Gottfried Wilhelm Leibniz Universität Hannover
Institut für Theoretische Informatik

Komplexitätsuntersuchungen von neuronalen Netzwerken

Bachelorarbeit

Laura-Sophie Renz
Matrikelnummer: 10044095

Hannover, den 17. Juli 2024

Erstprüfer: Prof. Dr. rer. nat. Heribert Vollmer
Zweitprüfer: PD Dr. rer. nat. habil. Arne Meier
Betreuer: M. Sc. Vivian Holzapfel

Erklärung der Selbstständigkeit

Hiermit versichere ich, die vorliegende Bachelorarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet zu haben. Die Arbeit hat in gleicher oder ähnlicher Form noch keinem anderen Prüfungsamt vorgelegen.

Hannover, den 17. Juli 2024

_____

Laura-Sophie Renz

Inhaltsverzeichnis

1 Einleitung

Neuronale Netze gewannen in den letzten Jahren an großer Popularität. Ein Grund dafür sind die guten Ergebnisse, die sie in vielen praktischen Anwendungen liefern. In Hinblick auf deren vielseitigen und häufigen Einsatz ist eine Betrachtung von neuronalen Netzen von einem theoretischen Standpunkt aus von besonderem Interesse. Im Gegensatz zu den in der theoretischen Informatik lange etablierten Modellen begannen die Untersuchungen der Komplexität von neuronalen Netzen erst spät. Diese Arbeit soll zu diesen Komplexitätsuntersuchungen beitragen, indem die Anfänge solcher präsentiert werden. Dabei wird dieses neuartigere Modell mit zwei bekannten Modellen, der Turingmaschine und dem Booleschen Schaltkreis, verglichen. Dazu werden die Ergebnisse der zwei Paper „On the Computational Power of Neural Nets“ [SS95] und „Analog Computation via Neural Nets“ von Siegelmann und Sontag [SS94] vorgestellt.

In Kapitel 2 werden die Grundlagen für die in den folgenden Kapiteln behandelten Themen vorgestellt. Die beiden Kapitel 3 und 4 bilden den Kern dieser Arbeit, in denen neuronale Netze mit anderen Modellen der theoretischen Informatik verglichen werden. Dabei wird in Kapitel 3 eine Methode vorgestellt wie aus einer Turingmaschine ein neuronales Netz konstruiert werden kann. Kapitel 4 führt die zwei Klassen NET und CIRCUIT ein, für die eine Beziehung zueinander gezeigt wird, aus der folgt, dass neuronale Netze und Boolesche Schaltkreise mit einem polynomiellen Overhead ineinander überführt werden können. In beiden Kapiteln werden die beschriebenen Ergebnisse jeweils an einem Beispiel verdeutlicht. Abschließend bietet Kapitel 5 einen Überblick über die behandelten Inhalte sowie einen Ausblick auf weitere mögliche Forschung.

2 Mathematische Grundlagen

Das folgende Kapitel dient der Erklärung und Einführung einiger Begriffe. Diese bilden eine Grundlage, die für das Verständnis der in Kapitel 3 und 4 beschriebenen Konzepte und Themen notwendig ist.

2.1 Cantormenge

Da im Bereich der Informatik Algorithmen auf Modellen ausgeführt werden, die in einem diskreten Raum arbeiten, können keine kontinuierlichen Zahlenspektren wie die reellen Zahlen vollständig abgebildet werden. Das führt dazu, dass es schwer wird, zwei Zahlen aus so einer Menge voneinander zu unterscheiden, wenn diese einen beliebig kleinen Abstand haben. Um dieses Problem zu umgehen, ist es wünschenswert, Zahlen aus einer Grundmenge zu wählen, welche die Eigenschaft aufweist, dass zwei beliebige Elemente dieser Menge immer leicht voneinander unterscheidbar sind.

Dafür wird in dieser Arbeit die sogenannte Cantormenge verwendet, die eine Menge mit Lücken darstellt, durch welche genau diese gewünschte Eigenschaft entsteht.

Definition 1. *[App21] Die Cantormenge C ist der Schnitt der Folge von Mengen C_n*

$$C := \bigcap_{i=0}^{\infty} C_n$$

wobei die Folge C_n wie folgt aufgebaut ist:

$$\begin{aligned} C_0 &:= [0,1]\,, \\ C_1 &:= \left[0, \frac{1}{3}\right] \cup \left[\frac{2}{3}, 1\right], \\ C_2 &:= \left[0, \frac{1}{9}\right] \cup \left[\frac{2}{9}, \frac{1}{3}\right] \cup \left[\frac{2}{3}, \frac{7}{9}\right] \cup \left[\frac{8}{9}, 1\right] \textit{ usw.} \end{aligned}$$

In der Konstruktion der weiteren C_n wird von jedem Teilintervall jeweils das mittlere Drittel entfernt, was dafür sorgt, dass in den enthaltenen Intervallen solange Lücken auftreten bis schlussendlich jede Zahl einen Abstand zu ihren Nachbarn hat. Obwohl dieses Vorgehen bis ins Unendliche wiederholt wird, ist die Cantormenge nicht leer, sondern enthält alle Zahlen im Intervall $[0, 1]$, die in einer der Mengen C_n als Intervallgrenze auftreten.

Die so entstehenden Zahlen können in Basis 3 dargestellt werden, indem jede Ziffer $a_i \in \{0, 2\}$ dem Drittel entspricht, in das diese Zahl fällt. Diese a_i werden als Nachkommastellen interpretiert, wodurch eine Zahl der Form $0, a_1 a_2 a_3 \ldots_3 \in [0, 1]$ gebildet wird. Da aufgrund der Konstruktion keine Zahl der Cantormenge in das mittlere Drittel fallen kann, sind alle $a_i \neq 1$.

Bei der Betrachtung zweier Cantorzahlen $0.0\ldots_3$ und $0.2\ldots_3$ können diese selbst für unbekannte Nachkommastellen leicht unterschieden werden können, da die möglichen Intervalle einen Abstand haben. Diese Eigenschaft gilt für reelle Zahlen nicht, da zum Beispiel $0.0\ldots$ und $0.1\ldots$ je nach folgenden Stellen einen sehr ähnlichen oder im Fall von $0.0\overline{9} = 0.1$ sogar den gleichen Wert annehmen können.

Analog zum hier beschriebenen Verfahren, welches zu Zahlen der Basis drei führt, können bei einer anderen Wahl der Anzahl von Teilintervallen in der Menge C_1 andere Mengen entstehen, die jedoch die gleiche Eigenschaft der Lücken aufweisen. Diese cantorartigen Mengen werden dann als Zahlen in einer andren Basis interpretiert, die der Anzahl an Teilintervallen entspricht. Im Folgenden werden solche Mengen auch als Cantormenge bezeichnet.

2.2 Aktivierungsfunktion

In neuronalen Netzen wird häufig eine Aktivierungsfunktion auf die neu berechneten Werte der Neuronen angewendet. Die Nutzung einer Aktivierungsfunktion bietet dabei verschiedene Vorteile. Zum einen kann durch diese ein nicht linearer Aspekt in die Berechnung eingebracht werden, wodurch die Berechnung vielfältiger Funktionen möglich ist, obwohl die Werte der Neuronen durch leichte Berechnungen wie einer Linearkombination bestimmt werden. Zum anderen kann der mögliche Zahlenbereich auf ein gewünschtes Intervall eingeschränkt werden, was eine Interpretation der Aktivierungen unter anderem als Wahrscheinlichkeiten ermöglicht. Des Weiteren sind Aktivierungsfunktionen meist differenzierbar, um beim Trainieren der Gewichte einen Gradientenabstieg durchführen zu können.

In dieser Arbeit wird die sigmoidartige Funktion $\sigma\colon \mathbb{R} \to [0, 1]$ mit n. def.

$$\sigma(x) = \begin{cases} 0, & x < 0, \\ x, & 0 \leq x \leq 1, \\ 1, & x > 1 \end{cases}$$

als Aktivierungsfunktion verwendet. Sie erfüllt die Eigenschaften der Nichtlinearität sowie der Beschränkung des Zahlenbereichs und ist zusätzlich leicht berechenbar. Die fehlende Differenzierbarkeit führt in dieser Arbeit nicht zu Problemen, da die Gewichte nicht angepasst werden müssen. Es ist daher von Vorteil eine möglichst einfache Funktion σ zu wählen, um das Verständnis der Berechnungen zu erleichtern.

2.3 Dynamisches System

In der Mathematik ist ein dynamisches System ein Modell, mit dem die Veränderung eines Systems über die Zeit beschrieben werden kann. Da ein System über seinen Zustand definiert ist, wird die zeitliche Veränderung durch den Wechsel des Zustandes dargestellt.

Definition 2. *[Den04] Ein dynamisches System ist ein Tupel* (Ω, G) *mit*

- *einem Zeitraum G, einer Halbgruppe mit 1, und*
- *einer Menge* $\Omega \neq \emptyset$ *von Zuständen,*

sodass eine Funktion $\Phi\colon (\omega, g) \in \Omega \times G \to g\omega \in \Omega$ *existiert, welche die folgenden Eigenschaften erfüllt:*

1. $(g\omega) \mapsto h(g\omega) = (hg)\omega$ *und*
2. $e\omega = \omega$ *mit der Einheit e von G.*

In dieser Arbeit werden dynamische Systeme betrachtet, bei denen die Zeitschritte $G = \mathbb{N}$ sind. Das bedeutet, dass die zweite Eigenschaft ausdrückt, dass der Zustand nicht gewechselt wird, wenn kein Zeitschritt vergeht. Die erste Eigenschaft drückt aus, dass man im gleichen Zustand landet, wenn man ausgehend vom gleichen Zustand

3 Turingmaschinen und neuronale Netze

In diesem Abschnitt wird das Modell des Prozessornetzwerkes, eines speziellen neuronalen Netzes, vorgestellt und mit der Turingmaschine verglichen. Dabei wird auf eine Methode eingegangen, wie aus einer beliebigen Turingmaschine so ein Prozessornetzwerk konstruiert werden kann, sodass beide die gleiche Funktion berechnen. Dazu wird zunächst das Prozessornetzwerk erklärt und formal definiert. Anschließend werden einige Definitionen zur in dieser Arbeit genutzten Turingmaschine gegeben und die Konstruktion des Netzwerks formal sowie anhand eines Beispiels erläutert.

Alle Ergebnisse aus diesem Kapitel beruhen auf Inhalten des Papers „On the Computational Power of Neural Nets“ von Siegelmann und Sontag [SS95].

3.1 Prozessornetzwerk mit rationalen Gewichten

Wie bereits genannt ist ein Prozessornetzwerk ein neuronales Netz, für welches jedoch einige weitere Eigenschaften gelten. Ein Prozessornetzwerk besteht aus einer endlichen Anzahl an Neuronen, auch Prozessoren bezeichnet, die synchron arbeiten. Die Anzahl der Prozessoren in einem Netzwerk wird mit $N \in \mathbb{N}$ bezeichnet, wohingegen $M \in \mathbb{N}$ die Anzahl der Eingabedatenleitungen beschreibt. Die Struktur dieses Netzes kann als Graph dargestellt werden, in dem auch Kreise auftreten dürfen und werden.

Die Aktivierungsfunktion eines jeden Prozessors ist eine Linearkombination der Aktivierungswerte $x_j(t)$ aller Prozessoren zum Zeitpunkt t sowie der Eingabeparameter $u_k(t)$, auf die eine Sigmoidfunktion angewendet wird. Allgemein kann der Wert jedes Prozessors im nachfolgenden Zustand daher wie folgt berechnet werden:

$$x_i(t+1) = \sigma\left(\sum_{j=1}^{N} a_{ij}x_j(t) + \sum_{k=1}^{M} b_{ik}u_k(t) + c_i\right), \forall\, i = 1, \ldots, N.$$

Die Gewichte $a_{ij}, b_{ik}, c_i \in \mathbb{Q}$ beschreiben den Einfluss eines Prozessors auf die Aktivierung eines anderen. Diese einfache Art von Aktivierungsfunktion ist ausreichend, um gewährleisten zu können, dass jede beliebige Turingmaschine in ein Netzwerk übertragen werden kann. Es sind also keine Funktionen höherer Ordnung notwendig, die Multiplikationen von

Prozessorktivierungen erlauben.

Der Zustand des Netzwerks zu einem Zeitpunkt $t \in \mathbb{N}$ wird durch den Vektor $x(t) \in \mathbb{Q}^N$ beschrieben. Jeder Eintrag in x entspricht der Aktivierung eines bestimmten Prozessors.

Ein- und Ausgabewörter ω des Prozessornetzwerks sind Streams. Im Folgenden wird mit jeweils zwei Ein- und Ausgabeleitungen gearbeitet. Dabei haben die zwei Leitungen unterschiedliche Aufgaben. Eine ist für die Übertragung des Ein- bzw. Ausgabesignals verarntwortlich, die andere für die Validierung dieses Signals.

Eingaben sind daher Tupel der Form $u(t) = (V(t), D(t))$ mit der Datenleitung $D(t)$ und der Validierungsleitung $V(t)$. Die beiden Funktionen sind für die Eingabelänge $k \in \mathbb{N}$ wie folgt definiert:

$$V_\omega(t) = \begin{cases} 1, & t \leq k, \\ 0 & \text{sonst}, \end{cases}$$

$$D_\omega(t) = \begin{cases} \omega_t, & t \leq k, \\ 0 & \text{sonst}. \end{cases}$$

Analog sind Ausgaben Tupel der Form $y(t) = (G(t), H(t))$ mit der Datenleitung $H(t)$ und der Validierungsleitung $G(t)$. Die Ausgabefunktionen sind für einen Zeitpunkt $r \in \mathbb{N}$ und Ausgabelänge $l \in \mathbb{N}$ wie folgt definiert:

$$G_{\omega,r}(t) = \begin{cases} 1, & r \leq t \leq r + l - 1, \\ 0 & \text{sonst}, \end{cases}$$

$$H_{\omega,r}(t) = \begin{cases} \beta_{t-r+1}, & r \leq t \leq r + l - 1, \\ 0 & \text{sonst}. \end{cases} \tag{3.1}$$

Definition 3. *Ein Prozessornetzwerk ist ein dynamisches System mit zwei binären Inputs und der Übergangsfunktion* $\mathcal{F} \colon \mathbb{Q}^N \times \{0,1\}^2 \to \mathbb{Q}^N$*, wobei*

$$\mathcal{F}(x, u) = \sigma(Ax + b_1 u_1 + b_2 u_2 + c)$$

ist mit $A \in \mathbb{Q}^{N \times N}$*,* $b_1, b_2, c \in \mathbb{Q}^N$ *und* $\sigma \colon \mathbb{Q}^N \to \mathbb{Q}^N$*.*

In der Matrix A werden die Gewichte festgehalten, die den gegenseitigen Einfluss der Prozessoren beschreiben. Dabei gibt es jeweils eine Zelle für alle möglichen Kombinationen von zwei Prozessoren. Die Aktivierung aller Prozessoren ist im Vektor x beschrieben. Die Vektoren b_1 und b_2 sind die Gewichte der beiden Eingabeleitungen u_1 und u_2. Der *bias* c ist ein Vektor, der die Aktivierung eines Prozessors um einen konstanten Wert verschiebt. Die

Funktion σ symbolisiert die komponentenweise Anwendung der Sigmoidfunktion auf den Lösungsvektor.

Ein Prozessornetzwerk berechnet eine Funktion $\Phi\colon A \subseteq \{0,1\}^+ \to \{0,1\}^+$, falls

(i) für jedes Wort ω, auf dem Φ definiert ist, ein Zeitpunkt $r \in \mathbb{N}$ existiert, sodass Formel 3.1 gilt.

(ii) für jedes Wort ω, auf dem Φ nicht definiert ist, die Validierungsleitung $G(t) = 0$ ist für alle $t \in \mathbb{N}$.

