UNIKASSEL VERSITÄT | ELEKTROTECHNIK INFORMATIK

Bachelorarbeit

Implementierung einer Übersetzung Neuronaler Netze in I/O-Äquivalente Endliche Automaten

Laurin Pöppe

[redacted]

Betreuer	Eric Alsmann
1. Gutachter	Prof. Dr. Martin Lange
2. Gutachter	Prof. Dr. Peter Zipf
Fachgebiet	Theoretische Informatik / Formale Methoden
Eingereicht am	16. April 2024

Eigenständigkeitserklärung

Hiermit bestätige ich, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe (dies beinhaltet Large Language Models). Die Stellen der Arbeit, die dem Wortlaut oder dem Sinn nach anderen Werken (dazu zählen auch Internetquellen) entnommen sind, wurden unter Angabe der Quelle kenntlich gemacht.

Laurin Pöppe

Inhaltsverzeichnis

1 Einführung

Neuronale Netze sind ein verbreitetes Konzept des maschinellen Lernens. Ein Nachteil dieser Netzes ist aber, dass sie eine Black Box sind. Es ist also schwer nachzuvollziehen, auf welche Weise sie zu ihren Ergebnissen kommen, was Probleme für die Verifikation aufwirft.
Der Ansatz, welcher mit dieser Arbeit verfolgt wird, ist es, Neuronale Netze in endliche Automaten zu überführen, sodass die Netze durch die Automatentheorie besser verstanden werden können.

Sälzer et al. haben in einer Veröffentlichung [7] bereits bewiesen, dass eine Konstruktion eines endlichen, IO-äquivalenten Automaten zu einem Neuronalen Netz möglich ist. Diese Arbeit baut darauf auf und stellt eine direktere und somit schnellere Konstruktion sowie eine praktische Implementierung dieser vor. Diese Konstruktion verwendet Activation Pattern, wie vorgestellt von Sälzer und Lange [8], um das Neuronale Netz als Lineares Programm darzustellen. Zur Konstruktion des Automaten aus dem Linearen Programm wird der Ansatz bezüglich der Automatenkonstruktion von Gleichungen und Ungleichungen von Ganzzahlen von Wolper und Boigelot [10] verwendet und für rationale Zahlen und mehrere Gleichungen erweitert.

In Abschnitt 2 werden die grundlegenden Begriffe, welche für diese Arbeit benötigt werden, definiert und erläutert. In Abschnitt 3 folgt das theoretische Konzept der Konstruktion; unterteilt in die Konstruktion des Linearen Programmes aus dem Neuronalen Netz in Abschnitt 3.1 und die Konstruktion des Automaten aus dem Linearen Programm in Abschnitt 3.2. Danach wird in Abschnitt 4 auf die praktische Implementierung der Konstruktion eingegangen, bevor in Abschnitt 5 eine abschließende Diskussion der Ergebnisse dieser Arbeit erfolgt.

2 Grundlegendes

In diesem Abschnitt werden die für diese Arbeit grundlegenden Begriffe definiert und erläutert.

2.1 Mathematische Notationen

In dieser Arbeit werden Vektoren mit Vektorpfeilen $\vec{v}$ gekennzeichnet und Matrizen mit Großbuchstaben M bezeichnet. Die Notation des Skalarprodukts zweier Vektoren ist $\langle \vec{v}_1, \vec{v}_2 \rangle$.
Bei der Anwendung einer unären Funktion auf einen Vektor wird die Funktion elementweise angewandt.

2.2 Neuronale Netze

Ein Neuronales Netz [6] [3] besteht aus einer Eingabe- sowie einer Ausgabeschicht und mehreren versteckten Schichten dazwischen. Schichten bestehen wiederum aus mehreren Knoten. Die Knoten einer Schicht sind dabei mit allen Knoten der vorherigen Schicht verbunden. Eine Ausnahme sind die Knoten der Eingabeschicht, welche jeweils mit den Eingaben des Neuronalen Netzes verbunden sind. Die Knoten haben einen Bias und zu jeder Eingabe, die sie erhalten, ein zugehöriges Gewicht. Somit ist der j-te Knoten $N_{i,j}$ der i-ten Schicht L_i ein Tupel der Form $N_{i,j} = (\vec{w}, b) \in \mathbb{Q}^m \times \mathbb{Q}$, wobei m der Anzahl der Knoten der Schicht L_{i-1} entspricht. Der Wert eines Knotens $N_{i,j}$ ergibt sich aus $v_{i,j} = \phi(\langle \vec{w}, \vec{v}_{i-1} \rangle + b)$, wobei ϕ eine Aktivierungsfunktion ist. Sei k die Anzahl der Schichten im Neuronalen Netz. Für jede Schicht L_i lässt sich aus den Gewichten aller Knoten der Schicht eine Gewichtsmatrix $W_i \in \mathbb{Q}^{n \times m}$ bilden, wobei n der Anzahl von Knoten der Schicht L_i und m der Anzahl der Knoten der Schicht L_{i-1} entspricht. Bei W_1, also der Gewichtsmatrix der Eingabeschicht, entspricht m der Anzahl der Eingaben des Neuronalen Netzes. Die Biases einer Schicht L_i lassen sich ebenfalls zu einem Biasvektor $\vec{b}_i \in \mathbb{Q}^n$ zusammenfassen.
Jeder Schicht ist eine Aktivierungsfunktion zugeordnet. Es gibt viele verschiedene Aktivierungsfunktionen, von denen in dieser Arbeit die Identitätsfunktion für die Ausgabeschicht und die Rectified Linear Unit (ReLU)-Funktion für die restlichen Schichten verwendet werden: $\mathrm{ReLU}(x) = \max(0, x)$ und $\mathrm{Id}(x) = x$. Der Ergebnisvektor einer Schicht L_i berechnet sich nun aus $\vec{v}_i = \mathrm{ReLU}(W_i \vec{v}_{i-1} + \vec{b}_i)$ für $2 \leq i \leq k-1$. Für L_1 wird der Vektor $\vec{x}$ aus den Eingaben des Neuronalen Netzes statt $\vec{v_{i-1}}$ verwendet und für $i = k$ die Identitätsfunktion statt der ReLU-Funktion. Durch das aufeinanderfolgende Berechnen der Ergebnisvektoren ergibt sich $\vec{v}_k$ als Ergebnisvektor und N als Funktion des Neuronalen Netzes bezüglich des Eingabevektors $\vec{x}$ durch:

$$N(\vec{x}) = \mathrm{Id}(W_k(\mathrm{ReLU}(W_{k-1}(\ldots \mathrm{ReLU}(W_1 \vec{x} + \vec{b}_1) \ldots) + \vec{b}_{k-1}) + \vec{b}_k))$$

Die IO-Relation eines Neuronalen Netzes N ist:

$$R_N = \{(x_1, \ldots, x_n, y_1, \ldots, y_m) \mid N(x_1, \ldots x_n) = (y_1, \ldots, y_m)\}$$

2.3 Activation Pattern

Die Berechnung der Ergebnisvektoren in Neuronalen Netzen, wie sie im vorigen Abschnitt beschrieben wurde, ist nahezu eine Linearkombination aus den Eingaben des Neuronalen Netzes. Um die Berechnung zu einer Linearkombination zu vereinfachen, muss die ReLU-Funktion aus der Gleichung

entfernt werden, ohne das Ergebnis zu verfälschen. Diese Arbeit verwendet dazu sogenannte Activation Pattern [8].

Aus der ReLU-Funktion folgt als Eigenschaft für die ReLU-Knoten, dass ihre Ausgabe Null ist, falls ihre Berechnung einen negativen Wert ergibt. Dadurch haben solche Knoten keinen Einfluss auf Knoten folgender Schichten. Man kann diese Knoten somit auch als deaktiviert oder ReLU-Off bezeichnen. Knoten mit einer positiven Ausgabe heißen entsprechend aktiviert oder ReLU-On. Diese Kategorisierung wird im Folgenden Aktivierungszustand genannt.
Ein Activation Pattern ist eine Funktion $A : K \to \{\text{ReLU-On}, \text{ReLU-Off}\}$, welche die Knoten K eines Neuronalen Netzes auf ihren Aktivierungszustand abbildet. In Abbildung 1 ist an einem kleinen Neuronalen Netz ohne Biases gezeigt, wie sich die Aktivierungszustände bezüglich einer bestimmten Eingabe ergeben. Auf der rechten Seite ist in den Knoten zu sehen, zu welchem Wert sie sich auswerten. Da sich der unterste Knoten zu -2 auswerten würde und der Wert durch die ReLU-Funktion auf 0 gesetzt wird, gilt dieser Knoten als ReLU-Off, während der linke und obere Knoten positive Ergebnisse haben und entsprechend ReLU-On sind.
Der Knoten der Ausgabeschicht gilt nicht als ReLU-Off, da Knoten der letzten Schicht die Identitätsfunktion verwenden.

