nutzung von gleichen Parametern für mehrere Funktionen im neuronalen Netz gemeint. In herkömmlichen Feed-Forward-Netzen wird jedes Element der Gewichtsmatrizen für die Berechnung der Aktivierungen der jeweiligen Schichten verwendet. Anschließend werden diese Gewichte dann nicht mehr gebraucht. Im Zusammenhang von CNNs bedeutet Parameter Sharing während der Faltungsoperation, dass nur eine bestimmte Menge von Parametern erlernt werden müssen Noch mehr erklären + Abbildung

5. Datenbankgestützte Implementierung von CNN

5.1. Die Faltungsoperation in SQL

5.2. Datenbankgestützte FFN

5.3. Evalution

6. Zusammenfassung und Ausblick

Algorithm 3 An algorithm with caption

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

Literatur

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[20] 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.

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

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

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

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

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

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

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

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

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

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

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

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

[33] 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.

[34] 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.

[35] 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.

[36] Juyang Weng, Narendra Ahuja und Thomas S Huang. „Cresceptron: a self-organizing neural network which grows adaptively“. In: *[Proceedings 1992] IJCNN International Joint Conference on Neural Networks*. Bd. 1. IEEE. 1992, S. 576–581.

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

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

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

Erklärung

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

Rostock, den 12.08.2021

_____
Vorname Nachname

A. Anhang

A.1. Listings

Listing A.1: C Code - direkt eingefügt

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

int main()
{
    int i;

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

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

    return 0;
}
```

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

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

A.2. Biber

BibLaTeX mit Biber

BibLaTeX

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

Biber

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

Einbinden von Biber in Editoren

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

Ablauf

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

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

Erläuterungen zum Ablauf

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

Literaturverwaltungsprogramm Citavi

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

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

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

Erzeugung einer *.bib-Datei mit Citavi:

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

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

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

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

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

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

Beispiel einer Literaturdatenbank test1.bib

```
% This file was created with Citavi 6.3.0.0

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