Da die Konstruktion eines Prozessornetzwerkes ohne Ein- und Ausgaben leichter verständlich ist, wird hier eine Methode zur Codierung dieser vorgestellt und das Verfahren anschließend für das Netzwerk ohne Ein- und Ausgaben beschrieben. Das bedeutet, dass Ein- und Ausgaben nicht mehr als Streams auftreten, sondern die Eingaben im initialen Zustand codiert sind und die Ausgaben im Ausgabeprozessor vorliegen, sobald die Berechnung abgeschlossen ist.

Dieses Vorgehen ist keine Einschränkung, da die Ein- und Ausgaben ohne Änderung des Verfahrens wieder hinzugefügt werden können. Beide Varianten sind demnach äquivalent.

Es folgt die Beschreibung der Codierung. Für ein Wort $\omega = a_1 \dots a_k \in \{0,1\}^+$ ist die Codierung des Wortes

$$\delta[a_1 \dots a_k] := \sum_{i=1}^{k} \frac{2a_i + 1}{4^i}. \tag{3.2}$$

Alle Zahlen, die als Codierung auftreten können, bilden eine Cantormenge mit der Basis vier, wodurch sie leicht voneinander unterscheidbar sind. Des Weiteren kommen als δ-Werte nur Zahlen bestehend aus den Ziffern 1 und 3 vor.

3.2 Turingmaschinen auf Stacks

Die in dieser Arbeit verwendeten Turingmaschinen arbeiten auf Stacks und sind äquivalent zu Turingmaschinen, welche auf Bändern arbeiten. Diese Darstellung des Modells ist jedoch vorteilhafter, um die Funktionen auf einzelne Neuronen zu übertragen und aufzuteilen.

Um die Ein- und Ausgaben sowie die Inhalte der Stacks so zu beschreiben, dass später eine Überführung ins Prozessornetzwerk gelingt, wird die zuvor in Formel 3.2 eingeführte Codierung mit der δ-Funkion verwendet.

Diese Art der Codierung sorgt dafür, dass mit Hilfe von Funktionen, welche die δ-Werte nutzen, Eigenschaften der Stacks geprüft werden können. Zwei solche Eigenschaften, die Bestimmung des obersten Stackelements sowie die Prüfung der Leere eines Stacks, werden im Folgenden ausgeführt. Um die Übersichtlichkeit der Formeln im weiteren Verlauf des Textes zu wahren, wird der codierte Stack mit $q = \delta[\omega]$ bezeichnet.

Das oberste Element eines Stacks kann durch $\zeta(q) \coloneqq \sigma(4q - 2)$ bestimmt werden. Diese Formel folgt aus der Lücke der gewählten Cantormenge im Intervall $\left[\frac{2}{4}, \frac{3}{4}\right)$. Alle Codierungen von Stacks mit oberstem Element eins haben einen δ-Wert größer oder gleich $\frac{3}{4}$, da in der Summe aus Formel 3.2 nur positive Summanden auftreten und der erste Summand

$$\frac{2 \cdot 1 + 1}{4^1} = \frac{3}{4}$$

diese oberste Eins codiert. Bei einer Null als oberstes Stackelement hingegen ist der erste Summand

$$\frac{2 \cdot 0 + 1}{4^1} = \frac{1}{4}.$$

Alle weiteren Summanden haben eine Wertigkeit, die insgesamt $\frac{1}{4}$ nicht überschreitet, da sie als Zahlen der Basis vier interpretiert werden. Dadurch ergibt sich eine obere Schranke von $\frac{2}{4}$.

Zum Überprüfen, ob ein Stack leer ist, kann die Funktion $\tau(q) \coloneqq \sigma(4q)$ verwendet werden. Da wie zuvor beschrieben alle Summanden positiv sind, ist der Stack mit nur einem Element null, derjenige mit der kleinstmöglichen Codierung von $\frac{1}{4}$. Daraus folgt, dass alle nichtleeren Stacks δ-Werte im Intervall $[\frac{1}{4}, 1]$ haben und daher auf eins abgebildet werden. Nur leere Stacks haben die Codierung $q = 0$ und werden auf null abgebildet.

Nun lassen sich die q-Werte der Stacks auch dafür nutzen die möglichen Stackoperationen (*push 0*, *push 1*, *pop*) als Funktion auszudrücken. Dabei repräsentiert das q^+ in den Formeln 3.3, 3.4, 3.5 den einen Zeitschritt später vorliegenden Wert ausgehend von q. Für die beiden *push*-Operationen wird zunächst der ganze Stackinhalt um eine Stelle nach rechts verschoben und dann das neue oberste Element addiert. Der Rechtsshift wird in Basis vier durch $\frac{q}{4}$ ausgedrückt. Daraus ergibt sich für die Operation *push* 0 die Berechnung

$$q^+ = \sigma\left(\frac{q}{4} + \frac{1}{4}\right) \tag{3.3}$$

und analog für die Operation *push* 1

$$q^+ = \sigma\left(\frac{q}{4} + \frac{3}{4}\right). \tag{3.4}$$

Die Operation *pop* hingegen wird mit einem Linksshift und anschließender Subtraktion von eins oder drei realisiert. Die Auswahl der richtigen Zahl wird mit Hilfe der ζ-Funktion getroffen. Es ergibt sich die Berechnung

$$q^+ = \sigma(4q - (2\zeta(q) + 1))). \tag{3.5}$$

Damit der Bezug im Beweis eindeutig ist, wird hier eine konkrete Definition der genutzen Turingmaschine gegeben.

Definition 4. *Eine Turingmaschine $\mathcal{M}$ mit p Stacks wird durch ein $(p+4)$-Tupel*

$$(S, s_I, s_H, \theta_0, \theta_1, \theta_2, \ldots, \theta_p)$$

beschrieben. Dabei ist S die Menge aller Zustände, s_I, der initiale Zustand, s_H der Haltezustand und θ_i die Übergangsfunktionen der Stacks und des Zustandes.

Die Übergangsfunktion

$$\theta_0 \colon S \times \{0,1\}^{2p} \to S \tag{3.6}$$

ist für die Berechnung des neuen Zustandes verantwortlich. Die Menge $\{0,1\}^{2p}$ repräsentiert dabei die ζ-Werte sowie τ-Werte aller Stacks.

Die weiteren Funktionen θ_1 bis θ_p bestimmen welche Operation auf dem jeweiligen Stack im aktuellen Zustand ausgeführt werden soll. Für $i \in \{1, \ldots, p\}$ gilt

$$\theta_i \colon S \times \{0,1\}^{2p} \to \left\{(1,0,0), \left(\frac{1}{4}, 0, \frac{1}{4}\right), \left(\frac{1}{4}, 0, \frac{3}{4}\right), (4,-2,-1)\right\}. \tag{3.7}$$

Die Tripel des Wertebereichs entsprechen dabei jeweils den in den Formeln 3.3, 3.4, 3.5 eingeführten Berechnungen sowie keiner Veränderung auf dem Stack. Ein Tripel (x, y, z) repräsentiert eine Veränderung des Stack in Form von

$$q^+ = x\,q + y\,\zeta(q) + z.$$

Zu jedem Zeitpunkt kann die Turingmaschine durch die Konfiguration

$$(s, q_1, q_2, \ldots, q_p)$$

genau beschrieben werden, welche aus dem Zustand und den q-Werten aller Stacks besteht. Die Menge aller Konfigurationen wird mit $\mathcal{X}$ bezeichnet. Die Funktion $\mathcal{P} \colon \mathcal{X} \to \mathcal{X}$ ist dann die Übergangsfunktion der Turingmaschine, welche für einen Zeitschritt sowohl die Zustandsänderung, als auch die Operationen auf allen Stacks beschreibt. Sie setzt sich aus den θ-Funktionen in Definition 4 zusammen.

3.3 Konstruktion des Prozessornetzwerks

Nach der Definition der beiden Modelle Prozessornetzwerk und Turingmaschine, wird in diesem Abschnitt der Zusammenhang dieser Modelle verdeutlicht.

Satz 5. *Sei $\mathcal{M}$ eine Turingmaschine mit p Stacks, welche in $T(|\omega|)$ Schritten eine Funktion Φ: $\{0,1\}^+ \to \{0,1\}^+$ berechnet. Dann existiert ein Prozessornetzwerk ohne Eingaben, welches*

ausgehend vom initialen Zustand

$$x(\omega)^0 := (\delta[\omega], 0, \ldots, 0) \in \mathbb{Q}^N$$

ein Ergebnis äquivalent zu $\mathcal{M}$ berechnet. Das bedeutet, dass die zweite Koordinate $x(\omega)_2^j$ zu jedem Zeitpunkt j gleich null ist, falls $\Phi(\omega)$ nicht definiert ist und andernfalls ein $\mathcal{T} \in O(T)$ exisiert, sodass

$$x(\omega)_2^j = 0, \quad j \in \{0, \ldots, \mathcal{T} - 1\},$$

$$x(\omega)_2^{\mathcal{T}} = 1$$

und

$$x(\omega)_1^{\mathcal{T}} = \delta[\Phi(\omega)].$$

In diesem Abschnitt folgt nun der Beweis von Satz 5. Hierfür werden die im vorherigen Abschnitt beschriebenen Definitionen verwendet, um alle benötigten Operationen der Stacks als Funktionen über Neuronen zu beschreiben.

Beweis. Zunächst wird eine Funktion $\widetilde{\mathcal{P}}$ benötigt, welche die gleichen Lösungen wie die Übergangsfunktion $\mathcal{P}$ herbeiführt, aber eine Struktur hat, die vom Prozessornetzwerk verarbeitet und im darauf folgenden Schritt ausgeführt werden kann.
$\widetilde{\mathcal{P}}\colon \mathbb{Q}^{s+p} \to \mathbb{Q}^{s+p}$ wird definiert als

$$(x_1, \ldots, x_s, q_1, \ldots, q_p) \mapsto (x_1^+, \ldots, x_s^+, q_1^+, \ldots, q_p^+).$$

Die x-Werte repräsentieren alle möglichen Zustände, was dazu führt, dass immer nur die den aktuellen Zustand repräsentierende Variable den Wert eins annimmt.

Zusätzlich werden zwei Funktionen benötigt, welche das Äquivalent zu den Formeln 3.6 und 3.7 im Prozessornetzwerk darstellen. Für den Zustand ist dies die Funktion $\beta_{ij}\colon \{0,1\}^{2p} \to \{0,1\}$ mit

$$\begin{aligned} &\beta_{ij}(\zeta(q_1), \ldots, \zeta(q_p), \tau(q_1), \ldots, \tau(q_p)) = 1 \\ \Leftrightarrow\ &\theta_0(j, \zeta(q_1), \ldots, \zeta(q_p), \tau(q_1), \ldots, \tau(q_p)) = i. \end{aligned}$$

β_{ij} ist genau dann eins, wenn der aktuelle Zustand j ist und in den Zustand i gewechselt wird. Die Änderungen der Stacks werden mit der Funktion $\gamma_{ij}^k\colon \{0,1\}^{2p} \to \{0,1\}$ dargestellt, wobei

$$\begin{aligned} &y_{ij}^k(\zeta(q_1), \ldots, \zeta(q_p), \tau(q_1), \ldots, \tau(q_p)) = 1 \\ \Leftrightarrow\ &\theta_i(j, \zeta(q_1), \ldots, \zeta(q_p), \tau(q_1), \ldots, \tau(q_p)) = c \end{aligned}$$

gilt. γ_{ij}^k ist genau dann eins, wenn der aktuelle Zustand j ist und auf Stack i die Operation ausgeführt wird, die $k \in \{1, 2, 3, 4\}$ entspricht. Die Zahl k wird dabei jeweils einem Tripel aus

3.7 zugeordnet. Die Funktionen γ_{ij}^k und β_{ij} können nun genutzt werden, um die Berechnung der einzelnen Elemente aus $\widetilde{\mathcal{P}}$ aufzustellen. Der neue Zustand kann daher mit

$$x_i^+ \coloneqq \sum_{j=0}^{s} \beta_{ij}(\zeta(q_1), \ldots, \zeta(q_p), \tau(q_1), \ldots, \tau(q_p))x_j \tag{3.8}$$

berechnet werden, der Haltezustand mit

$$x_0 \coloneqq 1 - \sum_{j=1}^{s} x_j \tag{3.9}$$

und der neue δ-Wert des i-ten Stacks mit

$$\begin{aligned} q_i^+ \coloneqq & \left(\sum_{j=0}^{s} \gamma_{ij}^1(\zeta(q_1), \ldots, \zeta(q_p), \tau(q_1), \ldots, \tau(q_p))x_j \right) q_i \\ & + \left(\sum_{j=0}^{s} \gamma_{ij}^2(\zeta(q_1), \ldots, \zeta(q_p), \tau(q_1), \ldots, \tau(q_p))x_j \right) \cdot \left(\frac{1}{4} q_i + \frac{1}{4} \right) \\ & + \left(\sum_{j=0}^{s} \gamma_{ij}^3(\zeta(q_1), \ldots, \zeta(q_p), \tau(q_1), \ldots, \tau(q_p))x_j \right) \cdot \left(\frac{1}{4} q_i + \frac{3}{4} \right) \\ & + \left(\sum_{j=0}^{s} \gamma_{ij}^4(\zeta(q_1), \ldots, \zeta(q_p), \tau(q_1), \ldots, \tau(q_p))x_j \right) \cdot (4q_i - 2\zeta[q_i] - 1) \,. \end{aligned} \tag{3.10}$$

Da die Bestandteile der $\widetilde{\mathcal{P}}$-Funktion genau so definiert sind, dass sie die gleichen Übergänge beschreiben, wie die θ-Funktionen aus denen $\mathcal{P}$ aufgebaut ist, ergibt sich die geforderte Äquivalenz der beiden Funktionen. Die unterschiedliche Codierung der Zustände ändert diesen Umstand nicht. Zu jedem Zeitpunkt während, vor und nach der Berechnung kann ein Tupel der einen Form in eines der anderen überführt werden, ohne, dass das Ergebnis davon betroffen ist.