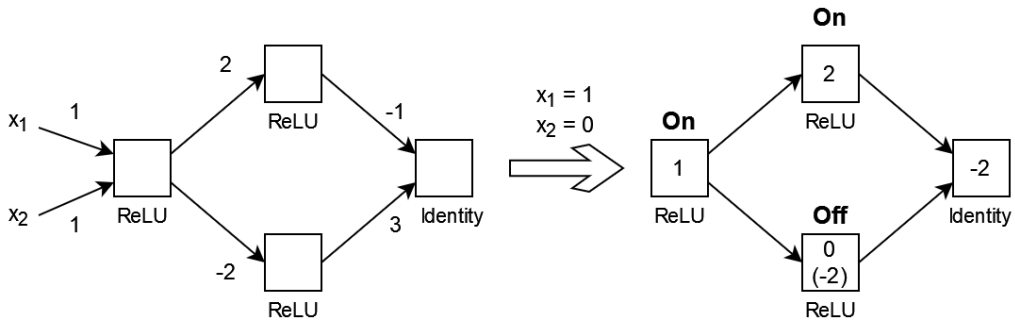

Abbildung 1: Beispiel eines Activation Patterns bzgl. eines Neuronalen Netzes und einer Eingabe

Die Anwendung des Activation Patterns funktioniert im Folgenden so, dass falls der j-te Knoten der i-ten Schicht $n_{i,j}$ ReLU-Off ist, die j-ten Gewichte aller Knoten der folgenden Schicht L_{i+1} auf Null gesetzt werden. Auf diese Weise können die Werte für ReLU-Off Knoten weiterhin berechnet werden, aber ihre Ergebnisse wirken sich weiterhin nicht auf die Ergebnisse der folgenden Knoten aus.

Definition. *Ein Neuronales Netz N beschränkt auf ein Activation Pattern A wird im Folgenden als $N|_A$ bezeichnet. Die zugehörige Relation ist* $R_{N|_A} = \{(x_1, \ldots, x_n, y_1, \ldots, y_m) \in R_N \,|\, \forall i, j : (A(N_{i,j}) = \textit{ReLU-On} \Rightarrow v_{i,j} \geq 0) \wedge (A(N_{i,j}) = \textit{ReLU-Off} \Rightarrow v_{i,j} \leq 0)\}$. *Im Vergleich zu der Relation R_N ohne Beschränkung wird also für jeden ReLU-Knoten des Neuronalen Netzes eine Bedingung hinzugefügt, welche sicherstellt, dass das Activation Pattern eingehalten wird.*

2.4 Mehrspurige Automaten

Ein mehrspuriger Automat (MTA) [5]ist ein Tupel $A = (Q, \Sigma^n, q_0, \delta, F)$. Dabei ist Q eine endliche Menge von Zuständen, $\Sigma^n \subseteq \underbrace{\Sigma \times \ldots \times \Sigma}_{n\text{-mal}}$ ein Alphabet mit n Spuren (wobei Σ das zugrundeliegende Alphabet einer einzelnen Spur ist), $q_0 \in Q$ der Startzustand, $\delta \subseteq Q \times \Sigma^n \times Q$ die Transitionsrelation und $F \subseteq Q$ die Menge von Akzeptanzzuständen.
Ein Eingabewort $w = a_0 a_1 \ldots a_k \in (\Sigma^n)^*$ wird von einem MTA akzeptiert, falls für den Lauf $\rho = q_0, q_1, \ldots, q_{k+1}$ auf diesem Wort gilt: q_0 ist der Startzustand des MTA, $(q_i, a_i, q_{i+1}) \in \delta$ für alle i mit $0 \leq i \leq k$ und $q_{k+1} \in F$. Die Sprache eines Automaten $L(A)$ ist definiert als $L(A) = \{w \in (\Sigma^n)^* | A \text{ akzeptiert } w\}$.

Das Produkt zweier Automaten $A_i = (Q_i, \Sigma^n, q_i, \delta_i, F_i)$ für $i = \{1, 2\}$ ist definiert als $A' = (Q_1 \times Q_2, \Sigma^n, (q_1, q_2), \delta', F_1 \times F_2)$ mit

$$((q_1, q_2), \vec{a}, (p_1, p_2)) \in \delta' \Longleftrightarrow (q_1, \vec{a}, q_2) \in \delta_1 \wedge (p_1, \vec{a}, p_2) \in \delta_2.$$

Es gilt $L(A') = L(A_1) \cap L(A_2)$.

2.5 Zahlendarstellung

Als binäre Zahlendarstellung wird in dieser Arbeit das Zweierkomplement [9] verwendet. Die Gewichtung der Bits ist in Abbildung 2 zu sehen. Das höchstwertige Bit ist in dieser Darstellung an erster Stelle und hat ein negatives Gewicht.

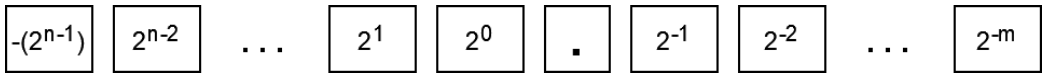

Abbildung 2: Gewichtungen der Bits im Zweierkomplement

Beispiel. $1011.101 \rightarrow -(2^3) + 2^1 + 2^0 + 2^{-1} + 2^{-3} = -4,375$

Jede Zahl x wird also als Tupel (k, l) mit $k \in \mathbb{Z}$ und $l \in [0, 1] \subseteq \mathbb{R}$ dargestellt, sodass $x = k + l$. Die Binärdarstellung mancher Zahlen, z.B. 0,1, ist dabei unendlich lang. In dieser Arbeit werden nur Zahlen mit endlicher Repräsentation betrachtet.
Die Funktionen *dec* und *bin* bezeichnen im Folgenden die Umwandlungen zwischen Dezimalzahlen und deren Darstellung im Zweierkomplement. Bei einem Vektor werden die Funktionen jeweils elementweise angewendet.
Beispiel. $dec(1011.101) = -4,375$ und $bin(-4,375) = 1011.101$

Die Binärdarstellung als Zweierkomplement der Vor- und Nachkommateile k und l haben jeweils eine Mindestlänge. Diese Mindestlänge kann nicht unterschritten werden, aber es ist möglich, die Binärdarstellungen beliebig zu verlängern, ohne die dargestellten Werte zu verändern.
Für den Nachkommateil l ist die Erweiterung trivial, da beliebig Nullen angefügt werden können, welche keinen Einfluss auf den Wert haben.
Bei dem Vorkommateil k muss hingegen zwischen positiven und negativen Zahlen unterschieden werden. Positive Zahlen können dabei, wie Dezimalzahlen auch, durch führende Nullen erweitert werden. Negative Zahlen müssen allerdings durch führende Einsen erweitert werden. Man betrachte das erste Bit b_1 des Zweierkomplements einer negativen Zahl, welches eine 1 ist, da die Zahl negativ ist. Sei i die Anzahl der Vorkommabits, dann ist $-(2^i)$ die Gewichtung des ersten Bits. Hängt man nun ein weiteres 1er Bit b_0 vorne an, so ist dessen Gewichtung $-(2^{i+1}) = 2 \cdot -(2^i)$. Gleichzeitig ändert sich die Gewichtung des Bits b_1 zu 2^i, da es nicht mehr das vorderste Bit ist und somit positiv gewichtet ist. Zusammen ergeben diese beiden Bits $2 \cdot -(2^i) + 2^i = -(2^i)$. Daraus folgt, dass das Anhängen von Einsen an negative Zahlen im Zweierkomplement den Wert der Zahl nicht verändert. Die Werte der restlichen Bits müssen nicht betrachtet werden, da sie gleich bleiben.

Die Menge der binär endlich repräsentierbaren Zahlen ist $\mathbb{F} = \{q \in \mathbb{Q} \mid \exists w \in \{0, 1, .\}^* \text{ mit } q = dec(w)\}$, wobei w eine endliche Darstellung im Zweierkomplement ist.

Da diese Zahlendarstellung zur Kodierung von Eingabeworten eines Automaten verwendet wird, muss die kodierte Zahl auch durch bitweises Einlesen iterativ ermittelbar sein.

Lemma 1. *Sei $w \in \{0, 1, .\}^*$ die Darstellung einer Zahl $n \in \mathbb{N}$ im Zweierkomplement. $dec(w)$ kann durch bitweises Einlesen auf folgende Weise er-*

mittelt werden:

$$dec(w) = n_{|w|}, \ \ wobei\ n_i = \begin{cases} -1 \cdot w_i & i = 1 \\ 2 \cdot n_{i-1} + w_i & sonst \end{cases}$$

Beweis. Per Induktion über Länge von w:

Induktionsanfang: $|w| = 1$
$n_1 = -1 \cdot w_1$.
Für $w = 1$: $dec(1) = -1$ und $n_1 = 1$. Für $w = 0$: $dec(0) = 0$ und $n_1 = 0$.

Induktionshypothese:
$dec(w_1, \ldots, w_i) = n_i$ für alle w mit $|w| = i$ für beliebiges, aber festes i.

