Grundlagen der Algebra

Stephan Epp
hjstephan86@gmail.com

23. Juli 2025

Inhaltsverzeichnis

Kapitel 1

Einleitung

Was ist Algebra? Von seiner Wortbedeutung her bedeutet Algebra die *Lehre von den Gleichungen* oder die *Theorie der Verknüpfungen mathematischer Strukturen*. Um Aussagen darüber treffen zu können, was Gleichungen oder Verknüpfungen sind, muss definiert werden, wie die Strukturen aussehen, die miteinander verglichen oder verknüpft werden. Um den Begriff Struktur besser zu verstehen, lohnt sich ein Blick in die Informatik. Mögliche Strukturen in der Informatik sind zum Beispiel *Listen* oder *Stacks*. Dabei enthält ein Liste eine Menge von Elementen, wobei die Elemente jeweils einen Wert annehmen. Die Beschreibung der Liste als Datenstruktur eignet sich aber auch für die Beschreibung der mathematischen Struktur des *Vektors*. Mögliche Verknüpfungen von Vektoren sind zum Beispiel die Addition oder Multiplikation zweier Vektoren, die in dieser Verknüpfung wieder einen Vektor als Ergebnis haben. Es ist nicht unüblich in der Informatik auch eine Liste von Listen zu nutzen, um zur Laufzeit effizient auf die Datenstruktur zugreifen zu können. Die Beschreibung der Liste von Listen eignet sich aber auch für die Beschreibung der mathematischen Struktur der *Matrix*. Mögliche Verknüpfungen von Matrizen sind zum Beispiel die Addition oder Multiplikation zweier Matrizen, die in dieser Verknüpfung wieder eine Matrix als Ergebnis haben.

In dieser Arbeit werden die grundlegenden Strukturen der Algebra definiert, der Vektor und die Matrix. Der Begriff *Projektion* wird eingeführt und als Ergebnis von linear kombinierten Vektoren aufgefasst. Dabei bildet der *Kern* die kleinste nicht mehr projizierbare Einheit in seiner Umgebung des Raumes, in dem er zum Projizieren verwendet wird. Anschließend werden Verknüpfungen für die Struktur des Vektors und für die Struktur der Matrix definiert, das sind die Addition und die Multiplikation. Dabei fällt auf, dass Vektoren in der Verknüpfung der Multiplikation sich leichter verknüpfen lassen als Matrizen. Vektoren bilden einen *wohl geformten Körper* und Matrizen nicht. Strukturen können einen Schwerpunkt haben. Den Schwerpunkt einer Struktur zu finden, lohnt sich, da Strukturen in ihrem Schwerpunkt miteinander verknüpft werden können und erst im Schwerpunkt die Verknüpfung am leichtesten ist hinsichtlich der benötigten Rechenoperationen.

Es wird gezeigt, dass die 1 die wichtigste Zahl ist. Es wird auch gezeigt, dass $\{0, 1\}$ die wichtigsten Zahlen sind, da der Wert von binären Zahlen exponentiell in der Basis 2 ist und dieser Wertebereich das effiziente Anwenden von Verknüpfungen im Schwerpunkt durch Halbieren ermöglicht.

Der Algorithmus Strassen-25 wird vorgestellt, der zwei $n \times n$ Matrizen mit einer Laufzeit von $O(n^{2.3219})$ multipliziert. Dieser modifiziert den bekannten Algorithmus von Strassen für die Matrixmultiplikation. Es wird gezeigt, dass zwei $n \times n$ Matrizen nicht schneller multipliziert werden können. Es gibt keinen Teile-und-Herrsche Algorithmus zur Multiplikation von $n \times n$ Matrizen, der durch Nutzen von linearen Abhängigkeiten weniger als 5 Matrixmultiplikationen benötigt. Es wird das optimale Teile-und-Herrsche Prinzip eingeführt. Das optimale Teile-und-Herrsche Prinzip ist die effizienteste Methode, um Probleme zu lösen, wenn die Probleminstanz bei jedem rekursiven Aufruf halbiert wird.

Der Algorithmus Strassen-25 eignet sich zum Finden von Kernen in ihrem Raum. Darauf aufbauend wird ein Verfahren vorgestellt, mit der Matrizen effizient verknüpft werden können, unabhängig davon, ob es sich bei der Verknüpfung um eine Addition oder Multiplikation handelt. Die Idee ist, die Menge aller Kerne in einem eingeschränkten Wertebereich zu finden und damit alle Ergebnisse für eine konkrete Verknüpfung zu projizieren.

Außerdem wird auf das Deep-Learning eingegangen mit einer Änderung in der Vorgehensweise. Die Idee ist, nicht die herkömmliche Matrixmultiplikation zweier $n \times n$ Matrizen zu verwenden. Effizienter ist es, die Gewichtsmatrix einer Ebene in ihren Vektoren aufzufassen und jeden einzelnen Vektor mit dem Vektor der Eingabe für das neuronale Netz zu multiplizieren.

Kapitel 2

Definitionen

Zur Betrachtung der Algebra folgen Definitionen, die für den weiteren Verlauf dieser Arbeit nützlich sind.

2.1 Strukturen

Definiton 2.1.1. *Ein Vektor* $\boldsymbol{v} = (v_1, \ldots, v_n)$ *aus dem Raum* $\mathbb{R}^n$ *hat die Größe* n *und ist ein Tupel von* n *Elementen, wobei jedes Element* $v_i \in \mathbb{R}$.

Zu beachten ist, dass der Vektor $\boldsymbol{v}$ fett geschrieben ist. Zum Beispiel ist $\mathbf{0}$ der Vektor, bei dem alle Elemente den Wert null haben.

Definiton 2.1.2. *Eine Matrix* A *aus dem Raum* $\mathbb{R}^{n\times m}$ *hat die Größe* $n \times m$ *und besteht aus* n *Zeilen und* m *Spalten, wobei jedes Element* $a_{ij} \in \mathbb{R}$.

Definiton 2.1.3. *Zwei Vektoren* $\boldsymbol{u}$ *und* $\boldsymbol{v}$ *projizieren den Vektor* $\boldsymbol{w}$ *genau dann, wenn es Konstanten* k_1 *und* k_2 *gibt, so dass*

$$k_1\boldsymbol{u} + k_2\boldsymbol{v} = \boldsymbol{w} \neq \mathbf{0},$$

dabei haben $\boldsymbol{u}$, $\boldsymbol{v}$ *und* $\boldsymbol{w}$ *dieselbe Größe und* $k_i \in \mathbb{R}$, $k_i \neq 0$.

Das heißt, die Vektoren $\boldsymbol{u}$ und $\boldsymbol{v}$ bilden eine *Projektion* $\boldsymbol{w}$ in Abhängigkeit der Konstanten k_1, k_2, wobei die Konstanten nicht den Wert null haben.

Definiton 2.1.4. *Für die Projektion* $\boldsymbol{w}$ *durch die Vektoren* $\boldsymbol{u}$ *und* $\boldsymbol{v}$ *liegen* $\boldsymbol{u}$ *und* $\boldsymbol{v}$ *in der Umgebung von* $\boldsymbol{w}$ *im Raum* $\mathbb{R}^n$, *wobei* $\boldsymbol{u}$, $\boldsymbol{v}$ *und* $\boldsymbol{w}$ *jeweils Größe* n *haben.*

Nicht alle Vektoren $\boldsymbol{u}$ und $\boldsymbol{v}$ liegen in der Umgebung von $\boldsymbol{w}$. Es gibt Vektoren im Raum $\mathbb{R}^n$, durch die $\boldsymbol{w}$ niemals projiziert werden kann.