Nun können die Berechnungen in 3.8 und 3.10 in dieser Form noch nicht vom Prozessornetzwerk simuliert werden, da diese nicht die in Definition 3 geforderte Struktur einer Linearkombination mit anschließender Sigmoidfunktion aufweisen. Vor der konkreten Beschreibung der Netzwerkstruktur ist es daher noch notwendig die Formeln für x^+ und q^+ in die gewünschte Form zu bringen, indem die Funktionen β und γ durch Linearkombinationen ersetzt werden. Zur Umformung von x wird die für beliebige Funktionen $\beta\colon \{0,1\}^t \to \{0,1\}$ geltende Regel

$$\beta(d_1, \ldots, d_t)x = \sum_{r=1}^{2^t} c_r \sigma(\langle v_r, (1, d_1, \ldots, d_t, x) \rangle)$$

verwendet. Dabei sind $c_r \in \mathbb{Z}$ neue Skalare und $v_r \in \mathbb{Z}^{t+2}$ neue Vektoren. Dies kann genutzt werden, um die Berechnung des neuen Zustandes in die gewünschte Form zu bringen. Dabei

gilt:

$$\begin{aligned} x_i^+ &= \sum_{j=0}^{s} \beta_{ij}(\zeta(q_1), \ldots, \zeta(q_p), \tau(q_1), \ldots, \tau(q_p)) x_j \\ &= \sum_{j=0}^{s} \sum_{r=1}^{2^{2p}} c_r \sigma\left(\langle v_r, (1, \zeta(q_1), \ldots, \zeta(q_p), \tau(q_1), \ldots, \tau(q_p), x_j)\rangle\right) \\ &= \sum_{j=0}^{s} \sum_{r=1}^{2^{2p}} c_r \sigma\left(\langle v_r, (1, \sigma(4q_1 - 2), \ldots, \sigma(4q_p - 2), \sigma(4q_1), \ldots, \sigma(4q_p), x_j)\rangle\right) \end{aligned} \tag{3.11}$$

Für die Berechnung der Stackinhalte kann eine ähnliche für beliebige $\beta\colon \{0,1\}^t \to \{0,1\}$ geltende Regel genutzt werden:

$$\beta(d_1, \ldots, d_t) x \cdot q = \sigma\left(q + \sum_{r=1}^{2^t} c_r \sigma(\langle v_r, (1, d_1, \ldots, d_t, x)\rangle) - 1\right).$$

Mit dieser erhält man die neue Berechnung

$$\begin{aligned} q_i^+ &= \sigma\left(q_i + \sum_{j=0}^{s} \sum_{r=1}^{2^{2p}} c_r \sigma(\langle v_r, (1, \zeta(q_1), \ldots, \zeta(q_p), \tau(q_1), \ldots, \tau(q_p), x_j)\rangle) - 1\right) \\ &+ \sigma\left(\left(\frac{1}{4} q_i + \frac{1}{4}\right) + \sum_{j=0}^{s} \sum_{r=1}^{2^{2p}} c_r \sigma(\langle v_r, (1, \zeta(q_1), \ldots, \zeta(q_p), \tau(q_1), \ldots, \tau(q_p), x_j)\rangle) - 1\right) \\ &+ \sigma\left(\left(\frac{1}{4} q_i + \frac{3}{4}\right) + \sum_{j=0}^{s} \sum_{r=1}^{2^{2p}} c_r \sigma(\langle v_r, (1, \zeta(q_1), \ldots, \zeta(q_p), \tau(q_1), \ldots, \tau(q_p), x_j)\rangle) - 1\right) \\ &+ \sigma\left((4q_i - 2\zeta[q_i] - 1) + \sum_{j=0}^{s} \sum_{r=1}^{2^{2p}} c_r \sigma(\langle v_r, (1, \zeta(q_1), \ldots, \zeta(q_p), \tau(q_1), \ldots, \tau(q_p), x_j)\rangle) - 1\right). \end{aligned} \tag{3.12}$$

Aufgrund der Länge der Formel wurden hier die Umformungsschritte sowie die Darstellung mit den expliziten Formen der ζ- und τ-Funktionen weggelassen. Diese sind jedoch analog zu 3.11.

Die Berechnungen in Formel 3.11 und 3.12 können von einem Prozessornetzwerk mit vier Ebenen ausgeführt werden. Das führt dazu, dass jeder Schritt der Turingmaschine vier Schritten des Prozessornetzwerks entspricht, wobei eine Ebene von Prozessoren immer einen der vier Teilschritte ausführt. In Abbildung 3.13 wird der Aufbau des Prozessonetzwerks vereinfacht dargestellt. Die vier Prozessoren für die Zähler sind dabei nicht aufgeführt und die Übergänge von Ebene F_1 nach Ebene F_4 nur angedeutet.

Die in Abbildung 3.13 grau eingefärbten Knoten stellen dar, dass der entsprechende Prozessor nur der Erhaltung der Werte aus der darüberliegenden Ebene dient und keine neuen

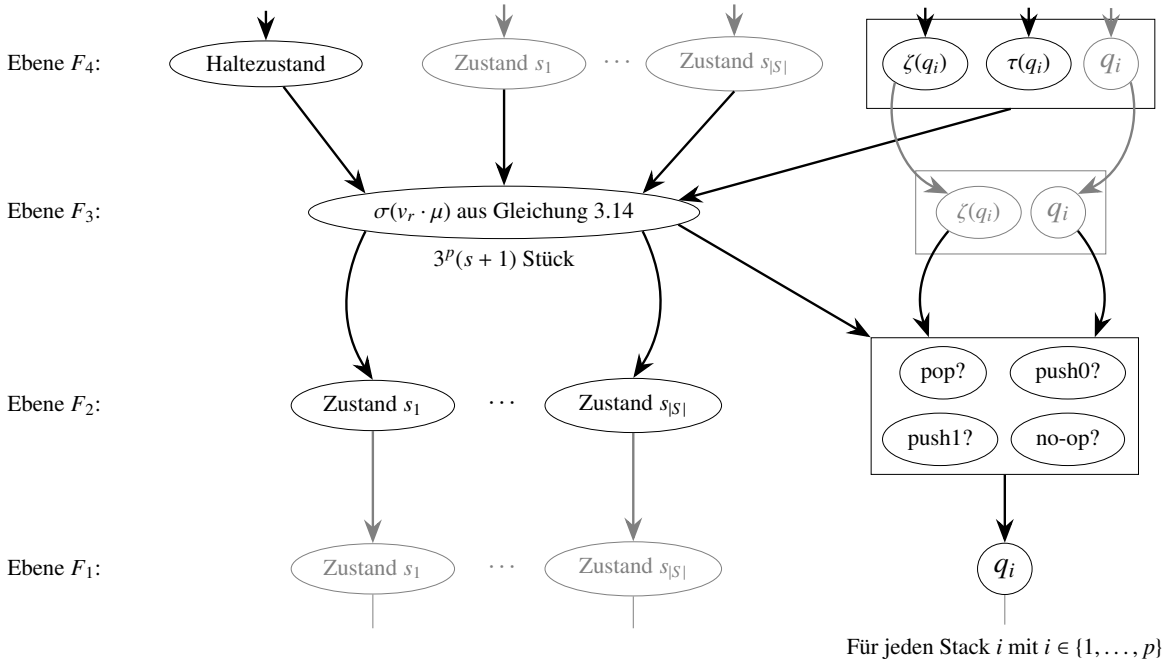

Abbildung 3.13: Aufbau des Prozessornetzwerks

Berechnungen durchführt.

Die Ebenen F_2 bis F_4 sind Zwischenschritte der Berechnungen, wohingegen in F_1 das Endergebnis jeder Turingmaschinenoperation mit dem neuen Zustand sowie allen Stackinhalten vorliegt.

Ausgehend davon wird in Ebene F_4 der Haltezustand x_0 nach 3.9 berechnet sowie die ζ- und τ-Werte aller Stacks bestimmt. In Ebene F_3 werden dann die Summanden

$$\sigma(\langle v_r, (1, \zeta(q_1), \ldots, \zeta(q_p), \tau(q_1), \ldots, \tau(q_p), x_j)\rangle) = \sigma(\langle v_r, \mu\rangle) \tag{3.14}$$

der Formeln 3.11 und 3.12 berechnet. Dafür werden nur $3^p(s+1)$ Prozessoren benötigt, da die ζ- und τ-Werte voneinander abhängig sind und die restlichen Werte daher mit Linearkombinationen der vorhandenen berechnet werden können.

Aus den Summanden wird in Ebene F_2 der Zustand x_j sowie die vier Teilformeln aus 3.12 berechnet, die den Operationen *pop*, *push 0*, *push 1* sowie keiner Operation entsprechen. Diese Teilformeln werden zuletzt in Ebene F_1 zu den neuen q-Werten zusammengesetzt und die Berechnung eines Zeitschritts der Turingmaschine ist abgeschlossen.

Zudem sorgen die Zähler der einzelnen Ebenen dafür, dass während der Berechnungen von Ebene F_2 bis F_4 keine Stackinhalte als Ausgaben interpretiert werden können. Die Werte der

vier Prozessoren werden wie folgt erhalten:

$$\begin{aligned} y_1^+ &= \sigma(y_2), \\ y_2^+ &= \sigma(y_3), \\ y_3^+ &= \sigma(y_4), \\ y_4^+ &= \sigma(1 - y_2 - y_3 - y_4). \end{aligned} \tag{3.15}$$

Dabei signalisiert $y_1 = 1$, dass ein Vielfaches von vier und damit ein gültiger Endzustand erreicht ist.

Dieses Prozessornetzwerk, welches mit der Abbildung $\widetilde{\mathcal{P}}$ arbeitet, berechnet nun die gleiche Funktion Φ wie die Turingmaschine mit der Abbildung $\mathcal{P}$. Dabei gilt, dass das Prozessornetzwerk dafür $4T$ Schritte benötigt, wenn T der Anzahl der Schritte der Turingmaschine entspricht. Für eine neue Anordnung der Prozessoren, in welcher der für die Ausgabe zuständige Prozessor die zweite Stelle im Vektor x repräsentiert, werden alle geforderten Bedingungen erfüllt und es gilt daher Satz 5. □

Wie in Abschnitt 3.1 erwähnt können die entfernten Ein- und Ausgaben wieder hinzugefügt werden. Dazu wird ein Prozessornetzwerk verwendet, das die folgenden drei Schritte ausführt: die Codierung der Eingabe, die Emulation des Netzes ohne Ein- und Ausgaben sowie die Decodierung der Ausgabe.

Für die Codierung der Eingabe wird ein 5-Prozessornetzwerk aufgebaut, welches über die Datenleitung u_1 und die Validierungsleitung u_2 die Eingabe empfängt und nach der Berechnung den Wert $\delta[\omega]$ im Prozessor y_4 ausgibt. Die fünf Prozessoren führen dabei die folgenden Berechnungen aus:

$$\begin{aligned} y_1^+ &= \sigma\left(\frac{1}{4}y_1 + \frac{1}{2}u_1 + \frac{1}{4} + u_2 - 1\right), \\ y_2^+ &= \sigma(u_2), \\ y_3^+ &= \sigma(y_2 - u_2), \\ y_4^+ &= \sigma(y_1 + y_2 - u_2 - 1), \\ y_5^+ &= \sigma(y_3 + y_5). \end{aligned} \tag{3.16}$$

In y_1 liegt in jedem Zeitschritt der δ-Wert vom bisher gelesenen Teil des Wortes ω vor. Dieser wird mit jedem neuen Binärsymbol α_1 der Datenleitung u_1 aktualisiert, indem die vorhandene Codierung mithilfe von $\frac{1}{4}y_1$ um eine Stelle nach rechts verschoben und anschließend der neue Wert $\frac{1}{2}u_1 + \frac{1}{4}$ addiert wird. Dieser Term geht direkt aus 3.2 hervor, da für $i = 1$ der Summand den Wert $\frac{2\alpha_1+1}{4}$ annimmt. Zuletzt sorgt $u_2 - 1$ dafür, dass nur Daten übernommen werden, wenn die Validierungsleitung u_2 auch eins ist. Die endgültige Lösung wird dann zum Zeitpunkt $t = |\omega| + 2$ in y_4 übertragen. Für die Bestimmung dieses Zeitpunktes sind die anderen drei Prozessoren zuständig. Im Zeitabschnitt vor der Übertragung der Ausgabe hat

der Prozessor y_2 den Wert eins, wohingegen der Abschnitt nach der Übertragung durch $y_5 = 1$ erkennbar ist. Der Prozessor y_3, der genau im Zeitschritt $t = |\omega| + 1$ den Wert eins bekommt, d. h. wenn das Lesen und Verarbeiten der Eingabe vollständig abgeschlossen ist, kennzeichnet die Übertragung der Ausgabe nach y_4.

Anschließend wird ausgehend davon mit der in y_4 vorliegenden Codierung das zuvor beschriebene Prozessornetzwerk ohne Eingaben ausgeführt.

Zum Schluss ist es noch notwendig die berechnete codierte Ausgabe $\delta\left[\Phi(\omega)\right]$ zu decodieren und als Binärzahl auszugeben. Dies wird vom folgenden Netzwerk realisiert:

$$
\begin{aligned}
z_1^+ &= \sigma(u_2 + z_1), & z_6^+ &= \sigma(4z_4 + z_1 - 2z_2 - 3),\\
z_2^+ &= \sigma(z_1), & z_7^+ &= \sigma(16z_8 - 8z_7 - 6z_3 + z_6),\\
z_3^+ &= \sigma(z_2), & z_8^+ &= \sigma(4z_8 - 2z_7 - z_3 + z_5),\\
z_4^+ &= \sigma(u_1), & z_9^+ &= \sigma(4z_8),\\
z_5^+ &= \sigma(z_4 + z_1 - z_2 - 1), & z_{10}^+ &= \sigma(z_7).
\end{aligned}
$$

In diesem Netzwerk stellt der Prozessor z_9 die Validierungsleitung und der Prozessor z_{10} die Datenleitung der Ausgabe dar. Im Gegensatz dazu sind u_1 und u_2 Daten- und Validierungsleitung der Eingabe, was bedeutet, dass die Ausgabe $\delta[\Phi(\omega)]$ des vorherigen Netzwerks auf u_1 liegt. Die neuen Datenbits werden jeweils mit der Formel in z_7 bestimmt und anschließend in z_{10} übernommen. Der dazu benötigte und in z_8 gespeicherte Wert $\delta[\Phi(\omega)]$ wird in jedem Zeitschritt um eine Stelle nach links verschoben und abgerundet, um damit das nächste Symbol berechnen zu können. Die Verschiebung wird durch der Operation $4z_8$ gewährleistet, wohingegen $-2z_7 - z_3$ dafür zuständig ist, das vorderste Datenbit zu entfernen und z_5 die Berechnung mit dem Anfangswert zu initialisieren. z_6 übernimmt diese Initialisierungsfunktion für die Berechnung von z_7. Die weiteren in z_7 vorkommenden Terme folgen dem gleichen Schema wie die in z_8, wobei in z_7 eine Verschiebung um zwei Stellen vorgenommen wird, woraus andere Faktoren für z_7 und z_3 resultieren. Es wird daher in jedem Zeitschritt das Datenbit entsprechend der zweiten Nachkommestelle von z_8 berechnet, sodass im folgenden Schritt in z_7 zeitgleich das Datenbit zur ersten Nachkommastelle von z_8 vorliegt. Die übrigen Prozessoren z_1, z_2 und z_4 sind dafür da, die Werte eins sowie $\delta[\Phi(\omega)]$ weiterzugeben, damit diese zum richtigen Zeitpunkt in z_3 sowie z_5 und z_6 sind.

Damit ist es auch möglich Satz 5 für Netzwerke mit Ein- und Ausgaben anzuwenden.

3.4 Konstruktion des Prozessornetzwerks am Beispiel der Palindromsprache

In diesem Abschnitt werden einige Teile des zuvor beschriebenen Vorgehens an einem Beispiel erklärt. Dazu wird ein einfaches Problem betrachtet, an dem die Konstruktion besser veranschaulicht werden kann und welches im Verlauf der Arbeit wiederkehren wird.

Dafür wird die Sprache

$$L = \{\omega \mid \omega \text{ ist ein Palindrom}\}$$

gewählt, was bedeutet, dass für ein Eingabewort ω entschieden werden soll, ob dieses symmetrisch ist.

Bevor auf die Konstruktion des Prozessornetzwerks eingegangen werden kann, wird zunächst eine kurze Beschreibung der Turingmaschine, welche die Sprache L akzeptiert, benötigt. Im Folgenden sind die Zustände einer solchen Turingmaschine aufgeführt.

s_I: Startzustand,

s_0: Suche nach dem Ende des Wortes (mit 0),

s_1: Kontrolle des Symbols (mit 0),

s_2: Suche nach Anfang des Wortes,

s_3: Suche nach dem Ende des Wortes (mit 1),

s_4: Kontrolle des Symbols (mit 1),

s_5: Lesen des ersten Symbols,

s_H: Haltezustand.

Die hier verwendete Turingmaschine arbeitet auf zwei Stacks, wobei das Eingabewort ω zu Beginn vollständig auf dem zweiten Stack liegt. Der erste Stack hingegen ist zum Zeitpunkt $t = 0$ leer. Mit diesem Informationen ist es nun möglich, aus Abbildung 3.13 die Anzahl der benötigten Prozessoren sowie deren Funktion zu bestimmen.

Das Netzwerk wird aus 118 Prozessoren aufgebaut, von denen vier die Zähler darstellen. Die Berechnungen dieser sind in Formel 3.15 explizit angegeben. Die verbleibenden 114 Prozessoren werden wie folgt auf die Ebenen aufgeteilt: 14 in F_4, 76 in F_3, 15 in F_2 und 9 in F_1.