Induktionsschritt: $|w| \to |w| + 1$
$w = (w_1, \ldots, w_{|w|}, w_{|w|+1})$. Es gilt $dec(w_1, \ldots w_|w|) = n_|w|$.
Aus der Definition des Zweierkomplements folgt, dass für $(w_1, \ldots, w_|w|)$ die Gewichtung des Bits w_1 $-(2^{|w|-1})$ entspricht. Wird nun ein weiteres Symbol ans Ende angefügt, so ändert sich die Gewichtung zu $-(2^|w|)$. Selbige Änderung erfolgt auch bei allen anderen Bits:

$$\begin{aligned}
\text{vorher: } & -(2^{|w|-1}) \cdot w_1 + 2^{|w|-2} \cdot w_2 + \ldots + 2^1 \cdot w_{|w|-1} + 2^0 \cdot w_{|w|} \\
\text{nachher: } & -(2^{|w|}) \cdot w_1 + 2^{|w|-1} \cdot w_2 + \ldots + 2^2 \cdot w_{|w|-1} + 2^1 \cdot w_{|w|} \\
& = -2 \cdot (2^{|w|-1}) \cdot w_1 + 2 \cdot 2^{|w|-2} \cdot w_2 + \ldots + 2 \cdot 2^1 \cdot w_{|w|-1} + 2 \cdot 2^0 \cdot w_{|w|} \\
& = 2 \cdot (-(2^{|w|-1}) \cdot w_1 + 2^{|w|-2} \cdot w_2 + \ldots + 2^1 \cdot w_{|w|-1} + 2^0 \cdot w_{|w|}) \\
& = 2 \cdot dec(w_1, \ldots, w_{|w|}) \overset{I.H.}{=} 2 \cdot n_{|w|}
\end{aligned}$$

Weiterhin kommt das neue Bit $w_{|w|+1}$ mit Gewicht $2^0 = 1$ hinzu. Somit gilt:

$$dec(w_1, \ldots, w_{|w|+1}) = 2 \cdot n_{|w|} + w_{|w|+1} = n_{|w|+1}$$

□

2.6 Eingabeworte

Diese Zahlendarstellung wird zur Kodierung der Eingabeworte der MTAs verwendet.
Das zugrundeliegende Alphabet zu dem Mehrspur-Alphabet Σ^n ist $\Sigma = \{0, 1, .\}$.
Zunächst wird nur ein einspuriges Eingabewort betrachtet. Die einzelnen Eingabeworte in dieser Arbeit entsprechen Gleitkommazahlen. Eine solche

Zahl wird in ein Eingabewort umgewandelt, indem man ihre Darstellung im Zweierkomplement (siehe Abschnitt 2.5) berechnet, und die Binärsymbole in Symbole des Alphabets Σ übersetzt.

Beispiel. $-1,5 \rightarrow 10.1 \rightarrow [1,0,.,1]$.

Diese einzelnen Worte werden zu einem mehrspurigen Eingabewort zusammengeführt, indem sie so übereinandergelegt werden, dass die Punkte an derselben Stelle liegen. Bei Worten mit unterschiedlich langen Vor- oder Nachkommateilen muss dabei der entsprechende Teil des kürzeren Eingabewortes, wie in Abschnitt 2.5 um Einsen bzw. Nullen erweitert werden, ohne dass sich deren Wert als entprechende Zweierkomplement-Zahl verändert.

Beispiel. Für zwei einspurige Worte $w_1 = [0,1,0,1,1,.,1,1]$ und $w_2 = [1,0,.,1]$ ist das zweispurige Wort:

$$w_{MT} = \begin{bmatrix}0\\1\end{bmatrix}\begin{bmatrix}1\\1\end{bmatrix}\begin{bmatrix}0\\1\end{bmatrix}\begin{bmatrix}1\\0\end{bmatrix}\begin{bmatrix}1\\0\end{bmatrix}\begin{bmatrix}.\\.\end{bmatrix}\begin{bmatrix}1\\1\end{bmatrix}\begin{bmatrix}1\\0\end{bmatrix}$$

Wie zu sehen ist, musste bei w_2 der Vorkommateil um Einsen und der Nachkommateil um Nullen erweitert werden, um dem Format des längeren Wortes w_1 zu entsprechen.

2.7 MTA Relation

Sei A ein MTA über Σ^n mit $\Sigma = \{0,1,.\}$. Dann ist $R_A = \{dec(w) \in \mathbb{F}^n \mid w \in L(A)$ und w ist korrekt kodiert nach Abschnitt 2.6$\}$.

2.8 Lineare Programme

Ein Lineares Programm [1] P ist eine Menge von Linearen Gleichungen und Ungleichungen G der Form $\langle \vec{a}_i, \vec{x} \rangle \bowtie_i b_i$ über einer Variablenmenge X. Dabei sei $\bowtie \in \{=, \leq, \geq\}$. Im Gegensatz zu der gängigen Definition von Linearen Programmen wird hier kein Optimierungsproblem behandelt, sondern nur eine Lösungsmenge für die Variablen gesucht, die alle Gleichungen und Ungleichungen des Linearen Programmes erfüllt. Diese Lösungsmenge ist $\mathbb{L}_P = \{\vec{x} \in \mathbb{Q}^k | \langle \vec{a}_i, \vec{x} \rangle \bowtie_i b_i \forall i\}$. Die Lösungsmenge beschränkt auf binär endlich repräsentierbare Lösungen sei $\mathbb{L}_P|_{\mathbb{F}}$.

3 Automatenkonstruktion

In der Veröffentlichung von Sälzer et al. [7] wurde gezeigt, dass es möglich ist, zu einem Neuronalen Netz einen IO-äquivalenten mehrspurigen Automaten zu bauen. Die induktive Übersetzung des Neuronalen Netzes, welche für

diesen Beweis verwendet wurde, ist allerdings langsam und umständlich.
Das Ziel dieser Arbeit ist eine direktere und damit schnellere Konstruktion mehrspuriger Automaten bezüglich Neuronaler Netze mit gegebenem Activation Pattern. Wie in dem vorigen Abschnitt erwähnt, ermöglicht ein gegebenes Activation Pattern die Vereinfachung des Neuronalen Netzes mit ReLU-Knoten auf lineare Gleichungen ohne die ReLU-Funktion. Dies ermöglicht eine Automatenkonstruktion anhand Linearer Programme, welche sich an der Konstruktion für einen Automaten zu einer Gleichung bzw. Ungleichung natürlicher Zahlen von Wolper und Boigelot [10] orientiert.

3.1 Erstellen eines Linearen Programmes

Für jeden Knoten des Neuronalen Netzes kann – bei gegebenem Activation Pattern – eine lineare Gleichung für den Wert ihrer Ausgabe in Abhängigkeit von den Eingaben des Netzes erstellt werden. Die Koeffizienten zu den Eingabevariablen ergeben sich dabei aus den Gewichten des aktuellen Knotens sowie den Gewichten aller Knoten der vorigen Schichten, ausgenommen derer, die ReLU-Off sind. Neben den Koeffizienten gibt es auch eine Konstante in der Gleichung, die sich aus den Biases der selben Knoten – wiederum unter Berücksichtigung der Gewichte und des Activation Patterns – ergibt.
Das Ziel dieser Gleichungen ist es, Bedingungen zu bilden, sodass diese Bedingungen erfüllt sind, genau dann wenn das Neuronale Netz bei der Eingabe eine bestimmte Ausgabe zurückgibt und dabei das Activation Pattern berücksichtigt. Daraus folgt, dass die Gleichungen der Knoten der Ausgabeschicht auch Variablen für die Ausgaben des Neuronalen Netzes benötigen. Für die Knoten der restlichen Schicht sind exakte Ausgaben nicht relevant, stattdessen aber, ob ihre Werte dem Activation Pattern entsprechen. Dies lässt sich umsetzen, indem Ungleichungen verwendet werden, um die ReLU-On bzw. ReLU-Off Bedingungen sicherzustellen.

Lemma 2. *Sei $N|_A$ ein Neuronales Netz beschränkt auf ein Activation Pattern A. Dann existiert ein Lineares Programm P über der Variablenmenge $\mathcal{X} = \{x_1, \ldots, x_n, y_1, \ldots, y_m\}$, sodass $\mathbb{L}_P = R_{N|_A}$.*

Beweis. Seien ℓ die Anzahl der Schichten, n die Anzahl der Eingabe- und m die Anzahl der Ausgabevariablen des Neuronalen Netzes N, $W_1^A, \ldots, W_\ell^A$ die Gewichtsmatrizen der Schichten von N, auf welche das Activation Pattern bereits angewandt wurde, und $\vec{b}_1, \ldots, \vec{b}_\ell$ die Biasvektoren der Schichten.
Für die Ausgabe von N gilt: $\vec{y} = W_\ell^A(\ldots(W_1^A \cdot \vec{x} + \vec{b}_1)\ldots) + \vec{b}_\ell = W^A \cdot \vec{x} + \vec{b}$ durch Ausmultiplizieren. Daraus lässt sich für jeden Ausgabeknoten eine lineare

Gleichung in Bezug auf die Eingabevariablen erstellen:

$$\begin{aligned} \forall i \text{ mit } 1 \leq i \leq m : y_i &= W^A_{i,1} \cdot x_1 + \ldots + W^A_{i,n} \cdot x_n + b_i \\ \iff -b_i &= W^A_{i,1} \cdot x_1 + \ldots + W^A_{i,n} \cdot x_n - y_i \end{aligned}$$

Dabei ist b_i der Wert an i-ter Stelle des Vektors $\vec{b}$ und wird als Konstante der Gleichung bezeichnet. $W^A_{i,j}$ ist der Wert der Matrix W^A in i-ter Reihe und j-ter Spalte. Die Werte der i-ten Reihe bilden zusammen die Koeffizienten der Gleichung.