Definiton 2.1.5. *Eine Projektion* $\boldsymbol{w}$ *durch die Vektoren* $\boldsymbol{u}$ *und* $\boldsymbol{v}$ *und Konstanten* k_1, k_2 *ist minimal, wenn für alle Konstanten* k_i *gilt, wird eine Konstante* $k_i = 0$, *dann ist*

$$k_1\boldsymbol{u} + k_2\boldsymbol{v} = \mathbf{0}.$$

Das bedeutet, wenn eine minimale Projektion gefunden wurde, entferne die eine Konstante k, d.h., $k = 0$, und erhalte mit $k_1\boldsymbol{u} + k_2\boldsymbol{v} = \mathbf{0}$ die *größte Einheit*, mit der, egal wie ihre Vektoren miteinander kombiniert werden, nichts mehr projiziert werden kann außer $\mathbf{0}$. Damit wurde eine größte und nicht mehr projizierbare Einheit gefunden, der *Kern*.

Der Kern in seiner Umgebung des Raumes ist nicht teilbar, er kann nicht weiter reduziert werden. Der *triviale Kern* besteht nur aus den Vektoren, bei denen jeweils nur ein Element den Wert eins hat, sonst haben alle anderen Elemente den Wert null.

Definiton 2.1.6. *Die Einheitsmatrix* E *ist gegeben durch den trivialen Kern in entsprechender Ordnung,*

$$E = (\boldsymbol{e_1}\ \boldsymbol{e_2}\ \boldsymbol{e_3}\ \boldsymbol{e_4}) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

wobei E *die Größe vier hat und die Vektoren* $\boldsymbol{e_1}, \ldots, \boldsymbol{e_4}$ *jeweils die Größe vier haben.*

Es ist beachten, dass bei den Vektoren $\boldsymbol{e_1}, \ldots, \boldsymbol{e_4}$ jeweils nur ein Element den Wert eins hat, sonst haben alle anderen Elemente den Wert null. Außerdem gilt:

$$k_1\boldsymbol{e_1} + k_2\boldsymbol{e_2} + k_3\boldsymbol{e_3} + k_4\boldsymbol{e_4} = \boldsymbol{0},$$

egal, welchen Wert die Konstanten $k_i \in \mathbb{R}$ annehmen. In dieser Ordnung $\boldsymbol{e_1}, \ldots, \boldsymbol{e_4}$ bilden sie die Einheitsmatrix und auch einen Kern im Raum $\mathbb{R}^{4\times4}$.

2.2 Verknüpfungen

Es ist bekannt, dass bei der Vektoraddition oder -multiplikation zwei Vektoren im Allgemeinen nicht miteinander addiert oder multipliziert werden können. Zwei Vektoren können zum Beispiel nur dann miteinander multipliziert werden, wenn sie dieselbe Größe haben. Erst dann ergibt das Produkt der beiden Vektoren wieder einen Vektor derselben Größe.

Der *Bezug* zwischen den Vektoren einer Umgebung ist durch die Verknüpfung, d.h., die Operation wie der Addition oder Multiplikation, im Ergebnis der Operation gegeben. Ist das Ergebnis der Operation $\boldsymbol{0}$, dann gibt es keinen Bezug. Ist das Ergebnis der Operation $\boldsymbol{w}$ und $\boldsymbol{w} \neq \boldsymbol{0}$, dann gibt es einen Bezug. Dieser Bezug oder, allgemeiner, diese Beziehungen werden veranschaulicht in den bekannten Operationen für Vektoren, das sind die Vektoraddition und die Vektormultiplikation. Insbesondere wird der Bezug aber auch beschrieben durch die Projektion $\boldsymbol{w}$ durch $\boldsymbol{u}$ und $\boldsymbol{v}$.

Definiton 2.2.1. *Die Summe zweier Vektoren* $\boldsymbol{u} = (u_1, \ldots, u_n)$ *und* $\boldsymbol{v} = (v_1, \ldots, v_n)$ *ist gegeben durch*

$$\boldsymbol{u} + \boldsymbol{v} = (u_1 + w_1, \ldots, u_n + w_n) = \boldsymbol{w},$$

wobei $\boldsymbol{u}$*,* $\boldsymbol{v}$ *und* $\boldsymbol{w}$ *jeweils Größe* n *haben.*

Definiton 2.2.2. *Die Produkt zweier Vektoren* $\boldsymbol{u} = (u_1, \ldots, u_n)$ *und* $\boldsymbol{v} = (v_1, \ldots, v_n)$ *ist gegeben durch*

$$\boldsymbol{u} \cdot \boldsymbol{v} = (u_1 \cdot w_1, \ldots, u_n \cdot w_n) = \boldsymbol{w},$$

wobei $\boldsymbol{u}$*,* $\boldsymbol{v}$ *und* $\boldsymbol{w}$ *jeweils Größe* n *haben.*

Es fällt auf, dass die Verknüpfung der Addition für die Vektoren $\boldsymbol{u}$ und $\boldsymbol{v}$ für die Verknüpfung der Multiplikation durch die Addition ersetzt wird. Bei der Definition der Addition und der Multiplikation sind die Elemente v_i der Vektoren reelle Zahlen, d.h., $v_i \in \mathbb{R}$. Diese Definition der Addition und der Multiplikation für die Vektoren $\boldsymbol{u}$ und $\boldsymbol{v}$ gilt aber auch, wenn die Elemente v_i der Vektoren binäre Zahlen sind, d.h., $v_i \in \{0, 1\}$. Dabei steht die Summe der Vektoren für die logische ODER-Verknüpfung, kurz $\vee$, und das Produkt der Vektoren für die logische UND-Verknüpfung, kurz $\wedge$.

Es ist bekannt, dass bei der Matrixaddition oder -multiplikation zwei Matrizen im Allgemeinen nicht miteinander addiert oder multipliziert werden können. Zwei Matrizen können zum Beispiel nur dann miteinander multipliziert werden, wenn die Anzahl der Spalten der ersten Matrix gerade die Anzahl der Zeilen der zweiten Matrix ist. Erst dann ergibt das Produkt der beiden Matrizen wieder eine Matrix mit entsprechender Größe. Die Anzahl der Zeilen der Produktmatrix ist die Anzahl der Zeilen der ersten Matrix und die Anzahl der Spalten der Produktmatrix ist die Anzahl der Spalten der zweiten Matrix.

Definiton 2.2.3. *Die Summe zweier Matrizen* A *und* B *ist gegeben durch*

$$A + B = (\boldsymbol{a_1} \ \ldots \ \boldsymbol{a_m}) + (\boldsymbol{b_1} \ \ldots \ \boldsymbol{b_m}) = (\boldsymbol{a_1} + \boldsymbol{b_1} \ \ldots \ \boldsymbol{a_m} + \boldsymbol{b_m}) = C,$$

wobei A *eine* $n \times m$ *Matrix,* B *eine* $n \times m$ *Matrix und* C *eine* $n \times m$ *Matrix ist.*

Die Matrix C als Summe von A und B wird gebildet durch die paarweise Addition $(\boldsymbol{a_1} + \boldsymbol{b_1} \ \ldots \ \boldsymbol{a_m} + \boldsymbol{b_m})$ der Vektoren $(\boldsymbol{a_1} \ \ldots \ \boldsymbol{a_m})$ und $(\boldsymbol{b_1} \ \ldots \ \boldsymbol{b_m})$.

Definiton 2.2.4. *Das Produkt zweier Matrizen* A *und* B *ist gegeben durch*

$$A \cdot B = C = (c_{ij}), \ \textit{mit } c_{ij} = \sum_{k=1}^{r} a_{ik} b_{kj},$$

wobei A *eine* $n \times r$ *Matrix,* B *eine* $r \times m$ *Matrix und* C *eine* $n \times m$ *Matrix ist.*

Die Produktmatrix C mit $C = (c_{ij})$ wird gebildet durch die Summe $\sum_{k=1}^{r} a_{ik}b_{kj}$ über alle $1 \leq i \leq n$ und alle $1 \leq j \leq m$ für c_{ij}. Dabei werden für das Element c_{ij} genau r Summanden $a_{ik}b_{kj}$ über alle $1 \leq k \leq r$ addiert.