In der Ebene F_4 befinden sich sieben Prozessoren, welche die Werte der Zustände s_I sowie s_0 bis s_5 weitergeben, wohingegen der Wert von s_H von einem Prozessor neu berechnet wird. Jeweils drei Prozessoren sind für die Erhaltung der ζ-, τ- und q-Werte der zwei Stacks zuständig. In der dritten Ebene hingegen gibt es $3^2(7 + 1) = 72$ Prozessoren, welche die

Werte der Summanden aus Formel 3.14 bestimmen, die für die Berechnung sowohl der neuen Stackinhalte als auch der neuen Zuständen verwendet werden. Außerdem speichern vier Prozessoren die ζ- und q-Werte der beiden Stacks. In der Ebene F_2 gibt es sieben Prozessoren für die Berechnung der neuen Werte aller Zustände und jeweils vier für die Wahrheitswerte der möglichen Operationen beider Stacks. Zuletzt werden in sieben von neun Prozessoren der Ebene F_1 die in F_2 erhaltenen Werte der Zustände gespeichert. Die übrigen zwei Prozessoren berechnen die neuen q-Werte der beiden Stacks.

Für den weiteren Verlauf wird das Beispielwort $\omega = 101101_2$ betrachtet, für das sich eine Codierung $\delta[\omega] = 0.313313_4$ ergibt. Daraus folgt, dass der initale Zustand des Prozessornetzwerks die folgende Form hat:

$$(s_I, s_0, s_1, s_2, s_3, s_4, s_5, s_6, s_H, q_1, q_2) = (1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.313313). \tag{3.17}$$

Da die Gewichte in den Formeln 3.11 und 3.12 nicht explizit angegeben sind, ist eine tabellarische Berechnung der Aktivierungswerte ausgehend vom initalen Zustand nicht möglich. Stattdessen wird beispielhaft der erste Zustandsübergang von s_I nach s_3 betrachtet, sodass an der Formel zur Berechnung von s_3 die Aktualisierung des Wertes weitestgehend dargestellt werden kann. Dabei wird vor allem deutlich, welche anderen Aktivierungen benötigt werden und welche Teile der Formel einen Wert ungleich null erhalten.

Zunächst wird dafür die Anzahl der Stacks $p = 2$ sowie Zustände $s = 7$ in die allgemeine Formel für Zustände übertragen, woraus sich

$$s_3^+ = \sum_{j=0}^{7} \sum_{r=1}^{16} c_r \cdot \sigma(\langle v_r, (1, \sigma(4q_1 - 2), \sigma(4q_2 - 2), \sigma(4q_2), \sigma(4q_2), s_j)\rangle)$$

ergibt. Da wie in Formel 3.17 beschrieben der aktuelle Zustand s_I ist, kann daraus gefolgert werden, dass alle Summanden mit $s_j \neq s_I$ den Wert null haben. Dadurch kann die Formel zu

$$s_3^+ = \sum_{r=1}^{16} c_r \cdot \sigma(\langle v_r, (1, \sigma(4q_1 - 2), \sigma(4q_2 - 2), \sigma(4q_2), \sigma(4q_2), s_I)\rangle)$$

vereinfacht werden. Nun können die Werte $q_1 = 0$ und $q_2 = 0.313313$ für die Stacks eingesetzt werden. Des Weiteren werden statt 16 Summanden nur neun benötigt, da es für zwei Stacks genau so viele mögliche Kombinationen der vier τ- und q-Werte gibt. Daraus lässt sich

$$s_3^+ = \sum_{r=1}^{9} c_r \cdot \sigma(\langle v_r, (1, 0, 1, 0, 1, 1)\rangle)$$

ableiten. Die vier Terme $\sigma(4q_1 - 2)$, $\sigma(4q_2 - 2)$, $\sigma(4q_2)$ und $\sigma(4q_2)$ entsprechen dabei den Berechnungen der Ebene F_4. Die neun Summanden $\beta = \sigma(\langle v_r, (1, 0, 1, 0, 1, 1)\rangle)$ erhalten dann in Ebene F_3 ihren neuen Wert, wobei alle übrigen der 72 Prozessoren den Wert null haben,

	init	1	2	3	4	5	6	7	8	9
u_1	0	1	0	1	1	0	1	0	0	0
u_2	0	1	1	1	1	1	1	0	0	0
y_1	0	0	0.3_4	0.13_4	0.313_4	0.3313_4	0.13313_4	0.313313_4	0	0
y_2	0	0	1	1	1	1	1	1	0	0
y_3	0	0	0	0	0	0	0	0	1	0
y_4	0	0	0	0	0	0	0	0	0.313313_4	0
y_5	0	0	0	0	0	0	0	0	0	1

Tabelle 3.18: Codierung der Eingabe

weil sie für einen anderen aktuellen Zustand stehen. Dabei repräsentiert jeder Summand eine mögliche Kombination von τ_1, τ_2, q_1, q_2 und x_j, dessen Auswertung daher mit jeweils anderen Gewichten vorgenommen wird. In der Ebene F_2 wird schließlich der neue Wert des Prozessors $s_3^+ = \sigma(c_r \cdot \beta) = 1$ bestimmt. Die Berechnung der Stackinhalte verläuft analog.

Zuletzt soll der Verlauf der Codierung anhand der Formeln 3.16 am Beispielwort ω dargestellt werden. Dieser ist in Tabelle 3.18 aufgeführt.

Die Ausgabe besteht für die gewählte Sprache L aus einer einzelnen Ziffer, weshalb diese hier nicht tabellarisch aufgeführt ist.

4 Schaltkreise und neuronale Netze

In diesem Abschnitt folgt der Vergleich von neuronalen Netzen mit Booleschen Schaltkreisen. Dabei wird gezeigt, dass beide Modelle von dem jeweils anderen mit gewissen Zeit- bzw. Platzbeschränkungen simuliert werden können. Dazu werden zunächst die in diesem Abschnitt verwendeten Modelle Prozessornetzwerk und Schaltkreis erklärt und für beide das Konzept der Familie eingeführt. Anschließend folgt die formale Beschreibung der Simulation, wobei beide Richtungen getrennt betrachtet werden. Zum Schluss wird diese Simulation am zuvor eingeführten Beispiel der Sprache der Palindrome verdeutlicht. Die Ergebnisse aus diesem Abschnitt beruhen auf den Inhalten des Papers „Analog Computation via Neural Nets“ von Siegelmann und Sontag [SS94].

4.1 Prozessornetzwerk mit reellen Gewichten

Das in diesem Kapitel verwendete Prozessornetzwerk weist große Ähnlichkeiten zum zuvor beschriebenen Modell auf. Es werden die gleichen Methoden für die Ein- und Ausgaben verwendet sowie die gleiche Aktivierungsfunktion. Das hier verwendete Prozessornetzwerk unterscheidet sich nur in der Wahl seiner Gewichte, da diese nun Werte aus den reellen Zahlen annehmen dürfen, wohingegen zuvor rationalwertige Gewichte verwendet wurden.

Definition 6. *Ein Prozessornetzwerk ist ein dynamisches System mit zwei binären Inputs und der Übergangsfunktion* $\mathcal{F}\colon \mathbb{R}^N \times \{0,1\}^2 \to \mathbb{R}^N$*, wobei*

$$\mathcal{F}(x,u) = \sigma(Ax + b_1u_1 + b_2u_2 + c)$$

ist mit $A \in \mathbb{R}^{N\times N}$*,* $b_1, b_2, c \in \mathbb{R}^N$ *und* $\sigma\colon \mathbb{R}^N \to \mathbb{R}^N$*.*

Die Matrix A sowie die Vektoren b und c enthalten Zahlen aus dem Körper $\mathbb{R}$, können aber wie in Abschnitt 3.1 interpretiert werden.

Die Eigenschaft der Kontinuität der Gewichte erzeugt eine gewisse Resistenz gegen kleine Fehler, die mit *robustness* bezeichnet wird. Dies führt zu der Möglichkeit trotz leicht abweichender Gewichte ein richtiges Ergebnis zu erhalten, was bei rationalen Gewichten nicht möglich ist.

Ein Prozessornetzwerk klassifiziert ein Wort $\omega \in \{0,1\}^+$, falls

(i) nach $\tau \leq T(\omega)$ Schritten eine Eins im Ausgabeprozessor steht, wenn $\omega \in L$ ist.

(ii) nach $\tau \leq T(\omega)$ Schritten eine Null im Ausgabeprozessor steht, wenn $\omega \notin L$ ist.

Gilt diese Eigenschaft für alle $\omega \in \{0,1\}^+$ mit der Funktion $T : \mathbb{N} \to \mathbb{N}$, dann wird die Sprache L vom Prozessornetzwerk in Zeit T akzeptiert. Das bedeutet, dass die vom Prozessornetzwerk berechnete Funktion $\Phi_{\mathcal{N}}$ der Sprache L entspricht und die Berechnungszeit $T_{\mathcal{N}}$ der Funktion T. Dies wird als $T_{\mathcal{N}} = T$ und $\Phi_{\mathcal{N}} = L$ notiert.

NET(T) bezeichnet die Klasse aller von einem Prozessornetzwerk in Zeit $T(n)$ berechenbaren Funktionen $\{0,1\}^+ \to \{0,1\}$.

Um eine Familie von Netzwerken zu definieren, wird die Funktion *q-truncation*(x) benötigt, die eine Zahl $x \in \mathbb{R}$ nach q Bits abschneidet. Wenn die Aktivierung aller Prozessoren eines Prozessornetzwerks mit den Gewichten $\tilde{a}_{ij} = q\text{-}truncation(a_{ij})$, $\tilde{b}_{ik} = q\text{-}truncation(b_{ik})$ und $\tilde{c}_i = q\text{-}truncation(c_i)$ durch

$$x_i^+ = q\text{-}truncation\left(\sigma\left(\sum_{j=1}^{N} \tilde{a}_{ij}x_j + \sum_{k=1}^{M} \tilde{b}_{ik}u_k + \tilde{c}_i\right)\right)$$

berechnet wird, dann wird dieses als *q-truncation* Netzwerk bezeichnet.

Für ein Netzwerk $\mathcal{N}$ gilt, dass die Familie $\mathcal{N}(n)$ dieses Netzwerks die Menge aller $T(n)$-*truncation* Netzwerke ist, deren Anzahl an Prozessoren sowie die Werte der beiden Ausgabeprozessoren in allen Zeitschritten $t \leq T(n)$ mit $\mathcal{N}$ gleich ist.

4.2 Boolesche Schaltkreise

Ein Boolescher Schaltkreis ist ein gerichteter, azyklischer Graph, dessen Knoten in Ebenen entsprechend ihrer Entfernung zur Ausgabe angeordnet sind und ein Label tragen, welches die Knotenart angibt.

Definition 7. *[Vol99] Ein Boolescher Schaltkreis mit n Eingaben und m Ausgaben über der Basis* $B = (\neg, \wedge, \vee)$ *ist ein Tupel*

$$c = (V, E, \beta, \omega)$$

wobei (V, E) *ein gerichteter, azyklischer Graph ist,* $\beta\colon V \to B \cup \{x_1 \dots x_n\}$ *eine Funktion, die den Typ eines Gatters festlegt und* $\omega\colon V \to \{y_1, \dots, y_m\} \cup \{*\}$ *eine Funktion, welche die Ausgabegatter bestimmt. Dabei haben alle Knoten v ohne eingehende Kanten, genannt Eingabegatter, einen einzigartigen Wert* $\beta(v) \in \{x_1 \dots x_n\}$*. Für alle anderen Knoten, die inneren Gatter, gilt* $\beta(v) \in B$*. Ausgabegatter sind mit* $\omega(v) \in \{y_1, \dots, y_m\}$ *einzigartig markiert, wohingegen alle anderen Gatter den Wert* $\omega(v) = *$ *haben.*

In der obersten Ebene befinden sich die Eingabegatter v, welche jeweils die Variable x_i entsprechend $\beta(v)$ repräsentieren. In allen weiteren Ebenen befinden sich innere Gatter w, die mit einer zu $\beta(w)$ passenden Anzahl – d. h. eins für $\neg$ und mindentens zwei für $\wedge$ und $\vee$ – an

anderen Gattern aus der darüber liegenden Ebene verbunden sind. Diese werden ausgewertet, indem die $\beta(w) \in \{\neg, \wedge \vee\}$ auf alle Werte der eingehenden Kanten angewendet wird. Dieses Vorgehen lässt sich bis zur letzen Ebene fortsetzten, in der sich nur Gatter mit $\omega(w) = y_i$ befindet. Dabei wird der von so einem Gatter ausgewertete Wert als Ausgabe des Schaltkreises angesehen.

Wenn für eine Eingabe auf diese Weise eine Ausgabe entsteht, berechnet der Schaltkreis die zugrundeliegende Funktion.

Die Größe $S(n)$ eines booleschen Schaltkreises c mit Eingabelänge n entspricht der Anzahl der in c enthaltenen Gatter. Die Tiefe $D(n)$ hingegen ist durch die Anzahl der Ebenen und die Breite $W(n)$ durch die maximale Anzahl von Gattern einer Ebene definiert.

Wie bei Netzwerken existiert für boolesche Schaltkreise das Konzept der Familie. Eine Familie von Schaltkreisen C ist eine Menge von Schaltkreisen c_n mit $n \in \mathbb{N}$, wobei jedes c_n die Größe $S(n)$, die Tiefe $D(n)$ und die Breite $W(n)$ hat. Jeder Schaltkreis $c_n \in C$ berechnet die gleiche Funktion Φ_C, nimmt dabei aber jeweils nur Eingaben der Länge n an.

Eine Sprache L wird von so einer Familie von Schaltkreisen entschieden, wenn jeder Schaltkreis c_n diejenige Funktion berechnet, welche alle Wörter $\omega \in L \cap \{0,1\}^n$, also alle Wörter der Sprache mit Länge n, akzeptiert. Sie wird als die charakteristische Funktion der Sprache bezeichnet. In diesem Fall entspricht die von der Schaltkreisfamilie berechnete Funktion der Sprache und es gilt $\Phi_C = L$.

Analog zu NET(T) wird für Schaltkreisfamilien die Klasse CIRCUIT(S) definiert, die alle Funktionen enthält, welche von einer Familie von Schaltkreisen C der Größe $S_C(n)$ berechnet werden können.

4.3 Simulation

Mit den Definitionen der Klassen NET(T) und CIRCUIT(S) können die Modelle des Prozessornetzwerks und des Booleschen Schaltkreises nun verglichen werden.

Satz 8. *Sei F eine Funktion mit $F(n) \geq n$. Dann gilt:*

$$CIRCUIT(F(n)) \subseteq NET(n \cdot (F(n))^2), \tag{4.1}$$

$$NET(F(n)) \subseteq CIRCUIT((F(n))^3). \tag{4.2}$$

Verallgemeinert ist die Aussage dieses Satzes, dass die Berechnung jeder Funktion von einem Modell in das andere mit einem polynomiellen Overhead der Laufzeit bzw. der Größe überführt werden kann.

In diesem Abschnitt wird der Beweis der beiden Aussagen 4.1 und 4.2 vorgestellt. Dazu wird aus einem Modell jeweils das andere Modell mit passenden Funktionen T_N und S_C konstruiert, sodass beide Modelle die gleiche Funktion Φ berechnen und somit $\Phi_N = \Phi_C$ gilt.

4.3.1 Simulation von Schaltkreisen durch Prozessornetzwerke

Zunächst wird die Aussage 4.1 mithilfe des Lemmas 9 gezeigt, wobei aus dem Beweis des Lemmas direkt die Aussage 4.1 hervorgeht.

Lemma 9. *Es gibt eine Zahl $N \in \mathbb{N}$, sodass für jede Familie von Schaltkreisen C der Größe $S_C(n)$ ein Prozessornetzwerk $\mathcal{N}$ mit N Prozessoren existiert für das $\Phi_\mathcal{N} = \Phi_C$ und $T_\mathcal{N} = O(nS_C^2(n))$ gilt.*

Beweis. Für die Konstruktion eines Netzwerkes $\mathcal{N}$, welches die gleiche Funktion wie die Familie der Schaltkreise C berechnet, wird zunächst eine Codierung von C als eine reelle Zahl benötigt. Dazu wird zu jeder Familie eine unendliche Folge $e(C)$ definiert und diese anschließend in eine reelle Zahl $\hat{C}$ konvertiert.