Zusätzlich werden noch Nebenbedingungen benötigt, welche sicherstellen, dass jeder ReLU-Knoten bezüglich der Variablenbelegung das Activation Pattern einhält. Die Gleichungen für die ReLU-Schichten k sind dabei $\vec{v}_k = W^A_k(\ldots(W^A_1 \cdot \vec{x} + \vec{b}_1)\ldots)\vec{b}_k = W^{A,k} \cdot \vec{x} + \vec{b}^{A,k}$. Man beachte, dass die Anwendung des Activation Patters auf eine Schicht nach Abschnitt 2.3 erfolgt, indem man die Gewichtsmatrix der folgenden Schicht anpasst. Somit sind die Werte in $\vec{v}_k$ nicht daran angepasst, wodurch negative Werte und auch die Überprüfung der Erfüllung des Activation Patterns möglich sind. Mit diesen Gleichungen lassen sich also die Nebenbedingungen zur Überprüfung für die einzelnen Knoten aufstellen:

$$\forall N_{k,i} : \begin{cases} \langle W^{A,k}_i, \vec{x} \rangle + \vec{b}^{A,k}_i \geq 0 & \text{, falls } A(N_{k,i}) = \text{ReLU-On} \\ \langle W^{A,k}_i, \vec{x} \rangle + \vec{b}^{A,k}_i \leq 0 & \text{, falls } A(N_{k,i}) = \text{ReLU-Off} \end{cases}$$

Zusammen bilden diese Gleichungen und Ungleichungen das Lineare Programm P.

Da die Bedingungen des Linearen Programms P und der IO-Relation des Neuronalen Netzes $R_{N|_A}$ jeweils aus denselben Gleichungen bestehen, gilt $\mathbb{L}_P = R_{N|_A}$ [8]. □

3.2 Konstruktion eines Automaten aus einer linearen Gleichung

In der Veröffentlichung von Wolper und Boigelot [10] wird die Konstruktion von Automaten zu einzelnen Gleichungen bzw. Ungleichungen mit natürlichen Zahlen beschrieben. Im Folgenden wird diese Konstruktion für eine Gleichung mit binär endlich repräsentierbaren rationalen Zahlen erweitert.

Die Gleichung ist entsprechend der Konstruktion des Abschnitts 3.1 von der Form: $c_1 \cdot x_1 + \ldots + c_n \cdot x_n = k$.

Betrachten wir die Konstruktion zunächst beispielhaft für die Gleichung $0,5 \cdot x_1 - x_2 = 1,5$.

Die Zustände des Automaten besitzen jeweils einen Wert, welche den Werten entsprechen, die die Gleichung $0,5 \cdot x_1 - x_2$ annimmt, wobei x_1 und x_2 der Dezimaldarstellung der Transitionssymbole entsprechen, welche vom Startzustand zu diesem Zustand geführt haben.
Die Konstruktion beginnt mit dem sogenannten Vorkommateil in einem Startzustand. Es werden nun alle Zustände erstellt, die durch Transitionen ohne Dot-Symbole erreichbar sind. Dazu werden von jedem Zustand aus alle möglichen binären Mehrspursymbole und deren entsprechende Transitionen betrachtet. Falls für einen Folgezustand ein Wert berechnet wird, für welchen es bereits einen Zustand gibt, wird kein neuer Zustand erstellt, sondern nur eine Transition zu dem bereits existierenden Zustand hinzugefügt. In Abbildung 3 sind diese Transitionen vom Startzustand aus zu sehen.

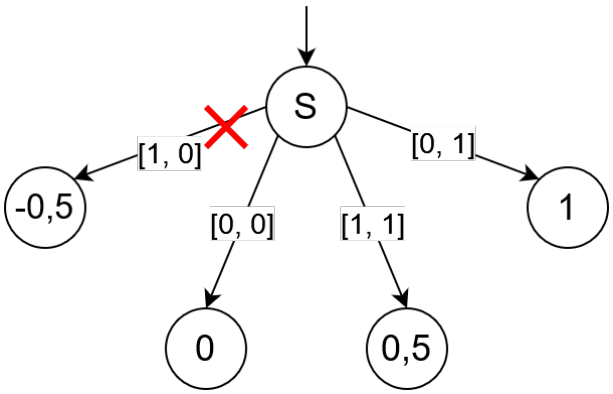

Abbildung 3: Beispiel: Transitionen vom Startzustand für $0,5 \cdot x_1 - x_2 = 1,5$

Man betrachte die Transition mit Mehrspursymbol $[1,1]$. Der Wert des Zielzustands entsprechend der bisher gelesenen Worte $w_{1,1} = 1$ und $w_{2,1} = 1$ ist $v = 0,5 \cdot dec(1) - dec(1) = 0,5 \cdot (-1) - (-1) = 0,5$.
Um bei der Betrachtung der Transitionen von einem Zustand aus nicht nachvollziehen zu müssen, welche Symbole bis zu diesem Zustand bisher gelesen wurden, wird die Formel zum bitweisen Einlesen der Eingabeworte aus Lemma 1 verwendet. So gilt für eine Transition von dem Zustand mit Wert v mit Bitvektor $\vec{b}$: $u = 2 \cdot v + \langle \vec{c}, \vec{b} \rangle$, sodass u der Wert des Folgezustands ist.

Betrachtet man nun die Transitionen ausgehend von dem Zustand mit Wert $-0,5$, so fällt auf, dass diese ebenfalls kleiner oder gleich $-0,5$ sind: z.B.

$$\begin{aligned}[1,0] \rightarrow w_{1,2} = 11 \text{ und } w_{2,2} = 00 \rightarrow v &= 0,5 \cdot dec(11) - dec(00) \\ &= 0,5 \cdot (-1) - 0 = -0,5\end{aligned}$$

Somit ist es von dem Zustand mit Wert -1 nicht möglich, zu dem gesuchten Wert $1,5$ zu kommen, weshalb der Zustand verworfen werden kann. Es ist nötig, bei solchen Zuständen abzubrechen, damit der Automat endlich groß bleibt.

Die Konstruktion des Vorkommateils wird so fortgesetzt, bis die Transitionen jedes Zustands betrachtet wurden. In Abbildung 4 ist der Vorkommateil zu der Beispielgleichung zu sehen.

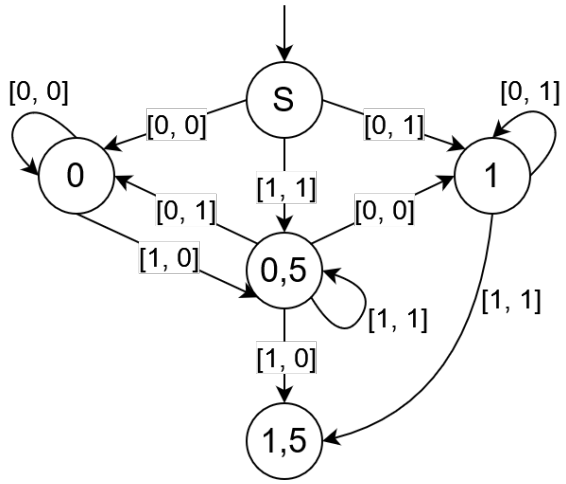

Abbildung 4: Beispiel: Automat Vorkommateil für $0,5 \cdot x_1 - x_2 = 1,5$

Nachdem der Vorkommateil konstruiert wurde, folgt der Übergang zum Nachkommateil. In diesem funktioniert der Ansatz zum bitweisen Einlesen der Eingabeworte nicht mehr wie im Vorkommateil. Dort war es möglich, da die Gewichtung der bisher gelesenen Zahlen sich verändert, während der Gewichtung der aktuell gelesenen Zahl immer 1 ist. Nach dem Komma ist dies andersherum, sodass die Gewichtung der bisher gelesenen Zahlen gleich bleibt und die Gewichtung der aktuellen Zahl von der Anzahl der bisher gelesenen Bits abhängig ist. Durch diese Abhängigkeit ist diese Vorgehensweise für einen endlichen Automaten nicht möglich, weshalb für den Nachkommateil ein anderer Ansatz nötig ist.
Die Zustandswerte im Nachkommateil beschreiben nun, wie viel Abstand zu dem gewünschten Wert noch besteht. Dazu wird beim Übergang durch eine eingelesene Dot-Transition – also einer Transition deren Mehrspursymbol nur aus Dots besteht – zunächst die Konstante der Gleichung mit dem Zustandswert des Vorkommateilszustands verrechnet. Weiterhin muss beachtet

werden, dass sich mit jedem weiterem Bit im Nachkommateil die Gewichtung der Bits halbiert, weshalb der Wert verdoppelt werden muss, um den Abstand daran anzupassen.
Der Übergang von dem Zustand mit Wert $1,5$ führt also zu dem Zustand mit dem Wert $2 \cdot (1,5 - 1,5) = 0$. Dabei ist zu beachten, dass keine Transitionen von Zuständen des Nachkommateils zu Zuständen des Vorkommateils erfolgen. Somit wird ein neuer Zustand mit Wert 0 erzeugt, obwohl es diesen bereits im Vorkommateil gibt. Dieser Zustand ist auch ein Akzeptanzzustand, da der Abstand zur Lösung 0 beträgt und somit eine Lösung gefunden wurde. Dieser Übergang wird von allen Vorkommateilzuständen – ausgenommen dem Startzustand – durchgeführt. Bei der Beispielgleichung werden die Übergänge der anderen Zustände allerdings abgebrochen.
Für die Transitionen im Nachkommateil wird der Ausgangszustand mit dem Produkt der gelesenen Bits und entsprechenden Koeffizienten verrechnet und der verbleibende Abstand verdoppelt, da das nächste Bit nur halb so viel Gewicht hat. Somit ist z.B. der Wert des Folgezustands vom Zustand mit Wert 0 mit gelesenen Bits $[0, 1]$: $2 \cdot (0 + (0,5 \cdot 0 + (-1) \cdot 1) = -2$. Auch bei diesem Zustand wird abgebrochen, da alle Folgezustände einen Wert kleiner oder gleich -2 hätten und somit 0 nicht erreicht werden kann.
Wie im Vorkommateil wird die Konstruktion fortgesetzt, bis die Transitionen von allen Zuständen des Nachkommateils aus durchgeführt wurden. In Abbildung 5 ist der fertige Automat zur Beispielsgleichung zu sehen.