Es lohnt sich, den Schwerpunkt von Strukturen für eine bestimme Verknüpfung zu finden, wenn es ihn für eine bestimmte Struktur gibt. Denn in ihrem Schwerpunkt lassen sich die Strukturen effizient verknüpfen. Sie an anderer Stelle zu verknüpfen, ist vielleicht auch möglich. Nur kann dies den Aufwand der Verknüpfung erhöhen. Wieso haben Vektoren einen leichteren Schwerpunkt als Matrizen in der Verknüpfung der Multiplikation? Es sind zwei Vektoren und paarweise zwei Multiplikationen pro Element des Vektors. Es sind zwei Matrizen und paarweise aber mehr als zwei Multiplikationen pro Element der Matrix. Wo liegt der Schwerpunkt der Matrix hinsichtlich der Verknüpfung der Multiplikation?

2.3 Körper

Warum sind Vektoren in ihrer Struktur so wohl geformt in dem Sinn, dass die Verknüpfung der Addition einfach durch die Multiplikation oder die Verknüpfung der Multiplikation einfach durch die Addition ersetzt werden kann? Die Vektoren bilden einen Körper mit einfachen Eigenschaften. Matrizen haben diese Eigenschaft in der Verknüpfung nicht. Die Addition kann nicht einfach durch die Multiplikation ersetzt werden, da die Struktur der Matrix aufwendiger ist in der Anwendung der Multiplikation. Der Aufwand für eine einfache Vorgehensweise zur Multiplikation von zwei Matrizen hat eine Laufzeit von $O(n^3)$, wenn zwei $n \times n$ Matrizen multipliziert werden. Vektoren dagegen können paarweise multipliziert werden. Der Aufwand für eine einfache Vorgehensweise zur Multiplikation von zwei Vektoren hat im Vergleich dazu eine Laufzeit von $O(n)$. Es wurde bereits darauf eingegangen, dass die Verknüpfungen (Addition und Multiplikation) bei Vektoren sowohl für Elemente aus den reellen als auch den binären Zahlen gelten.

Wie verhalten sich die Vektoren hinsichtlich des neutralen Elements und des inversen Elements? Wie verhalten sich Vektoren im Bereich der komplexen Zahlen? Welche Vorteile hat es, dass Vektoren die Struktur eines Körpers haben? Eignen sich die Vektoren aus algebraischer und analytischer Sicht besonders? Geht es mehr um die Instanzen dieses Körpers, welche die konkreten Vektoren dann sind und damit eine angenehme Behandlung hinsichtlich maximal vieler Verknüpfungen auf ihnen ermöglichen? Für welchen Wertebereich sind Instanzen des Körpers immer noch Instanzen des Körpers, nur für die reellen Zahlen und die binären Zahlen? Ist es das Ziel beim Finden von charakteristische Eigenschaften eines Körpers besonders viele charakteristische Eigenschaften zu finden?

Satz 2.3.1. *Diese charakteristischen Eigenschaften bilden einen wohl geformten Körper: (1) die Verknüpfungen: Addition und Multiplikation und (2) der Wertebereich.*

Beweis. Klar ist, je mehr charakteristische Eigenschaften ein Körper hat, desto weniger wahrscheinlich ist es, dass es Instanzen dieses Körpers auf möglichst vielen Verknüpfungen und Wertebereichen gibt. Bei der Untersuchung dieser Frage geht es um die Verknüpfungen und die Wertebereiche. Denn in ihrer Kombination wird das Ergebnis bestimmt und sichtbar, ob eine Instanz des Körpers noch Instanz des Körpers ist. Es muss also um Verknüpfungen und Wertebereiche gehen. Werden die Verknüpfungen nicht betrachtet, lässt sich kein Ergebnis berechnen, da die Instanzen miteinander noch nicht mal zu einem Ergebnis kombiniert werden. Wird der Wertebereich nicht betrachtet, lässt sich auch kein Ergebnis in einem konkreten Wertebereich wie den reellen oder binären Zahlen berechnen.

Es reicht für einen wohl geformten Körper, die Verknüpfungen und die Wertebereiche zu betrachten. Werden weitere Eigenschaften wie das neutrale oder das inverse Element als charakteristische Eigenschaften betrachtet, geht dies mit mehr Aufwand in der Anwendung der Verknüpfungen für solche Instanzen einher. Dies wird zum Beispiel daran deutlich, dass der Körper, wie ursprünglich definiert, die 1 als neutrales Element hat aber der Ring, wie ursprünglich definiert, nicht.

Die Differenz zwischen 1 und 0 ist die größte, es gibt keine größere Differenz. Dass für das Betrachten der größten Differenz nur benachbarte ganze Zahlen relevant sind, geht daraus hervor, dass nur die kleinsten, nicht weiter teilbaren Zahlen dazu in Betracht kommen und erst darin ihr eigentlicher Wert durch Vergleichen klar wird. Die Differenz zwischen 2 und 1 ist nicht so groß. Warum? Egal, mit welcher anderen Zahl außer 0 die 0 vervielfacht wird, das Ergebnis der Verfielfachung ist wieder 0. Bei der 1 ist das anders. Wird die 1 mit 2 vervielfacht, ist das Ergebnis der Verfielfachung 2. Das ist eigenartig und stellt die Frage, warum erscheint die Vervielfachung als angemessenere Verknüpfung in ihrer Wirkung als die einfache Addition? Die Verfielfachung ist angenehmer, weil mit ihr ein größerer Ergebnisraum erreicht wird als mit der Addition. Die Addition kann auf der

anderen Seite durch die Multiplikation nicht ersetzt werden. Dies begründet, dass die Addition und die Multiplikation beides notwendige Verknüpfungen sind für einen wohl geformten Körper. □

Satz 2.3.2. *Die 1 ist die wichtigste Zahl aller ganzen Zahlen.*

Beweis. Die Differenz zwischen 0 und 1 ist größer ist als die zwischen 2 und 1, aber die Differenz zwischen 3 und 2 ist nicht größer als die zwischen 2 und 1. Dies gilt für alle folgenden Zahlen von 2, 3, 4 usw. Damit ist klar, dass 0 und 1 die wichtigsten Zahlen sind, denn ihre Differenz ist unter allen ganzen Zahlen die größte. Da 0 aber keinen Wert hat, außer den, den sie der 1 in der Differenz gibt, ist von ihnen die 1 die wichtigere Zahl. □

Satz 2.3.3. *Die binären Zahlen* $\{0, 1\}$ *sind die wichtigsten Zahlen.*

Beweis. Die binären Zahlen sind die wichtigsten Zahlen aller Zahlen. Nicht die natürlichen oder die ganzen oder die reellen Zahlen sind wichtiger. Denn die binären Zahlen enthalten die wichtigsten aller Zahlen, die 1 und die 0. Die binären Zahlen sind in ihrer Anzahl so klein, dass ihnen darin auch die größte Bedeutung zukommt im Vergleich zu den anderen Zahlen. Die 1 ist die wichtigste Zahl und nach ihr die 0. Denn der Bezug zur 0 ist bezogen auf die Differenz der wichtigere, da 2 schon wieder die Summe aus 1 und 1 ist.