In der unendlichen Folge $e(C)$ werden dabei alle Schaltkreise aus C ihrer Größe nach aufsteigend sortiert codiert und mit der Ziffer 8 voneinander getrennt. Dabei kommen die Codierungen $\mathrm{en}[c_n]$ der einzelnen Schaltkreise in umgekehrter Reihenfolge vor. Diese Art von Codierung wird mit $\overline{\mathrm{en}}[c_n]$ bezeichnet.

Ein Schaltkreis c_n wird als $\mathrm{en}[c_n]$, einer Folge der Ziffern 0, 2, 4 und 6, codiert. Die Ziffern 6 und 0 repräsentieren dabei jeweils den Beginn eines neuen Abschnitts, wobei 6 eine neue Ebene des Schaltkreises und 0 ein neues Gatter innerhalb derselben Ebene einleitet. Die Codierung jedes Gatters besteht aus zwei Teilen: Der Codierung der Funktion und der Verbindungen zur vorherigen Ebene. Die Funktion wird repräsentiert durch ein Element der Menge $\{42, 44, 22\}$ entsprechend der Funktion aus $\{\wedge, \vee, \neg\}$. Darauf folgen n Ziffern, wobei n die Anzahl der Gatter aus der vorherigen Ebene beschreibt. Diese zeigen, ob vom entsprechenden Gate der vorherigen Ebene zum aktuellen eine Verbindung besteht. Das Bestehen einer Verbindung wird durch eine vier dargestellt und das Fehlen einer solchen Verbindung durch eine zwei.

Die Folge $e(C) = r_1 r_2 \ldots$ wird durch

$$\hat{C} := \sum_{i=1}^{\infty} \frac{r_i}{9^i} \tag{4.3}$$

in eine Zahl der Cantormenge mit der Basis neun umgewandelt und später als reellwertiges Gewicht im konstruierten Prozessorenetzwerk verwendet. Wie zuvor wird durch die Wahl der Codierung gewährleistet, dass Zahlen leicht unterscheidbar sind. Das liegt daran, dass keine ungeraden Zahlen in der Codierung vergeben werden und daher nur jedes zweite Neuntel ein Intervall darstellt, in dem Zahlen auftreten können.

Diese Art der Codierung ist auch für Eingabewörter ω definiert, wobei in $\mathrm{en}[\omega]$ alle Ziffern $\alpha_i \in \omega$ durch $2\alpha_i + 2$ substituiert sind. Für jede Folge $\mathrm{en}[x]$ bezeichnet $\widehat{\mathrm{en}}[x]$ zudem die Interpretation von $\mathrm{en}[x]$ als Zahl der Basis neun.

Da $e(C)$ eine unendliche Folge unterschiedlicher Zahlen ist, gilt für die resultierende Zahl

$\hat{C} \notin \mathbb{Q}$. Im Gegensatz dazu ist $\mathrm{en}[c_n]$ durch die Größe des Schaltkreises beschränkt, wodurch jeder Teil von $\hat{C}$, welcher der Codierung eines Schaltkreises entspricht, rational ist.

Mit der Nutzung von $\hat{C}$ kann nun ein Prozessornetzwerk konstruiert werden, welches die in Satz 9 verlangten Eigenschaften erfüllt. Dazu wird das Vorgehen in drei Aufgaben unterteilt, die von einzelnen Prozessornetzwerken ausgeführt werden.

1. Die Eingabe ω des Schaltkreises wird als $\widehat{\mathrm{en}}[\omega]$ und die Zahl $|\omega|$ binär codiert.

2. Aus $\hat{C}$ wird mit dem binären $|\omega|$ die Zahl $\widehat{\mathrm{en}}[c_{|\omega|}]$ extrahiert, welche der Codierung des Schaltkreises entspricht, der Eingaben der Größe $|\omega|$ verarbeiten kann.

3. Mithilfe der beiden Zahlen $\widehat{\mathrm{en}}[c_{|\omega|}]$ und $\widehat{\mathrm{en}}[\omega]$ wird der Schaltkreis simuliert und sein Ergebnis ausgegeben.

In Folgenden werden drei Prozessornetzwerke $\mathcal{N}_I$, $\mathcal{N}_R$ und $\mathcal{N}_S$ beschrieben, die jeweils eine der aufgeführten Aufgaben lösen. Da diese zusammengefügt werden können, indem die Bearbeitung einer Aufgabe begonnen wird, nachdem die vorherige abgeschlossen ist, ergibt sich daraus das gewünschte Prozessornetzwerk.

Die Bestimmung von $\widehat{\mathrm{en}}[\omega]$ und $|\omega|_2$ kann mit dem 6-Prozessornetzwerk $\mathcal{N}_I$ mit Datenleitung u_1 und Validierungsleitung u_2 durchgeführt werden, dessen Prozessoren die Berechnungen

$$
\begin{aligned}
y_1^+ &= \left(\frac{1}{9}y_1 + \frac{2}{9}u_1 + \frac{2}{9} + u_2 - 1\right),\\
y_2^+ &= \left(\frac{1}{2}y_2 + \frac{1}{2}u_2\right),\\
y_3^+ &= (u_2),\\
y_4^+ &= (y_3 - u_2),\\
y_5^+ &= (y_1 + y_3 - u_2 - 1),\\
y_6^+ &= (y_2 + y_3 - u_2 - 1)
\end{aligned}
$$

ausführen. Dabei wird in y_1 jeweils der aktuelle Wert der berechneten Codierung $\widehat{\mathrm{en}}[\omega]$ und in y_2 derjenige von $|\omega|_2$ gespeichert. In jedem Schritt wird mit $\frac{1}{9}y_1$ ein Rechtsshift der schon vorhandenen Zahl durchgeführt und mit $\frac{2}{9}u_1 + \frac{2}{9}$ das neue Datenbit 2 oder 4 addiert. Dieses Vorgehen gilt analog für y_2. Die beiden Prozessoren y_5 und y_6 sind für die Ausgabe der beiden Werte zuständig, wohingegen y_4 die Validierungsleitung dieser Ausgabe darstellt.

Um aus $\hat{C}$ die Zahl $\widehat{\mathrm{en}}[c_{|\omega|}]$ zu erhalten, werden zunächst die zwei Funktionen Ξ und Λ benötigt. Da diese in ihrer ursprünglichen Form aber nicht von einem Prozessornetzwerk berechenbar sind, werden im Folgenden zudem die Approximationen $\tilde{\Lambda}$ und $\widetilde{\Xi}$ angegeben.

$$\Lambda[z] \coloneqq \begin{cases} 0, & x < 0, \\ 9z - \lfloor 9z \rfloor, & 0 \leq z \leq 1, \\ 1, & z > 1 \end{cases} \quad , \quad \tilde{\Lambda}[z] = \sum_{j=0}^{8} (-1)^j \sigma(9z - j).$$

Die Funktion Λ führt einen Linksshift aus, da eine Multiplikation mit der Basis alle Ziffern um eine Stelle nach links verschiebt und die Operation $\lfloor 9z \rfloor$ die größte Ziffer selektiert, die entfernt werden soll. In $\tilde{\Lambda}$ wird das gleiche Verfahren ausgenutzt, denn der Summand aus $\tilde{\Lambda}$ nimmt die Form $9z - \lfloor 9z \rfloor$ an, wenn $j = \lfloor 9z \rfloor$ gilt, da σ keine Veränderung an einer Zahl im Intervall $[0, 1]$ verursacht und das Vorzeichen positiv ist, da nur gerade Zahlen in der Codierung auftreten können. Die anderen acht Summanden sind derart gewählt, dass sie sich gegenseitig auslöschen und können in zwei Kategorien eingeteilt werden. In allen Fällen, in denen $j > \lfloor 9z \rfloor$ ist, sind die Summanden gleich null und haben keine Auswirkungen auf das Ergebnis. Für $j < \lfloor 9z \rfloor$ sind die Summanden zwar eins, löschen sich durch die alternierenden Vorzeichen aber gegenseitig aus, da es durch die Wahl der Ziffern in der Codierung immer eine gerade Anzahl solcher Summanden gibt. Für alle Zahlen der gewählten Codierung liefern die Funktionen Λ und $\tilde{\Lambda}$ daher die gleichen Ergebnisse.

$$\Xi[z] \coloneqq \begin{cases} 0, & z < 0, \\ 2\lfloor \frac{9z}{2} \rfloor, & 0 \leq z \leq 1, \\ 1, & z > 1 \end{cases} \quad , \quad \widetilde{\Xi}[z] = 2 \sum_{j=0}^{3} \sigma(9z - (2j+1)).$$

Die Funktion Ξ hingegen selektiert die erste Stelle nach dem Komma und damit die Ziffer mit der höchsten Wertigkeit. Genau wie bei der Λ-Funktion wird ein Shift durch Multiplikation durchgeführt und alle Nachkommastellen abgeschnitten. Diesmal wird nach der Multiplikation zusätzlich durch zwei geteilt, um die vorhandenen Zahlen von der Menge $\{0, 2, 4, 6, 8\}$ auf das Intervall $[0, 4] \in \mathbb{N}$ abzubilden. Da z nur gerade Ziffern enthält, wird durch die anschließende Multiplikation mit zwei das richtige Ergebnis hergestellt. Diese Vorgehensweise dient der Approximierbarkeit, denn die Berechnung von $\lfloor \frac{9z}{2} \rfloor$ kann als die Summe in $\widetilde{\Xi}$ ausgedrückt werden. Dabei wird davon Gebrauch gemacht, dass immer genau die Hälfte der Zahlen kleiner als $9z$ ungerade sind, sodass aus dieser Anzahl das Ergebnis bestimmt werden kann. Um diese zu erhalten, werden von der Zahl $9z$ jeweils alle ungeraden Zahlen subtrahiert, wobei für jedes $2j + 1 < 9z$ ein Summand mit dem Wert eins entsteht.

Mit diesen beiden Funktionen kann nun die Codierung des Schaltkreises $c_{|\omega|}$ aus der Codierung der Familie extrahiert werden. Dazu wird das in Algorithmus 1 beschriebene Vorgehen verwendet. Der angegebene Pseudocode dient dabei lediglich dem Verständnis dieses Vorgehens.

Algorithmus 1 besteht aus zwei Schleifen, von denen die erste dafür zuständig ist, den Beginn von $\overline{\text{en}}\,[c_{|\omega|}]$ zu finden, indem die Anzahl der auftretenden Achten gezählt wird, welche jeweils die Codierung eines neuen Schaltkreises einleiten. Wenn der Schaltkreis passender

Algorithmus 1 Extrahiert die Codierung $\widehat{\text{en}}[c_n]$ aus der Codierung $\hat{C}$

```
function RETRIEVAL(Ĉ,n)
    counter ← 0, y ← 0, z ← Ĉ
    while counter < n do
        z ← Λ[z]
        if Ξ[z] = 8 then
            counter ← counter + 1
        end if
    end while
    while Ξ[z] < 8 do
        z ← Λ[z]
        y ← 1/9 (y + Ξ[z])
    end while
    return y
end function
```

Größe gefunden wurde, sorgt die zweite Schleife dafür, dass jedes gelesene Symbol die Zahl $\widehat{\text{en}}[c_n]$ aktualisiert.

Das Prozessornetzwerks $\mathcal{N}_R$, welches diesen Algorithmus ausführt, besteht aus 17 Prozessoren, deren Werte durch

$$
\begin{aligned}
x_i^+ &= \sigma(9x_{10} - i), \quad i = \{0, \ldots, 8\}, \\
x_9^+ &= \sigma(2u), \\
x_{10}^+ &= \sigma\left(\hat{C}x_9 + x_0 - x_1 + x_2 - x_3 + x_4 - x_5 + x_6 + x_7 + x_8\right), \\
x_{11}^+ &= \sigma\left(\frac{1}{9}x_{12} + \frac{2}{9}\left(x_1 + x_3 + x_5 + x_7\right) - 2x_{13}\right), \\
x_{12}^+ &= \sigma(x_{11}), \\
x_{13}^+ &= \sigma(u + x_{14} + x_{15}), \\
x_{14}^+ &= \sigma(2x_{13} + x_7 - 2), \\
x_{15}^+ &= \sigma(x_{13} - x_7), \\
x_{16}^+ &= \sigma(x_{12} + x_7 - 1)
\end{aligned}
\tag{4.4}
$$

bestimmt werden. Die Eingabe u repräsentiert das zuvor berechnete $|\omega|_2$ und es gilt $u(1) = 1 - 2^{-|\omega|}$ sowie $u(t) = 0$ für alle $t \neq 1$. Die Prozessoren x_0 bis x_8 berechnen die Summanden, die in den Formeln $\tilde{\Lambda}$ und $\widetilde{\Xi}$ auftreten. x_{10} stellt die Operation $z \leftarrow \Lambda[z]$ dar, wobei in allen ungeraden Schritten der aktualisierte Wert und in allen geraden der Wert null vorliegen. Diese Unterscheidung resultiert daraus, dass die Berechnung in zwei Schritten abläuft, sodass im ersten die Prozessoren x_0 bis x_8 aus dem in x_{10} stehenden Wert die Summanden berechnen und im zweiten der Prozessor x_{10} aus diesen seinen neuen Wert zusammensetzt. Um zunächst die Zahl $\hat{C}$ als Startwert in den Prozessor x_{10} zu laden, dient der Prozessor x_9, welcher nur im zweiten Schritt den Wert eins hat.

Die beiden Prozessoren x_{11} und x_{12} repräsentieren die Operation $y \leftarrow \frac{1}{9}(y + \Xi[z])$, wobei die Aktualisierung des Werts in x_{12} wieder alle zwei Schritte vorgenommen wird. Der Teil $\frac{2}{9}(x_1 + x_3 + x_5 + x_7)$ steht dabei für $\frac{1}{9}\Xi[z]$, wohingegen $-2x_{13}$ dafür sorgt, dass die Berechnung erst beim Schaltkreis richtiger Größe gestartet wird. Der Prozessor x_{13} wird als Zähler verwendet, in dem alle zwei Schritte der aktuelle Wert der bei $1 - 2^{-|\omega|}$ startenden Zahl steht. Dieser wird mit den Werten der beiden Prozessoren x_{14} und x_{15} aktualisiert, die jeweils der Herabsetzung sowie der Erhaltung des Wertes dienen. Im Prozessor x_{16} liegt am Ende die Ausgabe $\widehat{\text{en}}[c_n]$ vor.

Zur Simulation des Schaltkreises mit den vorhandenen Werten $\widehat{\text{en}}\left[c_{|\omega|}\right]$ und $\widehat{\text{en}}[\omega]$ wird auf das Ergebnis des vorherigen Kapitels zurückgegriffen, da es in diesem Fall ersichtlicher ist, wie die Simulation auf einer Turingmaschine durchgeführt werden kann.

Dazu wird eine Turingmaschine mit drei Bändern betrachtet, auf deren erstem Band die Codierung des Schaltkreises $\widehat{\text{en}}[c_{|\omega|}]$ gespeichert wird und als Anweisung für die auf den anderen Bändern auszuführenden Schritte dient. Da diese Codierung konkret beschreibt, welche Gatter auf welche Weise miteinander verknüpft sind, kann in jedem Schritt direkt festgestellt werden, welche Informationen in welcher Form verwendet werden sollen, um den Wert des nächsten Gatter zu berechnen. Die Zwischenergebnisse jeder Ebene werden auf dem zweiten Band gespeichert und können daher verwendet werden, um die Werte der nächsten Ebene zu bestimmen. Die Ein- und Ausgabe befindet sich auf dem dritten Band, welches daher zu Beginn benutzt wird, um die Ergebnisse der ersten Ebene zu erhalten und am Ende, um den Wert des Wurzelknotens auszugeben, wenn die Codierung vollständig verarbeitet wurde.