Es folgt nun eine genauere Definition der Konstruktion, wie sie anhand des Beispiels vorgestellt wurde sowie Beweise zu deren Korrektheit:

Sei $(c^+, c^-) \in \mathbb{Q}_{\geq 0} \times \mathbb{Q}_{\leq 0}$ das Tupel der Summe aller positiven bzw. negativen Koeffizienten der Gleichung mit

$$(c^+, c^-) = (\sum_{i=1}^{n} \max(0, c_i), \sum_{i=1}^{n} \min(0, c_i)).$$

Sei der MTA zunächst $A = (\{q_0\}, \Sigma^n, q_0, \{\}, \{\})$ mit $\Sigma = \{0, 1, .\}$.

Ein Eingabewort w zu dem MTA sind wie in Abschnitt 2.6 beschrieben die zu einem Mehrspurwort kombinierten Werte der Variablen der Gleichung in Zweierkomplementdarstellung. Entsprechend wird in jedem Schritt des Automaten ein Bit jeder Variable eingelesen. Der Automat ist in einen Vor- und einen Nachkommateil unterteilt. Der Startzustand gehört zum Vorkommateil. Der Wechsel in den Nachkommateil erfolgt über sogenannte Dot-Transitionen $\vec{d}$, also Transitionen bei denen der eingelesene Bitvektor $\vec{b} = [., \ldots, .]$ ist. Ein Wechsel vom Nach- in den Vorkommateil ist nicht möglich. Akzeptanzzustände existieren nur im Nachkommateil.

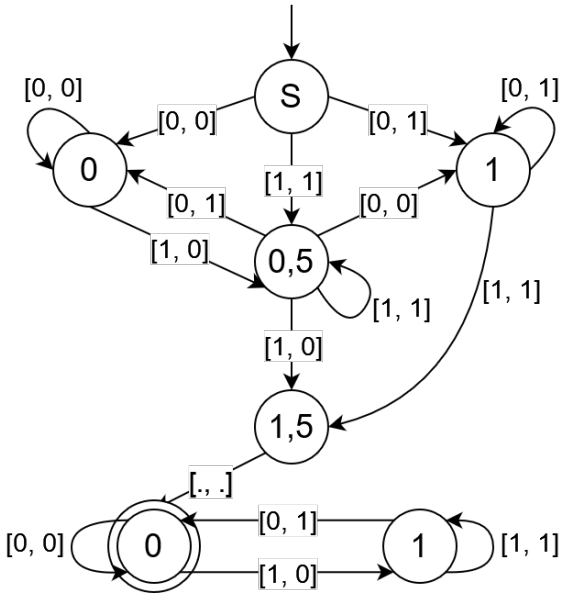

Abbildung 5: Beispiel: Kompletter Automat für $0,5 \cdot x_1 - x_2 = 1,5$

Jedem Zustand, ausgenommen dem Startzustand, wird ein Wert v zugewiesen. Im Vorkommateil entspricht v dem Wert der Gleichung bezüglich der bisher gelesenen Bits der Variablen. Lemma 1 besagt, dass $dec(w)$ bei bitweisem Einlesen durch

$$n_i = \begin{cases} -1 \cdot w_i & i = 1 \\ 2 \cdot n_{i-1} + w_i & sonst \end{cases}$$

ermittelt werden kann. Entsprechend wird diese Formel für die Berechnung von Folgezuständen bei Transitionen des Vorkommateils verwendet.
Die Transitionen ausgehend vom Startzustand erhalten die ersten Bits der Variablen als Bitvektor $\vec{b} \in \Sigma^n$, weshalb die Gleichung für die Werte dessen Folgezustände $-1 \cdot \langle \vec{c}, \vec{b} \rangle$ ist, wobei $\vec{b}$ der eingelesene Bitvektor ist. Die Gleichung für die Werte der Folgezustände aller anderen Zustände des Vorkommateils mit Wert v ist entsprechend: $2 \cdot v + \langle \vec{c}, \vec{b} \rangle$.
Bei jedem Zustand wird zunächst für jeden möglichen Bitvektor eine Transition und ein Folgezustand erstellt. Sollte ein Zustand mit gleichem Wert bereits existieren, so zeigt die Transition auf diesen und es wird kein weiterer Zustand mit selbem Wert erstellt.

Zustände und Transitionen werden außerdem abgebrochen, wenn der Zustandswert eine Abbruchbedingung erfüllt. Sei k die Konstante der Gleichung und v der Wert des zu prüfenden Zustands. Dann sind die Abbruchbedin-

gungen:

$$\begin{aligned} &1.\ v < k \wedge v + c^+ \leq 0 \text{ oder} \\ &2.\ v > k \wedge v + c^- \geq 0 \end{aligned}$$

Lemma 3. *Sei $\langle \vec{c}, \vec{x} \rangle = k$ die gegebene Gleichung G und v der Wert des betrachteten Zustands im Vorkommateil des Automaten. Sei (c^+, c^-) das Tupel aus der Summe aller positiven bzw. negativen Koeffizienten aus $\vec{c}$. Sei die Abbruchbedingung des Nachkommateils nach einer Dot-Transition nicht erfüllt.*

Dann gilt für die Abbruchbedingung des Vorkommateils:
Wenn

$$(v < k \wedge v + c^+ \leq 0) \vee (v > k \wedge v + c^- \geq 0)$$

dann ist es von dem Zustand mit Wert v selbst mit den bestmöglichen Eingabewerten in Transitionen zu Folgezuständen nicht möglich, zu einem Akzeptanzzustand zu gelangen.

Beweis. Die Disjunktion der Abbruchbedingung lässt sich durch eine Fallunterscheidung auflösen:

Fall 1: $(v < k \wedge v + c^+ \leq 0)$
Damit k von dem Zustand mit Wert v aus noch erreichbar ist, muss einer der Folgezustände einen Wert haben, welcher größer als v ist, da v aktuell kleiner als k ist. Mit der Formel zur Berechnung der Folgezustände bedeutet das: $2 \cdot v + \langle \vec{b}, \vec{c} \rangle > v$ für einen Bitvektor $\vec{b}$.
Aus der Annahme von Fall 1 folgt allerdings:

$$\begin{aligned} v + c^+ &\leq 0 \\ 2 \cdot v + c^+ &\leq v \end{aligned}$$

Da c^+ die Summe aller positiven Koeffizienten aus $\vec{c}$ ist, gilt $\langle \vec{b}, \vec{c} \rangle \leq c^+$ und somit $2 \cdot v + \langle \vec{b}, \vec{c} \rangle \leq v$. Also ist von v kein Akzeptanzzustand erreichbar, wenn die Abbruchbedingung erfüllt ist.

Fall 2: $(v > k \wedge v + c^- \geq 0)$ analog zu Fall 1. □

Die beschriebene Konstruktion wird durchgeführt, bis alle möglichen Transitionen aller Zustände des Vorkommateils betrachtet und konstruiert bzw. abgebrochen wurden.

Als nächstes werden von allen Zuständen des Vorkommateils – ausgenommen dem Startzustand – Dot-Transitionen konstruiert, um in den Nachkommateil überzugehen. Die Werte von Nachkommateilzuständen stehen im

Gegensatz zum Vorkommateil für den Abstand, der noch überwunden werden muss, um das korrekte Ergebnis zu erreichen. Um zu berücksichtigen, dass die Gewichtung der Bits sich mit jeder gelesenen Stelle halbiert, wird der noch zu überwindende Abstand, welcher den Wert des Zustands ausmacht, nach einer Transition verdoppelt.
Die Übergangsgleichungen der Dot-Transitionen $(u, \vec{d}, v)$ sind entsprechend $v = 2 \cdot (u + k)$. Die Gleichungen für Transitionen zwischen Nachkommateilzuständen sind: $v = 2 \cdot (u + \langle \vec{b}, \vec{c} \rangle)$.

Die Abbruchbedingungen für einen Zustand des Nachkommateils mit Wert v sind

1. $v + 2 \cdot c^+ \leq 0$ oder
2. $v + 2 \cdot c^- \geq 0$

Lemma 4. *Sei $\langle \vec{c}, \vec{x} \rangle = -k$ die gegebene Gleichung G und v der Wert des betrachteten Zustands im Nachkommateil des Automaten. Sei (c^+, c^-) das Tupel aus der Summe aller positiven bzw. negativen Koeffizienten aus $\vec{c}$.*

Dann gilt für die Abbruchbedingung des Nachkommateils:
Wenn

$$(v + 2 \cdot c^+ \leq 0) \vee (v + 2 \cdot c^- \geq 0)$$

dann ist es von dem Zustand mit Wert v selbst mit den bestmöglichen Eingabewerten in Transitionen zu Folgezuständen nicht möglich, zu einem Akzeptanzzustand zu gelangen.