Der Wert von binären Zahlen ist exponentiell in der Basis 2. Erst dies ermöglicht andererseits das Halbieren in der effizienten Anwendung von Verknüpfungen von Strukturen in ihrem Schwerpunkt. Denn damit wird die optimale Anwendung hinsichtlich der daraus resultierenden logarithmischen Laufzeit für die Rekursionstiefe erreicht. □

In der Natur findet das Teilen im Schwerpunkt auch in der Zellteilung biologischen Lebens statt. Ohne Zellteilung gibt es kein neues Leben. Dass die binären Zahlen heute mit der Informatik viele Lebensbereiche so erleichtern, überrascht daher nicht. Die binären Zahlen sind die Zahlen, die die größte Bedeutung haben, wenn es um das Finden von wohl geformten Körpern geht.

Lemma 2.3.1. *Vektoren bilden einen wohl geformten Körper für binäre oder reelle Zahlen.*

Beweis. Der Beweis geht daraus hervor, dass Vektoren diese charakteristischen Eigenschaften haben: (1) Die Verknüpfungen wie Addition und Multiplikation wie in Definition 2.2.1 und Definition 2.2.2 definiert für die (2) Wertebereiche der binären oder reellen Zahlen. □

Satz 2.3.4. *Die Algebra ist Grundlage für die Analyse.*

Beweis. Ohne Algebra macht die Analyse keinen Sinn, denn ohne Wertebereiche, ob für natürliche, reelle oder komplexe Zahlen, lässt sich keine analytische Aussage über das Ergebnis einer Verknüpfung machen. □

Wenn Matrizen hinsichtlich der Matrixmultiplikation eine interessante Anwendung für Graphen haben, wird das auch auf jeden Fall gelten für den wohl geformten Körper der Vektoren. Betrachtet

Algorithmus 1 APSP(A, B)

Eingabe: $\langle A, B\rangle$ mit $n \times n$ Matrix A und B
Ausgabe: $\langle C\rangle$ mit $n \times n$ Matrix C, die die Längen der kürzesten Wege aller (i, j) enthält

```
1: for i = 1 to n do
2:     for j = 1 to n do
3:         for k = 1 to n do
4:             if A[i][k] ≠ ∞ und B[k][j] ≠ ∞ then
5:                 C[i][j] = min(C[i][j], A[i][k] + B[k][j])
6: return C
```

man bei der Multiplikation zweier Matrizen, dass die Addition durch min(...), das Minimum, und die Multiplikation durch die Addition ersetzt wird, kann das All-Pairs-Shortest-Paths (APSP) Problem gelöst werden. Der Algorithmus 1 berechnet in der Matrix C die kürzeste Distanz für alle Knoten i, j des Graphen unter Verwendung dieser modifizierten Matrixmultiplikation. Werden die Kanten des Graphen mit der Adjazenzmatrix A beschrieben, dann wird Algorithmus 1 mit APSP(A, A) aufgerufen. Zur Ermittlung von C benötigt Algorithmus 1 eine Laufzeit von $O(n^3)$. Es lohnt sich also für den wohl geformten Körper der Vektoren effiziente Algorithmen zu entwickeln.

Kapitel 3

Anwendungen

Zur Anwendung der grundlegenden Definitionen der Strukturen und Verknüpfungen werden in diesem Kapitel konkrete Vorgehensweisen zur Verknüpfung der Strukturen beschrieben.

3.1 Matrixmultiplikation

Dieser Abschnitt beschreibt Vorgehensweisen zur Verknüpfung der Multiplikation auf der Struktur der Matrix. Dabei wird angenommen, dass zwei $n \times n$ Matrizen A und B multipliziert werden.

3.1.1 Herkömmliche Multiplikation

Die herkömmliche Matrixmultiplikation ist gegeben durch ihre Definition und benötigt in der Anwendung eine Laufzeit von $O(n^3)$, wenn wir der Einfachheit davon ausgehen, dass zwei $n \times n$ Matrizen miteinander multipliziert werden. Algorithmus 2 zeigt die Vorgehensweise zur Berechnung der Pro-

Algorithmus 2 Matrixmultiplikation(A, B)

Eingabe: $\langle A, B \rangle$ mit $n \times n$ Matrix A und B
Ausgabe: $\langle C \rangle$ mit $n \times n$ Matrix $C = A \cdot B$

```
1: for i = 1 to n do
2:     for j = 1 to n do
3:         for k = 1 to n do
4:             C[i][j] = C[i][j] + (A[i][k] · B[k][j])
5: return C
```

duktmatrix $C = A \cdot B$.

3.1.2 Verbesserte Multiplikation

Es folgt der Algorithmus 3 von Strassen als Pseudocode. Als Eingabe erhalten wir zwei $n \times n$ Matrizen. Der Einfachheit halber wird angenommen, dass n eine Zweierpotenz ist. Zu Beginn prüfen wir, ob die Größe der Matrizen bereits den Wert 1 hat. Haben die Matrizen den Wert 1, geben wir das Produkt AB zurück. In Zeile 2 und 3 werden die Matrizen A und B so definiert, dass in den Zeilen 4 bis 10 die 5 Matrixmultiplikationen jeweils durchgeführt werden. In den Zeilen 11 bis 14 werden 4 Matrizen C_{ij} durch Addition und Subtraktion der Matrizen A_{ij} berechnet und in Zeile 15 Matrix C als Ergebnis zurückgegeben.

Als Idee zum Beweis der Korrektheit betrachten wir das Produkt $A \cdot B$ der zwei Matrizen A und B mit

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \quad \text{und} \quad B = \begin{pmatrix} e & f \\ g & h \end{pmatrix}.$$

Zur Vereinfachung beinhalten die Matrizen nur skalare Werte. Das Ergebnis $C = A \cdot B$ ist

$$C = \begin{pmatrix} ae + bg & af + bh \\ ce + dg & cf + dh \end{pmatrix}.$$

Algorithmus 3 STRASSEN(A, B)

Eingabe: $\langle A, B\rangle$ mit $n \times n$ Matrizen A, B, $n = 2^k$, $k \in \mathbb{N}$
Ausgabe: $\langle C\rangle$ mit Produktmatrix $C = AB$

1: **if** $n = 1$ **then return** $C = AB$
2: $A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}$
3: $B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$
4: $P_1 = \text{STRASSEN}(A_{11} + A_{22}, B_{11} + B_{22})$
5: $P_2 = \text{STRASSEN}(A_{21} + A_{22}, B_{11})$
6: $P_3 = \text{STRASSEN}(A_{11}, B_{12} - B_{22})$
7: $P_4 = \text{STRASSEN}(A_{22}, B_{21} - B_{11})$
8: $P_5 = \text{STRASSEN}(A_{11} + A_{12}, B_{22})$
9: $P_6 = \text{STRASSEN}(A_{21} - A_{11}, B_{11} + B_{12})$
10: $P_7 = \text{STRASSEN}(A_{12} - A_{22}, B_{21} + B_{22})$
11: $C_{11} = P_1 + P_4 - P_5 + P_7$
12: $C_{12} = P_3 + P_5$
13: $C_{21} = P_2 + P_4$
14: $C_{22} = P_1 - P_2 + P_3 + P_6$
15: **return** $C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$

Der Algorithmus von STRASSEN berechnet P_r mit

$$\begin{aligned}
P_1 = \text{STRASSEN}(a + d, e + h) &= ae + ah + de + dh,\\
P_2 = \text{STRASSEN}(c + d, e) &= ce + de,\\
P_3 = \text{STRASSEN}(a, f - h) &= af - ah,\\
P_4 = \text{STRASSEN}(d, g - e) &= dg - de,\\
P_5 = \text{STRASSEN}(a + b, h) &= ah + bh,\\
P_6 = \text{STRASSEN}(c - a, e + f) &= ce + cf - ae - af,\\
P_7 = \text{STRASSEN}(b - d, g + h) &= bg + bh - dg - dh.
\end{aligned}$$

Dann werden C_{ij} berechnet mit

$$\begin{aligned}
C_{11} = P_1 + P_4 - P_5 + P_7 &= ae + bg,\\
C_{12} = P_3 + P_5 &= af + bh,\\
C_{21} = P_2 + P_4 &= ce + dg,\\
C_{22} = P_1 - P_2 + P_3 + P_6 &= cf + dh,
\end{aligned}$$

wobei z.B. $C_{11} = (ae + ah + de + dh) + (dg - de) - (ah + bh) + (bg + bh - dg - dh) = ae + bg$. Für einen formalen Beweis der Korrektheit verzichten wir auf die vollständige Induktion über $n \in \mathbb{N}$ der $n \times n$ Matrizen.