Jedes Band kann mit zwei Stacks repräsentiert werden, indem alle sich auf der linken Seite des Lesekopfes befindenden Inhalte auf dem einem Stack abgelegt werden und alle Inhalte auf der rechten Seite des Lesekopfes auf dem anderen. Eine Bewegung wird dann durch eine *push*-Operation auf dem einen und eine *pop*-Operation auf dem anderen Stack dargestellt. Für diese Turingmaschine kann Satz 5 angewendet werden, woraus folgt, dass es ein Prozessornetzwerk gibt, welches diese Simulation durchführen kann.

Zum Schluss folgt die Laufzeitabschätzung der drei Prozessornetzwerke $\mathcal{N}_I$, $\mathcal{N}_R$ und $\mathcal{N}_S$, um eine Gesamtlaufzeit von $O(|\omega| \cdot S(|\omega|))$ zu zeigen.

Die Laufzeit von $\mathcal{N}_I$ liegt in $O(|\omega|)$, da die für jedes Symbol in ω ausgeführte Substitution in konstanter Zeit erfolgt. Das Prozessornetzwerk $\mathcal{N}_R$ hingegen hat eine Laufzeit die abhängig von $\left|\text{en}\left[c_{|\omega|}\right]\right|$ und $|\omega|$ ist, da jedes Symbol von $\hat{\mathcal{C}}$ in konstanter Zeit verarbeitet wird, aber die Codierungen aller Schaltkreise bis zur Größe $|\omega|$ gelesen werden müssen. Aufgrund der Sortierung der Schaltkreise nach ihrer Größe lässt sich die Größe aller gelesenen Schaltkreise c_i mit $i \leq |\omega|$ durch $c_{|\omega|}$ abschätzen. Die Länge der Codierung $\text{en}\left[c_{|\omega|}\right]$ kann mit $O(W(|\omega|)) \cdot (S(|\omega|)) = S^2(|\omega|)$ ausgedrückt werden und es ergibt sich eine Laufzeit von $O(|\omega| \cdot S^2(|\omega|))$.

Für die Bestimmung der Laufzeit von $\mathcal{N}_S$ muss zunächst die Laufzeit der Turingmaschine betrachtet werden. Da diese jedes Symbol der Codierung $\text{en}[c_{|\omega|}]$ lesen und auf dem zweiten

Band pro Gate wieder zum Anfang der Ebene zurückkehren muss, ergibt sich für diesen Teil eine Laufzeit von $O(S^2(|\omega|))$. Zusätzlich werden $O(|\omega|)$ Schritte für das Auslesen der Eingabe benötigt, weshalb diese Turingmaschine insgesamt eine Laufzeit von $O(S^2(|\omega|) + |\omega|)$ hat. Nach Satz 5 ist die Laufzeit von $\mathcal{N}_S$ daher in $O(S^2(|\omega|) + |\omega|)$. Insgesamt ist daher die Laufzeit von $\mathcal{N}_R$ für die Gesamtlaufzeit des Prozessornetzwerkes ausschlaggebend und die gewünschte Schranke von $O(|\omega| \cdot S^2(|\omega|))$ wird eingehalten. □

4.3.2 Simulation von Prozessornetzwerken durch Schaltkreise

In diesem Abschnitt wird der zweite Teil von Satz 8, die Aussage 4.2, gezeigt. Dazu wird das Lemma 10 verwendet, aus dem direkt die Aussage 4.2 folgt.

Lemma 10. *Sei $\mathcal{N}$ ein Prozessornetzwerk, welches in Zeit $T : \mathbb{N} \to \mathbb{N}$ eine Lösung berechnet. Dann gibt es eine Familie von Schaltkreisen $C(\mathcal{N})$ mit Größe $O(T^3)$, Tiefe $O(T \log(T))$ und Breite $O(T^2)$, die die gleiche Sprache wie $\mathcal{N}$ akzeptiert.*

Um diese Aussage zu beweisen wird zunächst ein einzelnes Netzwerk $\mathcal{N}$ in eine Familie von Netzwerken $\mathcal{N}_1(|\omega|)$ überführt. Anschließend wird für $\mathcal{N}_1(|\omega|)$ eine Familie von Schaltkreisen konstruiert, welche die gleiche Sprache akzeptiert.

Beweis. Die Ausgabeprozessoren der Familie von Netzwerken $\mathcal{N}_1(|\omega|)$ sollen die Funktion $\sigma(2\tilde{f} - 0.5)$ berechnen, wobei $\tilde{f}$ der Funktion f entspricht, aber auf die Prozessoren von $\mathcal{N}_1(n)$ angewendet und nach $T(n)$-Bit abgeschnitten wird. Da bei einer Familie die Bedingung gilt, dass die Ausgaben von $\mathcal{N}_1(n)$ mit denen von $\mathcal{N}$ für alle Zeitpunkte $t \leq T(n)$ identisch sein müssen, wird eine Fehlerabschätzung angegeben, aus der diese Bedingung folgt. Dazu werden einige Variablen benötigt, die im folgenden aufgeführt sind. Die Zahl N beschreibt die Anzahl der Prozessoren, wohingegen M die Anzahl der Eingabeleitungen darstellt. $L = N + M + 1$ fasst beide in einem Wert zusammen. Der Wert W' entspricht dem betragsmäßig größten Gewicht und es gilt $W = W' + 1$. Die Fehler der Gewichte, welche durch die *truncation*-Funktion entstehen, werden mit δ_w und die analog bei den Prozessoren entstehenden Fehler mit δ_p bezeichnet und sind positiv. Zuletzt bezeichnet ε_t den Gesamtfehler der Aktivierung eines Prozessors x_i zum Zeitpunkt t. Alle Gewichte werden zunächst als positiv angenommen. Daraus können die folgenden Fehlerabschätzungen abgeleitet werden:

$$
\begin{aligned}
|\tilde{x}_i(t) - x_i(t)| &= x_i(t) - \tilde{x}_i(t) \leq \varepsilon_t, \\
|\tilde{a}_i(t) - a_i(t)| &= a_i(t) - \tilde{a}_i(t) \leq \delta_w, \\
|\tilde{b}_i(t) - b_i(t)| &= b_i(t) - \tilde{b}_i(t) \leq \delta_w, \\
|\tilde{c}_i(t) - c_i(t)| &= c_i(t) - \tilde{c}_i(t) \leq \delta_w.
\end{aligned}
$$

Da der Fehler ε_t so klein sein soll, dass die beiden Funktionen f und $\tilde{f}$ den gleichen Wert erhalten, wird eine obere Schranke benötigt, die diese Bedingung erfüllt. Diese Schranke wird als

$$\varepsilon_t \leq N(W' + \delta_w)\varepsilon_{t-1} + L\delta_w + \delta_p$$

gewählt. Es folgt eine Herleitung dieser Aussage:

$$\begin{aligned}
& N(W' + \delta_w)\varepsilon_{t-1} + (N + M + 1)\delta_w + \delta_p \\
\geq & \sum_{j=1}^{N} ((W' + \delta_w)\varepsilon_{t-1} + \delta_w) + (M + 1)\delta_w + \delta_p \\
\geq & \sum_{j=1}^{N} ((W' + a_{ij} - \tilde{a}_{ij})(x_j - \tilde{x}_j) + a_{ij} - \tilde{a}_{ij}) + (M + 1)\delta_w + \delta_p \\
= & \sum_{j=1}^{N} \left(a_{ij}x_j + W'\left(x_j - \tilde{x}_j\right) - \tilde{a}_{ij}x_j - a_{ij}\tilde{x}_j + \tilde{a}_{ij}\tilde{x}_j + a_{ij} - \tilde{a}_{ij}\right) + (M + 1)\delta_w + \delta_p \\
\geq & \sum_{j=1}^{N} \left(a_{ij}x_j + \tilde{a}_{ij}\left(x_j - \tilde{x}_j\right) - \tilde{a}_{ij}x_j - a_{ij}\tilde{x}_j + \tilde{a}_{ij}\tilde{x}_j + a_{ij} - \tilde{a}_{ij}\right) + (M + 1)\delta_w + \delta_p \\
= & \sum_{j=1}^{N} \left(a_{ij}x_j + \underbrace{\tilde{a}_{ij}x_j - \tilde{a}_{ij}x_j}_{=0} - \tilde{a}_{ij}\tilde{x}_j \underbrace{- a_{ij}\tilde{x}_j + \tilde{a}_{ij}\tilde{x}_j}_{=x_j(\tilde{a}_{ij} - a_{ij})} + a_{ij} - \tilde{a}_{ij} \right) + (M + 1)\delta_w + \delta_p \\
= & \sum_{j=1}^{N} (a_{ij}x_j - \tilde{a}_{ij}\tilde{x}_j - \tilde{x}_j(a_{ij} - \tilde{a}_{ij}) + 1(a_{ij} - \tilde{a}_{ij})) + (M + 1)\delta_w + \delta_p \\
= & \sum_{j=1}^{N} \left(a_{ij}x_j - \tilde{a}_{ij}\tilde{x}_j + \underbrace{\left(1 - \tilde{x}_j\right)\left(a_{ij} - \tilde{a}_{ij}\right)}_{\geq 0} \right) + (M + 1)\delta_w + \delta_p \\
\geq & \sum_{j=1}^{N} \left(a_{ij}x_j - \tilde{a}_{ij}\tilde{x}_j\right) + \sum_{k=1}^{M} (\delta_w u_k) + \delta_w + \delta_p \\
\geq & \sum_{j=1}^{N} \left(a_{ij}x_j - \tilde{a}_{ij}\tilde{x}_j\right) + \sum_{k=1}^{M} (b_{ik}u_k - \tilde{b}_{ik}u_k) + c_i - \tilde{c}_i + \delta_p \\
= & \underbrace{\left(\sum_{j=1}^{N} a_{ij}x_j + \sum_{k=1}^{M} b_{ij}u_k + c_i \right) - \left(\sum_{j=1}^{N} \tilde{a}_{ij}\tilde{x}_j + \sum_{k=1}^{M} \tilde{b}_{ik}u_k + \tilde{c}_i - \delta_p \right)}_{\sigma \text{ ist lipschitzstetig mit der Lipschitzkonstante 1}} \\
\geq & \sigma\left(\sum_{j=1}^{N} a_{ij}x_j + \sum_{k=1}^{M} b_{ij}u_k + c_i \right) - \sigma\left(\sum_{j=1}^{N} \tilde{a}_{ij}\tilde{x}_j + \sum_{k=1}^{M} \tilde{b}_{ik}u_k + \tilde{c}_i - \delta_p \right) \\
= & x_i(t) - \tilde{x}_i(t) \\
= & \varepsilon_t.
\end{aligned}$$

Für negative Gewichte kann diese Schranke analog hergeleitet werden. Daraus folgt, dass ε_t mit der gegebenen Schranke abgeschätzt werden kann. Diese kann mit $L = N + M + 1$ weiter vereinfacht werden.

$$\begin{aligned}
\varepsilon_t &\leq L(W' + \delta_w)\varepsilon_{t-1} + L\delta_w + \delta_p \\
&\leq LW\varepsilon_{t-1} + L\delta_w + \delta_p \\
&= \sum_{i=0}^{t-1} (LW)^i (L\delta_w + \delta_p) \\
&\leq (LW)^t (L\delta_w + \delta_p)
\end{aligned}$$

Um die geforderte Bedingung der Gleichheit zu erfüllen, muss $f \leq 0 \Rightarrow \tilde{f} < \frac{1}{4}$ und $f \geq 1 \Rightarrow \tilde{f} > \frac{3}{4}$ gelten. Daraus folgt, dass der Fehler $\varepsilon_t < \frac{1}{4}$ für alle Zeitpunkte $t \leq T(n)$ sein muss. Die Bedingung kann daher zu

$$(L\delta_w + \delta_p) \leq \frac{1}{4}(LW)^{-t}$$

umgestellt werden. Dies ist erfüllt, wenn die Fehler δ_w und δ_p, welche durch die truncation-Funktion entstehen kleiner als $\frac{1}{8}(LW)^{-(t-1)}$ sind, da

$$\begin{aligned}
(L\delta_w + \delta_p) &= L\frac{1}{8}(LW)^{-(t-1)} + \frac{1}{8}(LW)^{-(t-1)} \\
&= \underbrace{(L+1)}_{\leq LW} \frac{1}{8}(LW)^{-(t-1)} \\
&\leq \frac{1}{8} LW^{-t}
\end{aligned}$$

gilt. Das bedeutet, dass die Gewichte sowie Aktivierungen der Prozessoren nach $O(t \log(LW))$ Bits abgeschnitten werden müssen. Damit gibt es zu jedem Netzwerk $\mathcal{N}$ eine Familie $\mathcal{N}_1(n)$ von $T(n)$-truncation Netzwerken, da L und W Konstanten sind und somit $O(t \log(LW)) \subseteq O(T)$ gilt.

Zu der Familie von Netzwerken $\mathcal{N}_1(|\omega|)$ kann nun eine Schaltkreisfamilie C der Größe $O(T^3)$, Tiefe $O(T \log(T))$ und Breite $O(T^2)$ konstruiert werden. Dabei entsteht für jedes Netzwerk $\mathcal{N}(|\omega|)$ der entsprechende Schaltkreis $c_{|\omega|}$ aus C. Im Folgenden wird der Aufbau eines solchen Schaltkreises $c_{|\omega|}$ beschrieben.

Ein Schaltkreis $c_{|\omega|}$ wird aus den drei Grundschaltkreisen für die Eingabe, Ausgabe und die Berechnung eines Prozessors aufgebaut. Der Eingabeschaltkreis sorgt dafür, dass die einzelnen Bits des Wortes ω, die zu Beginn am Schaltkreis anliegen so lange erhalten bleiben bis sie in der Berechnung verwendet werden, um das schrittweise Lesen der Eingabebits im Netzwerk zu simulieren. Dies wird dadurch gewährleistet, dass alle Bits in jedem Schritt mit der Konstanten eins $\wedge$-verknüpft werden, bis diese in eine Berechnung eingegangen

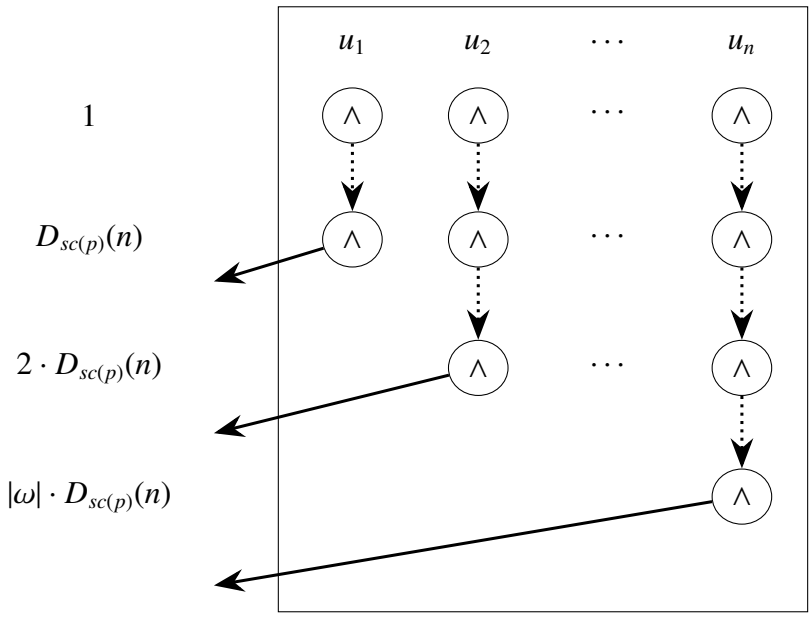

Abbildung 4.5: Aufbau des Eingabeschaltkreises

sind. Dabei wird das Bit i des Wortes ω im Schritt $iD_{sc(p)}(n)$ als Eingabe an die für die Berechnung zuständigen Schaltkreise weitergegeben, wobei $D_{sc(p)}(|\omega|)$ für die Tiefe eines solchen Schaltkreises steht. Dieser Aufbau ist in Abbildung 4.5 skizziert.