Beweis. Die Disjunktion der Abbruchbedingung lässt sich durch eine Fallunterscheidung auflösen:

Fall 1: $(v + 2 \cdot c^+ \leq 0)$
Da $c^+ \geq 0$ gilt, folgt aus der Annahme $v \leq 0$ durch Widerspruch.
Entsprechend ist es von dem Zustand mit Wert v nur möglich, einen Akzeptanzzustand, also einen Zustand mit Wert 0, zu erreichen, wenn ein Folgezustand einen Wert hat, welcher größer als v ist. Mit der Formel zur Berechnung der Werte von Folgezuständen im Nachkommateil bedeutet das: $2 \cdot (v + \langle \vec{b}, \vec{c} \rangle) > v$ für einen Bitvektor $\vec{b}$.
Aus der Annahme von Fall 1 folgt allerdings:

$$\begin{aligned} v + 2 \cdot c^+ &\leq 0 \\ 2 \cdot v + 2 \cdot c^+ &\leq 0 \\ 2 \cdot (v + c^+) &\leq 0 \end{aligned}$$

Da c^+ die Summe aller positiven Koeffizienten aus $\vec{c}$ ist, gilt $\langle \vec{b}, \vec{c} \rangle \leq c^+$ und somit $2 \cdot (v + \langle \vec{b}, \vec{c} \rangle) \leq v$. Also ist von v kein Akzeptanzzustand erreichbar, wenn die Abbruchbedingung erfüllt ist.

Fall 2: $(v + 2 \cdot c^- \geq 0)$ analog zu Fall 1.

□

Es ist möglich, dass ein Zustand des Vorkommateils eine Abbruchbedingung erfüllt, ein Folgezustand davon im Nachkommateil nach einer Dot-Transition aber keine Abbruchbedingung des Nachkommateils erfüllt. In diesem Fall darf der Zustand des Vorkommateils nicht abgebrochen werden. Somit muss im Vorkommateil auch die Abbruchbedingung des Nachkommateils geprüft werden.
Ein Beispiel ist die Gleichung $x = -0,5$. Wird als erstes Bit eine Eins gelesen, so ist der Wert Folgezustand des Startzustands -1. Somit ist die Abbruchbedingung des Vorkommateils erfüllt: $-1 < -0,5 \wedge -1+1 \leq 0 \Rightarrow$ True. Betrachtet man aber die Dot-Transition des Zustands, so ist Wert des Folgezustands im Nachkommateil $2 \cdot (-1 - (-0,5)) = -1$, welcher keine Abbruchbedingung des Nachkommateils erfüllt:

$$(-1 + 2 \cdot 1 \leq 0) \vee (-1 + 0 \geq 0) \Rightarrow (1 \leq 0) \vee (-1 \geq 0) \Rightarrow \text{False}$$

Die Akzeptanzbedingung für einen Zustand des Nachkommateils mit Wert v ist $v = 0$, da der Wert des Zustands dem Abstand zum korrekten Ergebnis entspricht.

Die Konstruktion des Automaten ist abgeschlossen, sobald alle möglichen Transitionen von allen Zuständen des Nachkommateils konstruiert oder abgebrochen wurden.

Lemma 5. *Sei $c_1x_1 + \ldots + c_nx_n = k$ eine Lineare Gleichung G, dann existiert ein MTA A, sodass $L(A) = bin(\mathbb{L}_G|_{\mathbb{F}})$.*

Beweis. Der Beweis erfolgt durch den Beweis der Korrektheit und den Beweis der Vollständigkeit.

Korrektheit: $w \in L(A) \Rightarrow dec(w) \in \mathbb{L}_G$
Aus der Konstruktion folgt, dass der Wert v_q eines Zustands q im Vorkommateil, welcher durch das bisher gelesene Teilwort w_i erreicht wurde, dem Zustand der Gleichung bezüglich dieses Teilworts entspricht: $v_q = \langle dec(w_i), \vec{c} \rangle$. Nach dem Übergang in den Nachkommateil enthalten die Zustände stattdessen den Abstand zur korrekten Lösung, welcher durch weitere Eingabesymbole noch zu überwinden ist. Da $w \in L(A)$ gilt und nur Zustände des Nachkommateils mit Wert 0 Akzeptanzzustände sind, ist der Abstand zur Lösung der Gleichung 0, weshalb die Dezimaldarstellung des Eingabewortes w die

Gleichung erfüllt und somit ein Teil der Lösungsmenge ist: $dec(w) \in \mathbb{L}_G$.

Vollständigkeit: $(x_1, \ldots, x_n) \in \mathbb{L}_G$ und $x_i \in \mathbb{F}\ \forall i \Rightarrow \text{bin}(\vec{x}) \in L(A)$
Entsprechend der Konstruktion folgen die Werte der Zustände des Automaten wieder dem Zustand der Gleichung.
Dabei wird zu keinem Zeitpunkt eine Abbruchbedingung des Automaten erfüllt, da diese unabhängig von dem Automaten auch für die Gleichung an sich gilt. Würde die Gleichung bei bitweisem Auswerten der Werte x_i einen Wert annehmen, welcher zu weit von der Lösung abweicht, sodass die Werte bei Hinzunehmen der folgenden Bits sich der Lösung nicht wieder nähern, so können die Werte x_i keine Lösung der Gleichung sein.
Dadurch enthält der Automat einen Pfad entlang des gesamten Wortes w. Da die Gleichung erfüllt ist, gilt $\langle \vec{x}, \vec{c} \rangle - k = 0$, der Abstand zur korrekten Lösung ist also 0. Da der Wert der Zustände des Nachkommateils bezüglich der gelesenen Bits dem Abstand zur Lösung entspricht, ist der Wert des Zustands nach Einlesen des gesamten Eingabewortes $w = bin(\vec{x})$ gleich 0. Es handelt sich damit um einen Akzeptanzzustand, woraus folgt: $bin(\vec{x}) \in L(A)$. □

3.3 Übertragen auf lineare Ungleichungen

Die Konstruktion von Automaten zu Ungleichungen ist sehr ähnlich. Ein Unterschied ist, dass kein exakter Wert erreicht werden muss sondern lediglich ein Wert größer bzw. kleiner als die Konstante k. Entsprechend werden die Abbruchbedingungen nur noch in die unerwünschte Richtung angewandt. Ist also ein Wert größer als k gesucht, so wird abgebrochen, wenn der Wert eines Zustands kleiner als k ist und durch die besten Transitionen auch keinen höheren Wert mehr annehmen kann. In die andere Richtung wird statt einer Abbruchbedingung eine ReLU-Erfüllungsgrenze überprüft. Wenn ein Wert diese Grenze überschreitet, so ist es nicht mehr möglich, dass folgende Werte die von der Ungleichung zu prüfende ReLU-Bedingung nicht erfüllen. Entsprechend können alle Werte, die diese Grenze überschreiten würden, auf den Grenzwert abgebildet werden. Dies ist nötig, damit der Automat nicht unendlich groß wird. Die Akzeptanzbedingung im Nachkommateil vereinfacht sich ebenfalls dahingehend, dass der Wert eines Zustands nicht exakt Null, sondern nur größer-gleich bzw. kleiner-gleich Null sein muss.

Korollar 1. *Sei G eine lineare Ungleichung der Form $\langle \vec{c}, \vec{x} \rangle \bowtie k$ mit $\bowtie\ \in \{\leq, \geq\}$. Dann existiert ein Automat A, sodass $L(A) = bin(\mathbb{L}_G|_{\mathbb{F}})$.*

3.4 Konstruktion eines Automaten aus einem linearen Programm

Der Automat zu einem gesamten Linearen Programm P ergibt sich aus dem Produkt der Automaten aller Gleichungen und Ungleichungen des Linearen Programmes. Da Transitionen nur dann in dem Produktautomaten A' existieren, wenn die es die entsprechende Transition in jedem der einzelnen Automaten A_i gibt, sind in A' nur Zustände erreichbar, deren Einzelzustände auch jeweils in A_i erreichbar sind. Umgekehrt bedeutet es, dass A' nur Zustände erreichbar sind, welche keine Abbruchbedingung eines A_i erfüllen. Weiterhin ist der Akzeptanzzustand des Produktautomaten der, welcher das Produkt aller Akzeptanzzustände der einzelnen Automaten ist und somit alle einzelnen Akzeptanzbedingungen erfüllt. Die akzeptierten Worte erfüllen somit alle Bedingungen des Linearen Programms: $L(A') = bin(\mathbb{L}_P|_{\mathbb{F}})$.

Theorem 1. *Seien $G_1, \ldots, G_k$ die lineare Gleichungen und Ungleichungen des Linearen Programms P. Seien $A_1, \ldots, A_k$ MTAs, sodass $\forall i : L(A_i) = bin(\mathbb{L}_{G_i})$. Sei A' das Produkt der Automaten. Dann gilt:*

$$L(A') = bin(\mathbb{L}_P|_{\mathbb{F}})$$

Beweis.

$$L(A') = L(A_1) \cap \ldots \cap L(A_k) = bin(\mathbb{L}_{G_1}|_{\mathbb{F}}) \cap \ldots \cap bin(\mathbb{L}_{G_k}|_{\mathbb{F}}) = bin(\mathbb{L}_P|_{\mathbb{F}})$$

□

Korollar 2. *Sei N ein Neuronales Netz und A ein Activation Pattern. Dann existiert ein MTA M, sodass gilt:*

$$R_{N_A} = R_M$$

4 Implementierung

In diesem Abschnitt wird die Implementierung der zuvor beschriebenen Konstruktion in OCaml behandelt.

4.1 Grundlegende Datentypen

Es folgt zunächst eine Erklärung der Datentypen der wichtigsten Konstrukte.