3.1.2.1 Laufzeitanalyse

Für die Multiplikation von zwei $n \times n$ Matrizen lautet die Rekurrenzgleichung für den Algorithmus von STRASSEN

$$T(n) = 7\,T(\tfrac{n}{2}) + c\,n^2.$$

Der Algorithmus halbiert in jedem rekursiven Aufruf die beiden $n \times n$ Matrizen zu vier $\frac{n}{2} \times \frac{n}{2}$ Matrizen. Substituieren wir n durch $\frac{n}{2}$, erhalten wir für

$$T(\tfrac{n}{2}) = 7\,T(\tfrac{n}{4}) + c(\tfrac{n}{2})^2 = 7\,T(\tfrac{n}{4}) + \tfrac{c}{4}n^2.$$

Nach dem *ersten* rekursiven Aufruf erhalten wir mit $T(\frac{n}{2})$ eingesetzt in $T(n)$ dann

$$\begin{aligned}
T(n) &= 7\,(7\,T(\tfrac{n}{4}) + \tfrac{c}{4}\,n^2) + cn^2\\
&= 7^2\,T(\tfrac{n}{4}) + \tfrac{7}{4}\,cn^2 + cn^2.
\end{aligned}$$

Mit dem *zweiten* rekursiven Aufruf werden die vier $\frac{n}{2} \times \frac{n}{2}$ Matrizen wieder halbiert zu acht $\frac{n}{4} \times \frac{n}{4}$ Matrizen. Damit ist

$$T(\tfrac{n}{4}) = 7\,T(\tfrac{n}{8}) + c(\tfrac{n}{4})^2 = 7\,T(\tfrac{n}{8}) + \tfrac{c}{16}n^2.$$

Wird $T(\frac{n}{4})$ eingesetzt in $T(n)$ ergibt sich

$$\begin{aligned} T(n) &= 7^2\,(7\,T(\tfrac{n}{8}) + \tfrac{c}{16}n^2) + \tfrac{7}{4}cn^2 + cn^2 \\ &= 7^3\,T(\tfrac{n}{2^3}) + \tfrac{7^2}{4^2}cn^2 + \tfrac{7}{4}cn^2 + cn^2. \end{aligned}$$

Betrachten wir nun den k-ten rekursiven Aufruf finden wir für

$$T(n) = 7^k\,T\left(\tfrac{n}{2^k}\right) + cn^2 \sum_{i=0}^{k-1} \left(\frac{7}{4}\right)^i.$$

Zur Vereinfachung belassen wir es bei dem k-ten rekursiven Aufruf auch in $T(n)$ bei k und nicht $k+1$. Kleinere Matrizen als 1×1 Matrizen gibt es nicht, daher können die $n \times n$ Matrizen nur k mal halbiert werden. Der größte Wert, den k annehmen kann, ist $k = \log_2 n$. Damit ist

$$\begin{aligned} T(n) &= 7^{\log_2 n}\,T\Big(\frac{n}{2^{\log_2 n}}\Big) + cn^2 \sum_{i=0}^{\log_2 n - 1} \left(\tfrac{7}{4}\right)^i \\ &= n^{\log_2 7}\,T(1) + cn^2\,\frac{\left(\frac{7}{4}\right)^{\log_2 n} - 1}{\frac{7}{4} - 1} \\ &= O\left(n^{2.8074}\right) + cn^2\left(\tfrac{7}{4}\right)^{\log_2 n} \\ &= O\left(n^{2.8074}\right) + cn^2 \cdot n^{\log_2 \frac{7}{4}} \\ &= O\left(n^{2.8074}\right) + cn^{2.8074} \\ &= O\left(n^{2.8074}\right). \end{aligned}$$

3.1.3 Optimale Multiplikation

Um die Anzahl der benötigten Matrixmultiplikationen von 7 auf 5 zu reduzieren, wird die Diagonale e und h der Matrix B betrachtet. Dadurch wird für n eine Laufzeit im Exponenten von 2.3219 statt 2.807 erreicht. Betrachtet man $P_2 = (c+d) \cdot e = ce + de$ und $P_5 = (a+b) \cdot h = ah + bh$, dann fällt auf, dass P_2 durch P_6 und P_1 ermittelt werden kann,

$$\begin{aligned} ce &= P_6 - cf + ae + af, \\ de &= P_1 - ae - ah - dh, \end{aligned}$$

und P_5 durch P_1 und P_7

$$\begin{aligned} ah &= P_1 - ae - de - dh, \\ bh &= P_7 - bg + dg + dh. \end{aligned}$$

Dazu müssen P_1, P_6 und P_7 berechnet werden bevor P_2 und P_5 ohne Multiplikation bestimmt werden können:

$$\begin{aligned} P_2 &= ce + de = (P_6 - cf + ae + af) + (P_1 - ae - ah - dh) = P_6 + P_1 - cf + af - ah + af, \\ P_5 &= ah + bh = (P_1 - ae - de - dh) + (P_7 - bg + dg + dh) = P_1 + P_7 - ae - de - bg + dg. \end{aligned}$$

Da nur 5 Matrixmultiplikationen verwendet werden, ergibt sich für die Laufzeitanalyse

$$T(n) = 5^{\log_2 n}\,T\Big(\frac{n}{2^{\log_2 n}}\Big) + cn^2 \sum_{i=0}^{\log_2 n - 1} \left(\tfrac{5}{4}\right)^i = n^{\log_2 5}\,T(1) + cn^2\,\frac{\left(\frac{5}{4}\right)^{\log_2 n} - 1}{\frac{5}{4} - 1} = O\left(n^{2.3219}\right).$$

Dazu wird der Algorithmus von Strassen zu Strassen-25 so geändert, dass P_2 und P_5 in Zeile 9 und 10 ermittelt werden. Damit P_2 und P_5 ermittelt werden können, müssen die zuvor ermittelten Produkte P_r, $r \in \{1, 6, 7\}$ verwendet werden. Das bedeutet für

$$\begin{aligned} P_2 &= P_6 + P_1 - A_{21}B_{12} + A_{11}B_{12} - A_{11}B_{22} + A_{11}B_{12}, \\ P_5 &= P_1 + P_7 - A_{11}B_{11} - A_{22}B_{11} - A_{21}B_{21} + A_{22}B_{21}. \end{aligned}$$

Algorithmus 4 STRASSEN-25(A, B)

Eingabe: $\langle A, B\rangle$ mit $n \times n$ Matrizen A, B, $n = 2^k$, $k \in \mathbb{N}$
Ausgabe: $\langle C, A_{ij}, B_{ij}\rangle$ mit Produktmatrix $C = AB$ und A_{ij}, B_{ij}

1: **if** $n = 1$ **then return** $C = AB$
2: $A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}$
3: $B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$
4: $P_1 =$ STRASSEN-25$(A_{11} + A_{22}, B_{11} + B_{22})$
5: $P_3 =$ STRASSEN-25$(A_{11}, B_{12} - B_{22})$
6: $P_4 =$ STRASSEN-25$(A_{22}, B_{21} - B_{11})$
7: $P_6 =$ STRASSEN-25$(A_{21} - A_{11}, B_{11} + B_{12})$
8: $P_7 =$ STRASSEN-25$(A_{12} - A_{22}, B_{21} + B_{22})$
9: $P_2 = P_6 + P_1 - A_{21}B_{12} + A_{11}B_{12} - A_{11}B_{22} + A_{11}B_{12}$
10: $P_5 = P_1 + P_7 - A_{11}B_{11} - A_{22}B_{11} - A_{21}B_{21} + A_{22}B_{21}$
11: $C_{11} = P_1 + P_4 - P_5 + P_7$
12: $C_{12} = P_3 + P_5$
13: $C_{21} = P_2 + P_4$
14: $C_{22} = P_1 - P_2 + P_3 + P_6$
15: **return** $C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$