Die Berechnungen werden über einzelne Schaltkreise $sc(p)$ für jeden Prozessor p im Netzwerk ausgeführt. Da jeder Prozessor eine Linearkombination von N Aktivierungen anderer Prozessoren sowie zwei Datenleitungen berechnet, kann dieses Vorgehen auf zwei Additionen für die zwei Datenleitungen und $T \cdot N$ Additionen für die N Aktivierungen, bei denen eine Mutliplikation mit dem Gewicht jeweils als T Additionen ausgedrückt werden kann, reduziert werden. Die Gewichte sind dabei in jedem Schaltkreis als Konstanten festgesetzt, was bedeutet, dass jeder konstruierte Schaltkreis, der einen Prozessor simuliert, insgesamt die Summe von $(TN + 2)$ $(2T)$-Bit Zahlen berechnen muss. Dass die Größe der Zahlen $2T$-Bit beträgt, folgt daraus, dass bei einer Multiplikation von zwei T-Bit Zahlen höchstens eine $2T$-Bit Zahl entstehen kann. Mit der *carry-look-ahead* Methode von Savage [Sav76] kann diese Summe von einem Schaltkreis der Tiefe $O(\log(TN))$, Breite $O(T^2N)$ und Größe $O(T^2N)$ realisiert werden. Um gewährleisten zu können, dass die Ausgaben aller Schaltkreise $sc(p)$ zeitgleich vorliegen, werden diese Schaltkreise auf die gleiche Berechnungslänge erweitert.

Da die Laufzeit des Netzwerks auch bei gleicher Eingabelänge unterschiedlich viele Schritte $t \in O(T)$ benötigen kann, muss gewährleistet werden, dass der Schaltkreis unabhängig der konkreten Dauer die korrekte Ausgabe hat. Dazu wird der Ausgabeschaltkreis verwendet, der die Werte der zwei Schaltkreise $sc(p_1)$ und $sc(p_2)$, wobei p_1 und p_2 die Ausgabeprozessoren sind, $\wedge$-verkpünft. Dadurch wird der Ausgabewert der Datenleitung bestimmt und kann durch eine $\vee$-Verknüpfung mit null solange erhalten werden bis ein neuer auf gleiche Weise gewon-

nener Ausgabewert vorliegt, damit diese wiederum $\vee$-verknüpft werden können. Auf diese Weise wird der in einem Zeitpunkt berechnete Ausgabewert bis zur Wurzel des Schaltkreises weitergegeben, auch wenn dieser schon zu einem früheren Zeitpunkt vorlag.

Nun können alle Prozessorschaltkreise $sc(p)$, der Eingabeschaltkreis sowie der Ausgabeschaltkreis zu einem Schaltkreis zusammengefügt werden. Dies ist in Abbildung 4.6 dargestellt.

Die Schaltkreise $sc(p)$ der einzelnen Prozessoren sind parallel angeordnet, da ihre Berechnungen zeitgleich ausgeführt werden und jeweils eine Aktualisierung der Netzwerkprozessoren darstellen. In allen Ebenen befinden sich die gleichen Schaltkreise $sc(p)$, da die Prozessoren in jedem Schritt die gleiche Berechnung ausführen und dies durch $sc(p)$ repräsentiert wird. Jede Ebene erhält ihre Eingaben von der darüber liegenden, was den Ergebnissen des vorherigen Zeitschrittes beim Netzwerk entspricht. Zusätzlich sind die ersten $|\omega|$ Ebenen jeweils mit einem Bit der Eingabe verbunden, welches aus dem Eingabeschaltkreis hervorgeht. In den übrigen Schritten sind keine weiteren Eingabebits mehr vorhanden und die beiden Leitungen u_1 und u_2 können durch eine konstante Null dargestellt werden. Die Anzahl der Ebenen kann aus der Laufzeit des Netzwerks abgeleitet werden, da eine Ebene genau einen Zeitschritt repräsentiert.

Aus diesem Aufbau folgt, dass der zusammengesetzte Schaltkreis $c_{|\omega|}$ eine Tiefe von $O(T \log(T))$ hat, da es $O(T)$ Ebenen gibt, in denen sich die Schaltkreise $sc(p)$ mit Tiefe $O(\log(TN))$ befinden, wobei N eine Konstante darstellt. Die Breite von $c_{|\omega|}$ ergibt sich aus der Breite von $sc(p)$ und ist somit $O(T^2)$. Die Breite des Eingabeschaltkreises kann für die Betrachtung des Gesamtschaltkreises vernachlässigt werden, da diese von $|\omega|$ und nicht von T abhängt. Aus diesen beiden Werten lässt sich die Größe von $c_{|\omega|}$ ableiten und liegt daher in $O(T^3)$.

Für jedes Netzwerk gilt daher, dass über die Familie von Netzwerken eine Familie von Schaltkreisen existiert, in der ein Schaltkreis enthalten ist, der die geforderten Schranken einhält. Damit gilt die Aussage 4.2. □

4.4 Simulation am Beispiel der Palindromsprache

In diesem Abschnitt werden einige der zuvor erklärten Methoden am im vorherigen Kapitel verwendeten Beispiel der Sprache L der Palindrome dargestellt. Diese Sprache war wie folgt definiert:

$$L = \{\omega \mid \omega \text{ ist ein Palindrom}\}.$$

Das bedeutet, dass für ein Eingabewort ω entschieden werden soll, ob dieses symmetrisch ist.

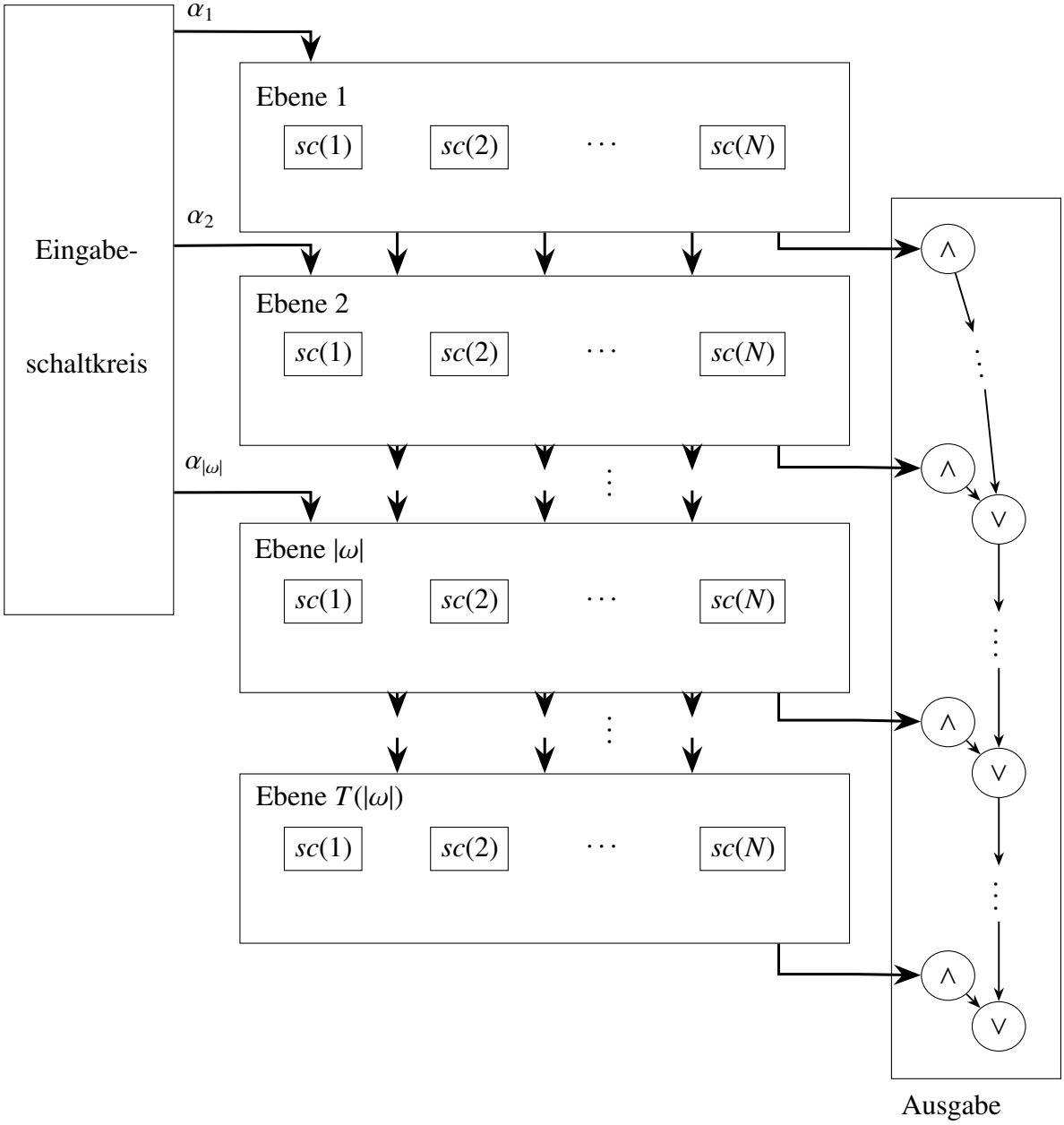

Abbildung 4.6: Aufbau des vollständigen Schaltkreises

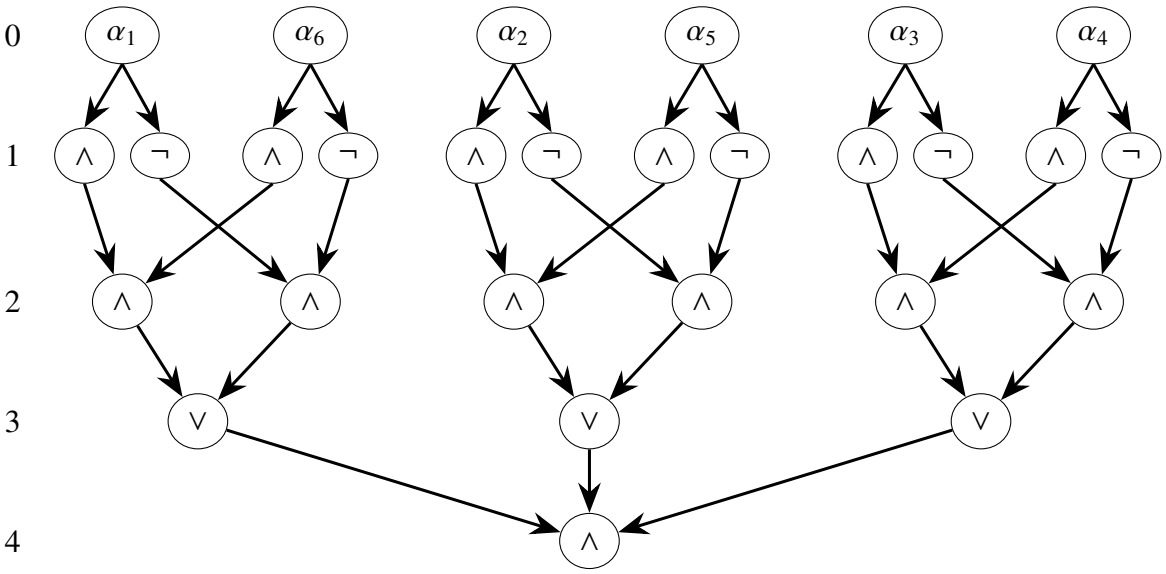

Abbildung 4.7: Aufbau des Schaltkreises c_6

4.4.1 Simulation von Schaltkreisen durch Prozessornetzwerke

Im Folgenden wird ein Schaltkreis c betrachtet, der die Sprache L akzeptiert. Da für die Simulation von c die unendliche Schaltkreisfamilie C verwendet wird, diese aber nicht explizit angegeben werden kann, ist in Abbildung 4.7 der Schaltkreis $c_6 \in C$ abgebildet, welcher das Beipsielwort $\omega = 101101_2$ verarbeiten kann. Die Eingabebits α des Wortes ω sind in der Abbildung derart angeordnet, dass die Pfleile möglichst wenig Überschneidungen aufweisen. Des Weiteren sind die Verbindungen der Konstanten eins zu den $\wedge$-Gatter nicht aufgeführt.

Wie zuvor angemerkt wird die irrationale Zahl $\hat{C}$ benötigt. Da im weiteren Vorgehen nur die Teile von $\hat{C}$ relevant sind, welche Schaltkreisen der Größe sechs und kleiner entsprechen und von diesen nur die Codierung von c_6 explizit vorliegen muss, werden en$[c_6]$ sowie die Struktur von $\hat{C}$ angegeben.

$$
\begin{aligned}
\mathrm{en}[c_6] = 60\,&\underbrace{424222224}_{x_1\wedge 1}\,0\,\underbrace{224222222}_{\neg x_1}\,0\,\underbrace{422422224}_{x_2\wedge 1}\,0\,\underbrace{222422222}_{\neg x_2}\,0\,\underbrace{422242224}_{x_3\wedge 1}\,0\,\underbrace{222242222}_{\neg x_3}\\
0\,&\underbrace{422224224}_{x_4\wedge 1}\,0\,\underbrace{222224222}_{\neg x_4}\,0\,\underbrace{422222424}_{x_5\wedge 1}\,0\,\underbrace{222222422}_{\neg x_5}\,0\,\underbrace{422222244}_{x_6\wedge 1}\,0\,\underbrace{222222242}_{\neg x_6}\\
60\,&\underbrace{42422222222242}_{x_1\wedge x_6}\,0\,\underbrace{42242222222224}_{\neg x_1\wedge\neg x_6}\,0\,\underbrace{42224222224222}_{x_2\wedge x_5}\\
0\,&\underbrace{42222422222422}_{\neg x_2\wedge\neg x_5}\,0\,\underbrace{42222242422222}_{x_3\wedge x_4}\,0\,\underbrace{42222224242222}_{\neg x_3\wedge\neg x_4}\\
60\,&\underbrace{44442222}_{x_1=x_6}\,0\,\underbrace{44224422}_{x_2=x_5}\,0\,\underbrace{44222244}_{x_3=x_4}\,60\,\underbrace{42444}_{\text{Ausgabe}}
\end{aligned}
$$

	init	1	2	3	4	5	6	7	8	9
u_1	0	1	0	1	1	0	1	0	0	0
u_2	0	1	1	1	1	1	1	0	0	0
y_1	0	0	0.4_9	0.24_9	0.424_9	0.4424_9	0.24424_9	0.424424_9	0	0
y_2	0	0	$1-2^{-1}$	$1-2^{-2}$	$1-2^{-3}$	$1-2^{-4}$	$1-2^{-5}$	$1-2^{-6}$	0	0
y_3	0	0	1	1	1	1	1	1	0	0
y_4	0	0	0	0	0	0	0	0	1	0
y_5	0	0	0	0	0	0	0	0	0.424424_9	0
y_6	0	0	0	0	0	0	0	0	$1-2^{-6}$	0

Tabelle 4.8: Bestimmung von en[ω]

Die siebte Ziffer in der Beschreibung der Verbindungen von Ebene eins entsteht durch die Verbindung mit der Konstanten eins.

Die Struktur von $\hat{C}$ hat die Form

$$\hat{C} = 8\overline{\text{en}}[c_1]8\overline{\text{en}}[c_2]8\overline{\text{en}}[c_3]8\overline{\text{en}}[c_4]8\overline{\text{en}}[c_5]8\overline{\text{en}}[c_6]\ldots$$

Um die Simulation auszuführen wird zusätzlich die Codierung der Eingabe benötigt. Für das Beispielwort $\omega = 101101_2$ ist diese en[ω] = 424424_9, woraus die Zahl $\widehat{\text{en}}[\omega] = 0.424424_9$ resultiert. In der Tabelle 4.8 ist vollständig dargestellt wie das angegebene Netzwerk $\mathcal{N}_I$ aus dem Wort ω die Codierung en[ω] berechnet. Dabei ist u_1 die Datenleitung und u_2 die Validierungsleitung.