4.1.1 Neuronales Netz

Ein Neuronales Netz ist eine Liste von Schichten. Eine Schicht ist eine Liste von Knoten. Ein Knoten ist ein Tupel aus den Gewichten als Float-Liste, dem Bias als Float und dem Typ der Aktivierungsfunktion, also Id bzw. ReLU. Die Länge der Gewichtsliste der Knoten der ersten Schicht entspricht der Eingabedimension und die Länge der Knotenliste der letzten Schicht der Ausgabedimension des Netzes.
Im Folgenden werden Listen von Listen als „in NN-Form“ bezeichnet, wenn die Länge der äußeren Liste der Länge der Schichtenliste des Neuronalen Netzes entspricht und die Längen der inneren Listen jeweils den Längen der Knotenlisten der Schichten.

4.2 Activation Pattern

Das Activation Pattern ist eine Liste von Listen in NN-Form, welche für jeden Knoten des Neuronalen Netzes jeweils den Aktivierungszustand enthält.

4.2.1 MTA

Ein MTA besteht aus mehreren Feldern: Einem Integer für die Anzahl der Zustände, einem Integer mit der ID des Startzustands, einer Integer-Menge für die IDs aller Akzeptanzzustände, einem Integer für die Eingabedimension, eine Map von der Zustands-ID zu einer Transitionsmenge und einer Hashtable von der ID eines Zustands und eines Mehrspursymbols zu den IDs aller Nachfolgerzustände.

4.2.2 MTA-Toolbox

Eine MTA-Toolbox enthält die Informationen für die MTA Konstruktion und ist einer der Parameter für die Konstruktionsfunktionen. Sie enthält dabei den MTA selbst, eine State-Value-Map von Zustands-IDs zu den Zustandswerten, eine Value-State-Map von den Zustandswerten zu der Zustands-ID, einen Integer als Zähler für die Anzahl der Zustände, einen Integer für die Anzahl der Zustände, deren Transitionen bereits konstruiert wurden und ein sogenanntes „Toolbox-NN “, welches als Liste von Listen in NN-Form Informationen über die einzelnen Knoten des Neuronalen Netzes enthält, u.a. die Konstanten und Koeffizienten für die Gleichungen der Knoten wie bei den Linearen Programmen in Abschnitt 3.1 und die Aktivierungszustände.

4.3 Konstruktion

Zunächst wird aus dem gegebenen Neuronalen Netz und Activation Pattern das Toolbox-NN für die MTA-Toolbox erstellt. Der erste Schritt dafür ist die Berechnung der Koeffizienten und Konstanten.

4.3.1 Koeffizienten

Die Berechnung der Koeffizientenmatrix $C_i = W^{A,i}$ einer Schicht L_i ist nach Abschnitt 3.1:

$$W^{A,i} = \begin{cases} W_1^A & i = 1 \\ W_i^A W^{A,i-1} & sonst \end{cases}$$

Man beachte, dass das gegebene Activation Pattern bereits gemäß Abschnitt 2.3 auf die Gewichtsmatrizen W_i^A angewendet wurde.
Für diese Matrixmultiplikationen wird die OCaml Library „Owl“ [4] verwendet.
Weiterhin ist zu beachten, dass für die Größe der Koeffizientenmatrizen jeweils $C_i \in \mathbb{Q}^{n \times m_i}$ gilt, wobei n die Anzahl der Eingabevariablen des Neuronalen Netzes und m_i die Anzahl der Knoten der Schicht L_i.

Wie zu Beginn des Abschnitts 3.1 erwähnt benötigen die Gleichungen der Ausgabeschicht zusätzlich die zu prüfenden Ausgaben des Neuronalen Netzes als Eingaben, weshalb auch Koeffizienten für diese Variablen hinzugefügt werden müssen. Dies geschieht bei der Umwandlung der Matrix-Objekte der Owl-Library zu Listen von Listen von Koeffizientenlisten in NN-Form.
Sei n die Anzahl der Eingabevariablen und m die Anzahl der Ausgabeknoten des Neuronalen Netzes. Entsprechend hat jeder Knoten nach der Berechnung der Koeffizienten zu jeder der n Eingabevariablen einen Koeffizienten. Durch die Berücksichtigung der Ausgaben werden die Koeffizienten aller Knoten – und damit jede Koeffizientenliste – um m Koeffizienten erweitert.
Da jeder Knoten der Ausgabeschicht für genau eine Ausgabe steht, ist für jeden dieser Knoten nur die zugehörige Ausgabe relevant. Entsprechend wird für den i-ten Knoten der Ausgabeschicht der $(n+i)$-te Koeffizient auf minus Eins gesetzt und alle restlichen Ausgabekoeffizienten auf Null.
Die Ausgaben sind für Knoten, welche nicht zur Ausgabeschicht des Neuronalen Netzes gehören, nicht relevant, weshalb deren Koeffizienten für die Ausgabevariablen auf Null gesetzt sind und also m Nullen an die Reihen der Koeffizientenmatrizen angehängt werden.

4.3.2 Konstanten

Die Berechnung der Konstantenvektoren $\vec{k_i} = \vec{b}^{A,i}$ ist ähnlich zu der der Koeffizienten. Sie beginnt dabei mit den Biases der ersten Schicht, während die Gewichte der ersten Schicht irrelevant sind:

$$\vec{b}^{A,i} = \begin{cases} \vec{b}_1 & i = 1 \\ W_i^A \cdot \vec{b}^{A,i-1} + \vec{b}_i & sonst \end{cases}$$

Aus den Koeffizienten jedes Knotens wird zusätzlich das Tupel (c^+, c^-) der Summen der positiven bzw. negativen Koeffizienten gebildet. Weiterhin wird für jeden ReLU-Knoten die ReLU-Erfüllungsgrenze berechnet. Diese Informationen und die Aktivierungszustände der Knoten werden als Tupel in Listen von Listen in NN-Form als Toolbox-NN gespeichert.

4.3.3 MTA

Im Gegensatz zum theoretischen Ansatz werden bei der Implementierung nicht die einzelnen Automaten zu jeder Gleichung und Ungleichung des Linearen Programms erstellt und deren Produkt gebildet, sondern stattdessen direkt der Produktautomat gebaut. Entsprechend enthält ein Zustand nicht mehr den aktuellen Wert einer Gleichung bezüglich eines Knotens, sondern die Werte aller Gleichungen des Linearen Programmes in einer Liste von Listen von Werten in NN-Form. Nach einer Transition wird daher auch der Folgewert jedes der aktuellen Werte berechnet und bilden zusammen die Werte des Folgezustands.

In dem Pseudo-Code in Algorithmus 1 ist die Konstruktion des Vorkommateils des Automaten gegeben eines Toolbox-NNs gezeigt.

Die Konstruktion des MTA beginnt auch in der Implementierung mit dem Startzustand und der Konstruktion des Vorkommateils. Somit ist die Anzahl der Zustände des MTA $c = 1$ und die Anzahl der Zustände mit abgeschlossenen Transitionen $d = 0$.
In einer while-Schleife mit der Bedingung $d < q$ wird nun immer der Zustand mit ID d folgendermaßen abgearbeitet: Für jedes binäre Mehrspursymbol $\vec{b} \in \{0, 1\}^n$ mit n Spuren werden in Zeile 5 die Transitionen von dem aktuellen Zustand aus betrachtet und dazu die Werte der Folgezustände wie in Abschnitt 3.2 berechnet.
Für diese wird jeweils überprüft, ob ein Zustand mit diesen Werten bereits in der Value-State-Map existiert. Falls ja, wird eine Transition von dem aktuellen Zustand d mit dem entsprechenden Mehrspursymbol zu dem gefundenen Zustand zum MTA hinzugefügt.

Algorithm 1 Konstruktion Vorkommateil

Eingabe: Toolbox-NN T

Ausgabe: MTA-Toolbox $MT = \{mta; M_1, M_2, c = k; d = k; T\}$, wobei mta ein MTA mit allen Zuständen und Transitionen des Vorkommateils ist und k der Anzahl der Zustände entspricht.

```
1:  let MT ← {mta : new MTA; M1 : {}; M2 : {}; c : 1; d : 0; T : T}
2:  while d < c do
3:      let V_d = M1.get(d)
4:      for b⃗ ∈ {0,1}^n do
5:          V_{d+1} ← trans(V_d, b⃗, T)
6:          if V_{d+1} in M2.keys then
7:              mta.add_transition(d, b⃗, M2.get(V_{d+1}))
8:          else
9:              if not break(V_{d+1}, T) then
10:                 mta.add_transition(d, b⃗, c)
11:                 M1.add(V_{d+1}, c)
12:                 M2.add(c, V_{d+1})
13:                 c ← c + 1
14:     d ← d + 1
```

Gibt es noch keinen Zustand mit diesen Werten, so wird in Zeile 9 des Algorithmus überprüft, ob diese Werte eine Abbruchbedingung erfüllen. Ist eine Bedingung erfüllt, so wird dieser Zustand und die zugehörige Transition verworfen und mit dem nächsten Mehrspursymbol fortgefahren. Ist keine Abbruchbedingung erfüllt, so wird eine Transition von d zu c dem MTA und das Paar aus ID und Werten (c, V_{d+1}) den beiden Maps hinzugefügt. Außerdem wird der Zähler der Zustandsanzahl c inkrementiert, da dem MTA ein Zustand hinzugefügt wurde.
Sobald jedes binäre Mehrspursymbol für den aktuellen Zustand d bearbeitet wurde, wird d inkrementiert und damit der Zustand als bearbeitet markiert. Dies wiederholt sich, bis alle Zustände bearbeitet wurden, was möglich ist, da bei wachsender Zustandsmenge sich die Zustandswerte vermehrt wiederholen oder Abbruchbedingungen erfüllen und somit immer weniger neue Zustände entstehen.