Dabei ist P_2 *abhängig* von P_6 und P_7 und P_5 ist *abhängig* von P_1 und P_7. Damit P_2 und P_5 ohne Matrixmultiplikation ermittelt werden können, müssen die Produkte $A_{ij}B_{ij}$ im vorherigen Rekursionsschritt jeweils gespeichert werden. Damit in jedem Rekursionsschritt die Produkte $A_{ij}B_{ij}$ effizient gespeichert werden können, muss auf eine Datenstruktur mit einem eindeutigen Schlüssel (k, i_A, j_A, i_B, j_B) in $O(1)$ zugegriffen werden, $n = 2^k$, $k \in \mathbb{N}$.

Satz 3.1.1. *Es gibt keinen Teile-und-Herrsche Algorithmus zur Multiplikation von $n \times n$ Matrizen, der durch Nutzen von Abhängigkeiten weniger als 8 Matrixmultiplikationen benötigt und damit auch weniger als 5 Matrixmultiplikationen benötigt.*

Beweis. Angenommen, es gibt einen Teile-und-Herrsche Algorithmus zur Multiplikation von $n \times n$ Matrizen, der mit weniger als 5 Matrixmultiplikationen auskommt, dann muss dieser Algorithmus auch Abhängigkeiten $P_r = (P_s, P_t, A_{ij}B_{ij})$ verwenden, mit denen P_r ermittelt werden kann, $r < 5$. Da $dim(A_{ij}) = dim(B_{ij}) = \frac{n}{2}$, ist es nicht möglich $C = AB$ zu berechnen. Das gesamte Bild der *projizierten* Produktmatrix C kann nur projiziert werden, wenn mindestens mehr als die Hälfte beider Matrizen A_{ij} und B_{ij} zum Projizieren verfügbar ist. □

Satz 3.1.2. *Das optimale Teile-und-Herrsche Prinzip ist die effizienteste Methode, um Probleme zu lösen, wenn die Probleminstanz bei jedem rekursiven Aufruf halbiert wird.*

Beweis. In diesen drei Kriterien ist der Beweis zu führen: (1) Rekursionstiefe, (2) Anzahl der Teilprobleme, (3) Größe eines Teilproblems. Dabei wird schnell klar, dass es alles in der Halbierung zusammengefasst optimal ist. Denn die Halbierung ist nichts anderes als eine Verdoppelung im Nenner.

Zu (1): Wichtig in der Analyse der Laufzeit eines Algorithmus oder einer Lösungsvorschrift, bestehend aus einfachen Operationen und dem Aufruf von Funktionen, ist das Betrachten der gerufenen Funktionen. Weniger wichtig sind einfache Operationen und Zuweisungen. Daher ist für die Laufzeitanalyse die Rekursionstiefe von wichtigster Bedeutung (sieht man z.B. beim Algorithmus von STRASSEN oder beim Algorithmus MERGE-SORT).

Die Rekursionstiefe ist logarithmisch. Kleiner kann die Tiefe für die gesamte Lösung nicht sein, da das Problem mit der 2 im Nenner logarithmisch geteilt wird. Die Funktion, die am stärksten wächst und dabei immer noch eine Umkehrfunktion hat, ist die Exponentialfunktion. Daher ist ihre Umkehrfunktion am schnellsten in der Reduzierung der Rekursionstiefe für die Laufzeitanalyse für Teile-und-Herrsche Algorithmen. Eine Erhöhung des Nenners führt zwangsläufig zu einer Veränderung der anderen beiden Kriterien.

Zu (2), (3): Für die Anzahl der Teilprobleme und für die Größe eines Teilproblems gilt: Angenommen, der Nenner wird vergrößert. Dann wird die ursprüngliche Probleminstanz aber nicht in ihrem leichtesten Punkt geteilt. Denn ein Problem ist nur da am leichtesten zu teilen, wo sein Schwerpunkt

liegt. Es gibt für jede Probleminstanz nur einen Schwerpunkt. Wenn der Nenner ganzzahlig verkleinert wird, wird nicht mehr geteilt. Es muss ganzzahlig verkleinert werden, da z.B. ein einzelnes Array-Element nicht teilbar ist. Die Zahl e scheidet damit in der Wahl der Exponentialfunktion aus. Daher muss die Exponentialfunktion 2 als Basis haben. □

Auffällig ist, dass nur das Problem, bei dem zwei Instanzen durch einen Operator verbunden sind, mit dem optimalen Teile-und-Herrsche Prinzip zu lösen ist. Das sind Probleme mit Operatoren wie z.B., $+$, $-$, $<$. Im Prinzip werden diese Probleme an ihrem Schwerpunkt gelöst, wo sie trotz ihrer Schwere am leichtesten zu tragen sind, wie bei einer Wippe mit zwei gleichen Gewichten jeweils am äußersten Ende. Die Wippe ist dann exakt horizontal ausgerichtet im Gleichgewicht.

Lemma 3.1.1. *Schneller als mit $T(n) = O(n^{2.3219})$ können zwei $n \times n$ Matrizen nicht multipliziert werden.*

Beweis. Der Beweis folgt daraus, dass STRASSEN-25(A, B) zwei $n \times n$ Matrizen nach dem optimalen Teile-und-Herrsche Prinzip multipliziert. □

Satz 3.1.3. *Wenn es bei einer Lösung erforderlich ist, jedes Element der Eingabe während der Abarbeitung der Lösungsvorschrift einmal zu betrachten, dann gibt es keine bessere Laufzeit als die logarithmische.*

Beweis. Wenn Laufzeit so aufgefasst wird, dass in jedem Schritt ein Register besucht wird, dann kann die Anzahl der unterschiedlich besuchten Register nicht weniger als logarithmisch sein. Die Sonnenblumenkerne in einer Sonnenblume sind logarithmisch angeordnet und damit optimal untergebracht mit minimal wenig Platz. Sie sind im goldenen Winkel angebracht. Würden sie besser angebracht werden können, dann wäre der goldene Winkel kein goldener Winkel. □

Lemma 3.1.2. *Das optimale Teile-und-Herrsche Prinzip ist optimal unter allen Lösungsprinzipien, wo es während der Abarbeitung der Lösungsvorschrift erforderlich ist, jedes Element der Eingabe während der Lösung einmal zu betrachten.*

Beweis. Der Beweis folgt daraus, dass Algorithmen, die nach dem optimalen Teile-und-Herrsche Prinzip Lösungsvorschriften abarbeiten, logarithmische Laufzeit haben müssen. □

3.1.3.1 Erwartete Ergebnisse

Bei der Analyse der zu messenden Laufzeit von Algorithmus STRASSEN-25 ist zu erwarten, dass dieser mit nur 5 Matrixmultiplikationen deutlich weniger Rechenzeit benötigt als der Algorithmus von STRASSEN mit 7 Matrixmultiplikationen. Die Tabelle 3.1 zeigt die theoretische Anzahl benötigter

Tabelle 3.1: Vergleich der theoretischen Anzahl an Operationen

n	Standard	STRASSEN	STRASSEN-25	Matrixgröße $(n \times n)$
10	$10^3 = 1.000$	$10^{2.8074} \approx 642$	$10^{2.3219} \approx 209$	100
100	$100^3 = 10^6$	$100^{2.8074} \approx 6.4 \times 10^5$	$100^{2.3219} \approx 2.09 \times 10^4$	10^4
1.000	$1.000^3 = 10^9$	$1.000^{2.8074} \approx 6.4 \times 10^8$	$1.000^{2.3219} \approx 2.09 \times 10^7$	10^6
10.000	$10.000^3 = 10^{12}$	$10.000^{2.8074} \approx 6.4 \times 10^{11}$	$10.000^{2.3219} \approx 2.09 \times 10^9$	10^8
100.000	$100.000^3 = 10^{15}$	$100.000^{2.8074} \approx 6.4 \times 10^{14}$	$100.000^{2.3219} \approx 2.09 \times 10^{11}$	10^{10}
1.000.000	$1.000.000^3 = 10^{18}$	$1.000.000^{2.8074} \approx 6.4 \times 10^{17}$	$1.000.000^{2.3219} \approx 2.09 \times 10^{13}$	10^{12}
10^7	10^{21}	6.4×10^{20}	2.09×10^{15}	10^{14}
10^8	10^{24}	6.4×10^{23}	2.09×10^{17}	10^{16}

Operationen für die Matrixmultiplikation in Abhängigkeit von n. Die Operationen sind elementare arithmetische Operationen wie Multiplikationen und Additionen/Subtraktionen von Skalaren. Die Anzahl der Operationen in der Tabelle sind asymptotisch und ignorieren konstante Faktoren, um die Skalierungseffekte zu betonen.

Die Wahl des geeigneten Algorithmus zur Matrixmultiplikation hängt stark von der Größe der Matrizen und dem spezifischen Anwendungsbereich ab. Interessant ist, dass schon für $n = 100$ nur 10^4 Operationen benötigt werden von STRASSEN-25 und dagegen 10^5 Operationen von STRASSEN. Für $n \geq 10.000$ unterscheiden sich STRASSEN-25 und STRASSEN um 2 Zehnerpotenzen und für $n \geq 10^8$ unterscheiden sie sich um 6 Zehnerpotenzen. Praktische Anwendungsbereiche für die $n \times n$ Matrixmultiplikation sind die folgenden:

1. **Kleine Matrizen** ($n < 200$): Für kleinere Matrizen, wie sie in der Computergrafik oder bei einfachen linearen Gleichungssystemen vorkommen, ist der Standardalgorithmus mit $O(n^3)$ aufgrund seines geringen zusätzlichen Aufwands für die Rekursion und optimaler Cache-Nutzung die erste Wahl.
2. **Mittlere bis große Matrizen** ($n \approx 200 - 10.000$): Im wissenschaftlichen Rechnen, bei Optimierungsproblemen oder im maschinellen Lernen dominieren weiterhin optimierte Implementierungen des Standardalgorithmus.
3. **Sehr große Matrizen** ($n \geq 10.000$): Bei großen Matrizen, wie sie in Big Data Analysen oder beim Training umfangreicher Deep-Learning-Modelle auftreten, wird der Bedarf an Alternativen zu optimierten Standardalgorithmen deutlich. Damit gewinnt der Algorithmus von STRASSEN an Attraktivität und Relevanz für die Bewältigung rechenintensiver Multiplikationen.

3.1.4 Deep-Learning

Die Idee des Deep-Learning ist, dass das Wissen eines neuronalen Netzes durch Gewichtsmatrizen W_i beschrieben wird. Das neuronale Netz hat s Ebenen und jeder Ebene wird eine Gewichtsmatrix W_i für ein bestimmtes zu lernendes Merkmal zugeordnet. Die Eingabe für das Netzwerk ist ein Vektor $\boldsymbol{v}$, der in seiner *charakteristischen Art* vom Netzwerk gelernt werden soll und durch s *Merkmale* beschrieben ist. Macht das Netzwerk auf einer Ebene beim Lernen des Merkmals einen Fehler, wird dieser durch Subtraktion korrigiert und die korrigierte Gewichtsmatrix $W_i^{'}$ für die nächste Berechnung in der Ebene i verwendet. Für Ebene i wurde $W_i^{'}$ also in Abhängigkeit des abweichenden Fehlers gelernt. Je kleiner die Fehler werden, desto besser sind die gelernten Gewichtsmatrizen $W_i^{'}$ in der Zuordnung des Merkmals von $\boldsymbol{v}$. Ein neuronales Netz, das ein beliebiges $\boldsymbol{v}$ in seiner charakteristischen Art immer richtig zuordnet, hat auf allen Ebenen Gewichtsmatrizen $W_i^{'}$ gelernt, die in der Zuordnung keinen Fehler mehr machen.

Im Allgemeinen wird das Lernen in neuronalen Netzen für eine beliebige Ebene i mit der Matrixmultiplikation gelöst. Betrachtet man die Multiplikation von W_i und $\boldsymbol{v}$, wobei W_i eine $n \times n$ Matrix und $\boldsymbol{v}$ ein $n \times 1$ Vektor ist, dann ist das Ergebnis dieser Multiplikation wieder ein Vektor der Größe $n \times 1$. Um n Vektoren gleichzeitig zu lernen, wurden n Vektoren als $n \times n$ Matrix aufgefasst und dazu die Matrixmultiplikation verwendet. Eine Alternative dazu ist, die Gewichtsmatrix W_i der Größe $n \times n$ als n Vektoren der Größe $1 \times n$ aufzufassen und n Vektormultiplikationen zu verwenden. Sollen dann k Vektoren gleichzeitig gelernt werden, hat das Lernen bezogen auf die Multiplikation eine Laufzeit von $k \cdot O(n^2)$. Dabei muss k nicht den Wert n annehmen, da n sehr groß werden kann. Also ist $k \cdot O(n^2)$ immer noch schneller als $O(n^{2.3219})$, wenn STRASSEN-25 für die Matrixmultiplikation verwendet wird.

Die Gewichtsmatrix W_i einer Ebene i eines neuronalen Netzwerks beschreibt wie stark die Neuronen in dieser Ebene für die Erkennung des Merkmals ausgeprägt sind. Die Neuronen W_i werden dann stärker, wenn sie in der Erkennung eines Merkmals einen Fehler gemacht haben und entsprechend korrigiert werden. Eine andere Möglichkeit, ein beliebiges $\boldsymbol{v}$ in seiner charakteristischen Art noch besser zuordnen zu können, besteht darin, die Gewichtsmatrizen W_i auf allen Ebenen zu invertieren. Kann das neuronale Netz $\boldsymbol{v}$ in seiner charakteristischen Art zuordnen, wird auch geprüft, ob das neuronale Netz mit W_i^{-1} die Eingabe $\boldsymbol{v}$ in seiner charakteristischen Art verwirft, $1 \leq i \leq s$. Dadurch ist die Aussage des neuronalen Netzes in der Zuordnung einer beliebigen Eingabe $\boldsymbol{v}$ sicherer.

Zum Beispiel wird dazu ein einfaches neuronales Netz betrachtet, welches zuordnet, ob eine vorgegebene Zeichnung ein Haus darstellt oder nicht. Wenn das neuronale Netz zuordnet, die Zeichnung ist ein Haus, dann wird das neuronale Netz auf allen Ebenen mit den Gewichtsmatrizen W_i^{-1} ausgeführt. Verwirft das neuronale Netz dann die Zeichnung, ist die Zeichnung mit höherer Wahrscheinlichkeit ein Haus.