Nach dem Vorkommateil wird von jedem bisher konstruierten Zustand – ausgenommen dem Startzustand – eine Dot-Transition $b = [., \ldots, .]$ durchgeführt. Dazu kann eine leicht abgewandelte Version des Pseudocodes in Algorithmus 1 verwendet werden.

Die MTA-Toolbox besteht dabei aus dem bisher erstellten MTA, neuen Maps, $c = c$, $d = 1$ und dem gleichen Toolbox-NN. Die Maps müssen jeweils geleert werden, damit in der weiteren Konstruktion keine Transitionen von Zuständen des Nachkommateils zu Zuständen des Vorkommateils entstehen. Die State-Value-Map des Vorkommateils M_V muss für die Dot-Transitionen aber noch außerhalb der MTA-Toolbox gespeichert werden. Selbiges gilt für die Anzahl der Zustände des Vorkommateils $c_V = c$.
Die Bedingung der while-Schleife für den Übergang zum Nachkommateil ist nun $d < c_V$. Es wird also nur über die Zustände des Vorkommateils iteriert. In Zeile 3 wird statt M_1 die State-Value-Map des Vorkommateils M_V verwendet und statt allen binären Mehrspursymbolen in Zeile 4 wird nur das Dot-Mehrspursymbol verwendet. Die Funktion trans in Zeile 5 entspricht der Übergangsgleichung der Dot-Transitionen aus Abschnitt 3.2 und die Funktion break überprüft die Abbruchbedingungen des Nachkommateils. Beim Hinzufügen eines neuen Zustands zum MTA muss zusätzlich noch überprüft werden, ob dieser die Akzeptanzbedingungen erfüllt und falls ja, wird er zu der Menge der Akzeptanzzustände des MTA hinzugefügt.

Als letztes folgt die Konstruktion des restlichen Nachkommateils. Auch hier kann der Pseudocode von Algorithmus 1 leicht abgewandelt verwendet werden.
Die MTA-Toolbox besteht aus dem bisher erstellten MTA. Die Maps der Übergangskonstruktion bleiben diesmal erhalten, da diese bereits Zustände des Nachkommateils enthalten. Weiterhin gilt $c = c$ und $d = c_V$. Da die Zustände fortlaufend nummeriert werden, entspricht die Anzahl der Zustände des Vorkommateils der ID des ersten Zustands des Nachkommateils und kann somit als Startpunkt der while-Schleife verwendet werden. Das Toolbox-NN bleibt unverändert.
Die Konstruktion verläuft wie die des Vorkommateils, nur dass bei den Funktionen trans und break die Übergangsgleichungen bzw. Abbruchbedingungen des Nachkommateils verwendet werden und dass wie bei den Dot-Transitionen auch die Akzeptanzbedingungen bei neuen Zuständen überprüft werden.

4.4 Tests

Die Korrektheit der Implementierung wird durch Tests mit OUnit sichergestellt.

4.4.1 Unit Tests

Für die meisten Funktionen der Implementierung wird unabhängig vom Gesamtprogramm einzeln getestet, ob sie die gewünschte Funktionalität erfüllen. Es wird dabei auch überprüft, ob falsche Eingaben die erwarteten Fehlermeldungen auslösen.

4.4.2 Automatenkonstruktion zu festem NN

Weiterhin wird die Konstruktion eines Automaten als Ganzes bezüglich einem festen Neuronalen Netz getestet. Dazu wird die Konstruktion zu dem Neuronalen Netz und einem ebenfalls festen Activation Pattern durchgeführt und auf dem entstandenen MTA überprüft, ob bestimmte Eingaben wie erwartet akzeptiert bzw. nicht akzeptiert werden.

4.4.3 Automatenkonstruktion auf zufälligen Eingaben

Darüber hinaus wird die Konstruktion mit zufälligen Eingaben getestet. Dabei kann die Struktur des Neuronalen Netzes beliebig festgelegt werden. Die Werte der Gewichte und Biases des Netzes sowie die Eingaben des Netzes, welche überprüft werden, sind dabei zufällig, wobei aber die Größe der Binärdarstellungen dieser Werte gewählt werden können. Die zu prüfenden Ausgaben werden aus dem Netz und den zufälligen Eingaben berechnet, um gültige Lösungen zu erhalten. Das Activation Pattern ergibt sich ebenfalls aus dieser Berechnung.
Aus dem zufälligen Neuronalen Netz und dem berechneten Activation Pattern wird der MTA gebaut. Es wird letztendlich überprüft, ob der MTA das Eingabewort aus den zufälligen Eingaben und den berechneten Ausgaben des Netzes akzeptiert.

5 Diskussion

In diesem Abschnitt wird abschließend betrachtet, welche Einschränkungen getroffen wurden, wie auf die Arbeit aufgebaut werden könnte und wie sie sich in den aktuellen Stand der Forschung einfügt.

Zunächst behandelt diese Arbeit nur die Konstruktion eines Automaten zu einem Neuronalen Netz beschränkt auf ein Activation Pattern. Ein Ansatz, um diese Beschränkung aufzuheben ist, zu allen möglichen Activation Pattern des Neuronalen Netzes einen Automaten – wie in dieser Arbeit gezeigt – zu konstruieren. Diese werden jeweils durch eine ε-Transition von

einen gemeinsamen Startzustand aus erreicht. Das Resultat ist ein nichtdeterministischer MTA, welcher IO-äquivalent zu einem uneingeschränkten Neuronalen Netz ist.

Eine weitere Einschränkung ist die Verwendung von binär endlich repräsentierbaren Zahlen. Dabei ist noch zwischen den Gewichten und Biases des Neuronalen Netzes und den Ein- und Ausgaben des Netzes und somit auch den Eingaben des Automaten zu unterscheiden.
Die Gewichte und Biases müssen eingeschränkt werden, da es in den Zuständen des Automaten sonst zu Rundungsfehlern kommt, welche zur Folge haben, dass unendlich viele Zustände mit minimal abweichenden Werten entstehen. Um dieses Problem zu beheben, müsste eine andere Datenstruktur verwendet werden.
Zahlen mit unendlicher Binärdarstellung können von den in dieser Arbeit behandelten MTAs nicht als Eingaben verwendet werden, da sie nur endliche Eingaben akzeptieren. Für unendliche Eingabeworte – und somit alle reellen Zahlen als mögliche Eingaben – muss die Konstruktion abgewandelt werden, um einen Büchi-Automaten zu erstellen.

Während der Ausarbeitung des in dieser Arbeit vorgestellten Ansatzes zur Konstruktion eines Automaten zu einem Neuronalen Netz ist das Buch „Automata Theory - An Algorithmic Approach“ von Javier Esparza und Michael Blondin [2] erschienen. Dieses stellt einen sehr ähnlichen Ansatz zur Konstruktion eines Automaten zu einer linearen Gleichung vor. Sie verwenden dabei Büchi-Automaten, sodass alle reellen Lösungen des Gleichungssystems von dem Automaten erkannt werden können, wodurch dieser Ansatz besser für die Konstruktion eines Automaten zu einem Neuronalen Netz wäre, als der in dieser Arbeit verwendeten Ansatz.

Literatur

[1] Dantzig, G. B. „Maximization of a linear function of variables subject to linear inequalities“ In: Koopman, T. C. (ed.): Activity Analysis of Production and Allocation, 339–347. (1951).

[2] Esparza, J. & Blondin, M. „Automata Theory - An Algorithmic Approach“ MIT Press (2023).

[3] Katz, G., Barrett, C., Dill, D., Julian, K. & Kochenderfer, M. „Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks“ arXiv:1702.01135v2. (2017). https://arxiv.org/abs/1702.01135

[4] Liang, W. & Jianxin, Z. „Owl - OCaml Scientific Computing“ (2018-2024). https://github.com/owlbarn/owl

[5] Rabin, M. & Scott, D. „Finite automata and their decision problems.“ IBM Journal of Research and Development, 3(2):114–125. (1959).

[6] Rosenblatt, F. „The perceptron: A probabilistic model for information storage and organization in the brain“ Psychological Review, 65(6), 386–408. (1958).

[7] Sälzer, M., Alsmann, E., Bruse, F. & Lange, M. „Verifying And Interpreting Neural Networks using Finite Automata.“ arXiv:2211.01022. (2022). https://arxiv.org/abs/2211.01022

[8] Sälzer, M. & Lange, M. „Reachability Is NP-Complete Even for the Simplest Neural Networks“ arXiv:2108.13179 (2021). https://arxiv.org/abs/2108.13179v2

[9] von Neumann, J. „First Draft of a Report on the EDVAC“ (1945).

[10] Wolper, P. & Boigelot, B. „On the Construction of Automata from Linear Arithmetic Constraints.“ In: Graf, S., Schwartzbach, M. (eds): Tools and Algorithms for the Construction and Analysis of Systems. TACAS 2000. Lecture Notes in Computer Science, vol 1785. Springer. (2000). https://doi.org/10.1007/3-540-46419-0_1