3.2 Finden von Kernen

In der Beweisführung, dass es keinen Teile-und-Herrsche Algorithmus zur Multiplikation von $n \times n$ Matrizen gibt, der weniger als 5 Matrixmultiplikationen benötigt, wurde die Eigenschaft des Projizierens genutzt. Zwei Vektoren $\boldsymbol{u}$ und $\boldsymbol{v}$ projizieren $\boldsymbol{w}$ genau dann, wenn es Konstanten k_1 und k_2 gibt, so dass

$$k_1\boldsymbol{u} + k_2\boldsymbol{v} = \boldsymbol{w} \neq \mathbf{0}.$$

Es ist im Allgemeinen von Interesse, eine minimale Projektion zu finden: Finde eine minimale Projektion, nimm noch eine Konstante k weg, dann ist es die *größte Einheit*, die aus sich heraus, egal wie, nichts mehr projizieren kann außer $\mathbf{0}$. Die Idee des Algorithmus von Strassen-25 eignet sich zum Finden von Kernen.

3.3 Effiziente Verknüpfung

Matrizen bilden keinen wohl geformten Körper. Es ist daher von Interesse zu untersuchen, wie sich die Struktur der Matrix effizient verknüpfen lässt unabhängig von der konkreten Verknüpfung. Eine mögliche Vorgehensweise für das effiziente Verknüpfen ist die folgende:

1. Halte die Wertebereiche von Elementen einer Matrix in ihrer Größe klein.
2. Finde die Menge aller Kerne.
3. Projiziere mit Hilfe der Kerne das Ergebnis der konkreten Verknüpfung.
4. Vergrößere den Wertebereich um ein Delta.
5. Wiederhole Schritte 1 - 4 für die Projektion mit dem neuen Wertebereich.

Mit diesem Vorgehen bleibt das Finden aller Kerne für einen kleinen Wertebereich lösbar. Daher lassen sich alle Ergebnisse für den kleinen Wertebereich mit Hilfe aller Kerne in dem konkreten Raum projizieren. Je öfter dieses Vorgehen wiederholt und die Wertebereiche vergrößert werden, desto aufwendiger wird es.

Kapitel 4

Zusammenfassung

In dieser Arbeit werden die grundlegenden Strukturen der Algebra wie Vektoren und Matrizen definiert. Sie werden als Strukturen verstanden, die durch Verknüpfungen miteinander verbunden sind. Vektoren sind dabei eine besondere Struktur. Sie bilden einen wohl geformten Körper. Es wird gezeigt, dass die charakteristischen Eigenschaften eines wohl geformten Körpers gebildet werden durch: (1) die Verknüpfungen: Addition und Multiplikation und (2) den Wertebereich. Vektoren bilden einen wohl geformten Körper. Es wird gezeigt, dass die 1 und die wichtigste Zahl ist und dass die binären Zahlen $\{0, 1\}$ die wichtigsten Zahlen sind. Die Algebra ist Grundlage für die Analyse, da die Analyse ohne Wertebereiche nicht möglich ist.

In Kapitel 3 wird die Matrixmultiplikation und der Algorithmus von Strassen zur effizienten Multiplikation von Matrizen vorgestellt. Der Strassen-Algorithmus multipliziert zwei $n \times n$ Matrizen A und B unter Verwendung von nur 7 Matrixmultiplikationen anstelle der üblichen 8. Die Laufzeit des Algorithmus wird durch die Rekurrenzgleichung $T(n) = 7 \cdot T\left(\frac{n}{2}\right) + c \cdot n^2$ beschrieben. Der Algorithmus von Strassen hat damit eine Laufzeit von $O(n^{2.8074})$, welche analysiert wird. Der Algorithmus wird in einem Pseudocode vorgestellt. Er unterteilt die Matrizen in 4 Teilmatrizen und berechnet 7 Produkte, die dann zur Berechnung der 4 Blöcke der Ergebnismatrix verwendet werden.

Eine Methode zur Reduzierung der Matrixmultiplikationen von 7 auf 5 wird beschrieben, was die Laufzeit auf $O(n^{2.3219})$ senkt. Diese Optimierung nutzt die Diagonale der Matrix B und erfordert vorherige Berechnungen der Produkte. Der Algorithmus Strassen-25 setzt diese Methode um. Das *optimale Teile-und-Herrsche Prinzip* wird vorgestellt und gezeigt, dass es die effizienteste Methode ist, um Probleme zu lösen, wenn die Probleminstanz bei jedem rekursiven Aufruf halbiert wird. Es wird gezeigt, dass schneller als mit Laufzeit $O(n^{2.3219})$ zwei $n \times n$ Matrizen *nicht* multipliziert werden können.

4.1 Ausblick

In Kapitel 2.2 wurden Verknüpfungen wie die Addition und die Multiplikation beschrieben. Möchte man Strukturen effizient verknüpfen, sind sie in ihrem Schwerpunkt zu verknüpfen. Daher lohnt sich das Suchen des Schwerpunkts für Strukturen in einer konkreten Verknüpfung.

Matrizen bilden keinen wohl geformten Körper. Vektoren bilden einen wohl geformten Körper. In der Verknüpfung der Multiplikation sind Matrizen aufwendiger in der Verknüpfung. Vektoren lassen sich effizienter verknüpfen in der Multiplikation. Es lohnt sich also Strukturen zu gebrauchen, die einen wohl geformten Körper bilden und für diese effiziente Algorithmen zu entwickeln.

In Kapitel 3 wird der Algorithmus Strassen-25 vorgestellt, der sich eignet für das Finden von Kernen. Es wird eine Vorgehensweise beschrieben, mit der für angemessen große Wertebereiche das Finden aller Kerne und das Projizieren aller Ergebnisse effizient möglich ist. Je öfter der Wertbereich erhöht wird, desto aufwendiger wird dieses Vorgehen.

Es wird für das Deep-Learning die Alternative beschrieben, die Gewichtsmatrix W_i der Größe $n \times n$ als n Vektoren der Größe $1 \times n$ aufzufassen und n Vektormultiplikationen zu verwenden. Wenn dann k Vektoren gleichzeitig gelernt werden, hat das Lernen bezogen auf die Multiplikation eine Laufzeit von $k \cdot O(n^2)$. Wie groß ist k zu wählen, dass $k \cdot O(n^2)$ immer noch schneller als $O(n^{2.3219})$ von Strassen-25 für die Matrixmultiplikation ist? Damit die k-fache Vektormultiplikation effizienter ist, muss gelten:

$$k \cdot O(n^2) < O(n^{2.3219}).$$

Unter Berücksichtigung der Konstanten c_1 und c_2:

$$k \cdot c_1 \cdot n^2 < c_2 \cdot n^{2.3219}.$$

Dies führt zu der Bedingung:

$$k < \frac{c_2}{c_1} \cdot n^{0.3219}.$$

Da $n^{0.3219}$ mit wachsendem n zunimmt, wird die obere Schranke für k größer, je größer n wird. Für unterschiedliche Werte von n ergeben sich damit beispielhaft folgende Schranken für k:

$$\begin{aligned} n = 1000: &\quad k < 1000^{0.3219} \approx 9, \\ n = 5000: &\quad k < 5000^{0.3219} \approx 15, \\ n = 10000: &\quad k < 10000^{0.3219} \approx 19, \\ n = 1000000: &\quad k < 1000000^{0.3219} \approx 85. \end{aligned}$$

Als abschließender Ausblick wird das Problem des gewichteten perfekten Matchings betrachtet. Wenn die Kanten Gewichte haben (z.B. Kosten), sucht man ein perfektes Matching mit minimalen Kosten. Der HUNGARIAN Algorithmus löst dieses Problem mit einer Laufzeit von $O(n^3)$, für einen vollständigen bipartiten Graphen mit n Knoten auf beiden Seiten V_1 und V_2. Es ist zu prüfen, ob es für dieses Problem einen Schwerpunkt gibt und wie es dann im Schwerpunkt lösbar ist. Abbilden lässt sich das Problem auf Vektoren mit dem Wertebereich der binären Zahlen. Gibt es mit dem Schwerpunkt ein rekursives Verfahren, welches zu einer effizienteren Lösung führt?