Im nächsten Schritt wird mit dem Netzwerk $\mathcal{N}_R$ die Codierung $\widehat{\text{en}}[c_6]$ aus $\hat{C}$ extrahiert. Dabei gilt für die Dateneleitung $u(1) = 111111\ldots_2 = 1-2^{-6}$ und $u(t) = 0$ für alle Zeitpunkte $t > 1$. Die wichtigsten Schritte dieses Verfahrens werden in Tabelle 4.9 und 4.10 aufgezeigt.

In Tabelle 4.9 ist dargestellt wie der retrieval Algorithmus initialisiert wird. Dabei kann beobachtet werden wie sich der Wert des Zählers x_{13} verringert, wenn eine Acht in der Codierung gelesen wird. Dieses Vorgehen wird im weiteren Verlauf noch fünf Mal wiederholt bis der Wert von x_{13} auf Null gesetzt wurde. Die in der Tabelle als x gekennzeichneten Ziffern repräsentieren dabei die Codierung von en[c_1], weshalb die Aktivierung der Prozessoren x_0 bis x_9 in Schritt 6 unbekannt ist.

Ab diesem Zeitpunkt wird die Codierung $\widehat{\text{en}}[c_6]$ ausgelesen und der Wert des Prozessors x_{11} aktualisiert. Dies ist in der Tabelle 4.10 abgebildet, wobei t den Zeitpunkt beschreibt, in dem die sechste Acht gelesen wurde.

Die in der Tabelle 4.10 gezeigten Berechnungen werden solange weitergeführt bis die vollständige Codierung in x_{12} steht. Beim Lesen der Acht, welche die Codierungen en[c_6] und en[c_7] voneinander trennt, erhält der Prozessor x_7 das erste Mal seit Zeitpunkt t wieder einen Wert und die Codierung wird nach x_{16} übertragen.

Auf diese Weise liegen beide für die Ausführung von $\mathcal{N}_S$ notwendigen Informationen en[ω]

	init	1	2	3	4	5	6
x_0	0	0	0	0	1	0	-
x_1	0	0	0	0	1	0	-
x_2	0	0	0	0	1	0	-
x_3	0	0	0	0	1	0	-
x_4	0	0	0	0	1	0	-
x_5	0	0	0	0	1	0	-
x_6	0	0	0	0	1	0	-
x_7	0	0	0	0	1	0	-
x_8	0	0	0	0	$0.xxx$	0	-
x_9	0	0	1	0	0	0	0
x_{10}	0	0	0	$0.8xx$	0	$0.xxx$	0
x_{13}	0	0	$1-2^{-6}$	0	$1-2^{-6}$	0	$1-2^{-5}$
x_{14}	0	0	0	0	0	$1-2^{-5}$	0
x_{15}	0	0	0	$1-2^{-6}$	0	0	0

Tabelle 4.9: Initialisierung von $\mathcal{N}_R$

	$t+1$	$t+2$	$t+3$	$t+4$	$t+5$	$t+6$	$t+7$	$t+8$	$t+9$
x_0	0	1	0	1	0	1	0	1	0
x_1	0	1	0	1	0	1	0	1	0
x_2	0	1	0	1	0	1	0	0.406	0
x_3	0	1	0	1	0	1	0	0	0
x_4	0	0.442	0	0.424	0	0.240	0	0	0
x_5	0	0	0	0	0	0	0	0	0
x_6	0	0	0	0	0	0	0	0	0
x_{10}	0.444	0	0.442	0	0.424	0	0.240	0	0.406
x_{11}	0	0	0.4	0	0.44	0	0.444	0	0.2444
x_{12}	0	0	0	0.4	0	0.44	0	0.444	0
x_{13}	0	$1-2^0$	0	0	0	0	0	0	0
x_{14}	$1-2^0$	0	0	0	0	0	0	0	0

Tabelle 4.10: Bestimmung von $\widehat{\text{en}}[c_6]$

sowie $\widehat{\text{en}}[c_6]$ vor. Da das Netzwerk $\mathcal{N}_S$ zuvor über eine Turingmaschine eingeführt wurde, aus der mit dem Vorgehen des vorherigen Kapitels das Netzwerk konstruiert wird, sind im Folgenden die Zustände dieser Turingmaschine aufgeführt:

s_0 : Startzustand,

s_1 : Zustand zur Vorbereitung eines neuen Gatter,

s_2 : Zustand zum Auslesen der Gateoperation,

s_3 : Zustand zur Berechnung eines $\wedge$ -Gatter,

s_4 : Zustand zur Berechnung eines $\vee$ -Gatter,

s_5 : Zustand zur Berechnung eines $\neg$-Gatter,

s_6 : Zustand zum Löschen der Eingabe,

s_7 : Zustand zum Übertragen der Ausgabe,

s_8 : Endzustand.

4.4.2 Simulation von Prozessornetzwerken durch Schaltkreise

In diesem Abschnitt wird am Beispiel der Sprache L beschrieben, wie aus der Familie der Netzwerke $\mathcal{N}_1(n)$ die Schaltkreisfamilie C konstruiert werden kann. Dabei wird der Aufbau anhand des Schaltkreises c_6 dargestellt.

Die Sprache der Palindrome kann von einem Prozessornetzwerk berechnet werden, welches zunächst das Eingabewort ω liest und codiert, anschließend diese Codierung bitweise umdreht, sodass zwei Codierungen des Wortes ω vorliegen und zum Schluss die beiden Codierungen bitweise vergleicht. Das Vorgehen der Codierung eines beliebigen Wortes ω mit $|\omega| = 6$ wird in Tabelle 4.11 dargestellt, wobei die in Abschnitt 4.3.1 eingeführte Codierung mit $\text{en}[\omega] = 0.abcdef$ verwendet wird.

Der nächste Schritt, das bitweise Umkehren von $\text{en}[\omega]$, um $\overline{\text{en}}\,[\omega]$ zu erhalten, kann mithilfe der beiden Funktion $\widetilde{\Xi}$ und $\tilde{\Lambda}$ aus Abschnitt 4.3.1 realisiert werden. Die zugrundeliegenden Berechnungen sind vereinfacht in Tabelle 4.12 dargestellt. Der Prozessor y_2 dient dabei der Speicherung von $\text{en}[\omega]$, wohingegen in y_4 der Wert $\overline{\text{en}}\,[\omega]$ entsteht.

Das Vergleichen der beiden Zahlen erfolgt wieder mit den Funktion $\widetilde{\Xi}$ und $\tilde{\Lambda}$, wobei in jedem Schritt das vorderste Bit von $\text{en}[\omega]$ sowie $\overline{\text{en}}\,[\omega]$ extrahiert und verglichen wird. Dies wird in Tabelle 4.13 gezeigt. Dabei hat der Prozessor y_8 solange den Wert eins bis eine von y_7 bestimmte Differenz ungleich null ist. Das führt dazu, dass am Ende nur eine Eins in y_8 vorliegt, wenn alle Differenzen null waren und das Wort damit ein Palindrom ist.

	init	1	2	3	4	5	6	7
u_1	0	1	0	1	1	0	1	0
u_2	0	1	1	1	1	1	1	0
y_1	0	0	$0.a$	$0.ab$	$0.abc$	$0.abcd$	$0.abcde$	$0.abcdef$

Tabelle 4.11: Codierung von ω

y_2	$0.abcdef$	$0.abcdef$	$0.abcdef$	$0.abcdef$	$0.abcdef$	$0.abcdef$	$0.abcdef$
y_3	$0.abcdef$	$0.bcdef$	$0.cdef$	$0.def$	$0.ef$	$0.f$	0
y_4	0	$0.a$	$0.ba$	$0.cba$	$0.dcba$	$0.edcba$	$0.fedcba$

Tabelle 4.12: Umkehren der Codierung en[ω]

Im vorherigen Abschnitt wurde gezeigt, dass zu jedem Netzwerk eine Familie von Netzwerken existiert, sodass aus dieser eine Schaltkreisfamilie C konstruiert werden kann. Ausgehend davon wird im Folgenden der Schaltkreis c_6 der Schaltkreisfamilie C beschrieben.

Da c_6 mit Eingabewörtern der Länge sechs arbeitet, hat der Eingabeschaltkreis sechs Variablen, die gespeichert und an die Prozessorschaltkreise weitergegeben werden müssen. Für das Beispielwort $\omega = 101101$ gilt daher $x_1 = 1, x_2 = 0, x_3 = 1, x_4 = 1, x_5 = 0$ und $x_6 = 1$. Jedes x_i wird an die Ebene i mit $i \in \{1, \ldots .6\}$ weitergegeben, indem diese eine Verbindung zu jedem $sc(p)$ der Ebene aufweist. Zusätzlich wird mit der Validierungsleitung, die in den ersten sechs Ebenen durch die Konstante eins repräsentiert werden kann, genauso verfahren.

Die Prozessorschaltkreise sind abhängig von der konkreten Berechnung des Prozessors, weshalb zunächst die Berechnung von y_1 im Prozessornetzwerk angegeben wird.

$$y_1^+ = \sigma\left(\frac{1}{9}y_1 + \frac{2}{9}u_1 + \frac{2}{9} + u_2 - 1\right).$$

Für die erste Ebene besitzt $sc(y_1)$ daher Verbindungen durch eingehende Kanten zu $x_1 = u_1(1)$, der Konstanten eins, die u_2 repräsentiert, und der Konstanten null, die für den Initialisierungswert von y_1 steht. Der Schaltkreis $sc(y_1)$ berechnet eine Summe, die von diesen Eingaben sowie den festen Gewichten $\frac{1}{9}, \frac{2}{9}$ und $\frac{-7}{9}$ abhängt. In den weiteren Ebenen wird der gleiche Schaltkreis $sc(y_1)$ für die Berechnung verwendet und weist daher Verbindungen auf, die analog zu dem in der ersten Ebene sind. Die Ausgabe von jedem $sc(y_1)$ entspricht dann dem in Tabelle 4.11 im jeweiligen Zeitschritt aufgeführten Wert. Ab der siebten Ebene gibt es

y_5	$0.abcdef$	$0.bcdef$	$0.cdef$	$0.def$	$0.ef$	$0.f$	0	0
y_6	$0.fedcba$	$0.edcba$	$0.dcba$	$0.cba$	$0.ba$	$0.a$	0	0
y_7	0	$a - f$	$b - e$	$c - d$	$d - c$	$e - b$	$f - a$	0
y_8	1	1	-	-	-	-	-	$\text{en}[\omega] = \overline{\text{en}}\,[\omega]$?

Tabelle 4.13: Vergleich von en[ω] und $\overline{\text{en}}\,[\omega]$

keine weiteren Eingabebits, weshalb die Daten- und Validierungsleitungen u_1 und u_2 mit einer konstanten Null repräsentiert werden können. Dieses Vorgehen gilt analog für alle übrigen Prozessoren des Prozessornetzwerks.

Zuletzt wird der Ausgabeschaltkreis betrachtet, der für das vorgestellte Prozessornetzwerk erst in der letzten Ebene einen Wert erhält, da alle Eingaben der Länge sechs in gleicher Zeit verarbeitet werden. Das liegt daran, dass das Verfahren nicht vorzeitig abgebrochen wird, wenn eines der vorderen Symbolpärchen nicht den gleichen Wert hat. Daraus folgt, dass der Wert des Ausgabeprozessors in jeder außer der letzten Ebene null ist und dieser Wert an den Ausgabeschaltkreis weitergegeben wird.

5 Zusammenfassung und Ausblick

In dieser Arbeit wurde das Modell des neuronalen Netzes im Spezialfall des Prozessornetzwerks den beiden Modellen der Turingmaschine und des Booleschen Schaltkreises gegenübergestellt. Dabei wurde untersucht, wie sich das Prozessornetzwerk in Hinsicht auf die Mächtigkeit erkennbarer Sprachen und berechenbarer Probleme im Vergleich zu jeweils einem der beiden anderen Modelle verhält. In Kapitel 3 wurde festgestellt, dass alle von einer Turingmaschine berechenbaren Funktionen auch von einem Prozessornetzwerk berechnet werden können. In Kapitel 4 hingegen wurde für die beiden Modelle des Schaltkreises und des Prozessornetzwerks gezeigt, dass jede von dem einen Modell akzeptierte Sprache auch von dem anderen Modell akzeptiert wird und diese Berechnung mit einem polynomiellen Overhead der Laufzeit des Netzwerks bzw. Größe des Schaltkreises ausgeführt werden kann.

Das in dieser Arbeit verwendete Prozessornetzwerk weißt durch die Wahl der Sigmoidfunktion sowie die Beschränkung auf Linearkombinationen bei der Berechnung neuer Prozessorwerte einige Einschränkungen auf, die weitere Untersuchungen erlauben.

Eine Berechnung neuer Prozessorwerte, die Multiplikationen zweier Prozessorwerte zulässt, würde in einer Version des Netzwerks resultieren, die unter anderem Potenzen von Zahlen in wenigen Schritten berechnen kann. Des Weiteren wäre die Nutzung von veränderlichen Skalierungsfaktoren in Form eines Prozessorwertes für die Berechnung einer Aktivierung möglich. Zusätzlich könnte der Einfluss einer anderen Aktiverungsfunktion auf die Mächtigkeit des Modells betrachtet werden.

Einige weitere Themen, die in den in dieser Arbeit behandelten Papern, beschrieben bzw. erwähnt werden, könnten weiter ausgeführt werden. Im Paper „On the Computational Power of Neural Nets“ von Siegelmann und Sontag [SS95] wird das in Abschnitt 3.3 eingeführte Verfahren derart angepasst, dass eine Echtzeitsimulation der Turingmaschine möglich ist. Des Weiteren wird darauf eingegangen wie das Prozessornetzwerk um Nichtdeterminusmus erweitert werden kann. Im Paper „Analog Computation via Neural Nets“ [SS94] hingegen vergleichen Siegelmann und Sontag das hier betrachtete Prozessornetzwerk mit dem von Blum, Shub und Smale eingeführten Modell. Diese Parallelen könnten näher betrachtet werden.

Ein Vergleich von neuronalen Netzen mit anderen in der Informatik bewährten Modellen könnte neue Ergebnisse hervorbringen.

Literatur

[App21] Jürgen Appell. *Analysis in Beispielen und Gegenbeispielen. Eine Einführung in die Theorie reeller Funktionen*. Springer Spektrum, 2021. ISBN: 978-3-662-63433-2. DOI: https://doi.org/10.1007/978-3-662-63433-2. URL: https://link.springer.com/book/10.1007/978-3-662-63433-2.

[Den04] Manfred Denker. *Einführung in die Analysis dynamische Systeme*. Springer Berlin, Heidelberg, 2004. ISBN: 3-540-20713-9. DOI: 10.1007/b137966. URL: https://doi.org/10.1007/b137966.

[Sav76] Jown E. Savage. *The Complexity of Computing*. John Wiley & Sons, 1976. ISBN: 0-471-75517-6.

[SS94] Hava T. Siegelmann und Eduardo D. Sontag. „Analog computation via neural networks“. In: *Theoretical Computer Science* 131.2 (1994), S. 331–360. ISSN: 0304-3975. DOI: https://doi.org/10.1016/0304-3975(94)90178-3. URL: https://www.sciencedirect.com/science/article/pii/0304397594901783.

[SS95] H.T. Siegelmann und E.D. Sontag. „On the Computational Power of Neural Nets“. In: *Journal of Computer and System Sciences* 50.1 (1995), S. 132–150. ISSN: 0022-0000. DOI: https://doi.org/10.1006/jcss.1995.1013. URL: https://www.sciencedirect.com/science/article/pii/S0022000085710136.

[Vol99] Heribert Vollmer. *Introduction to Circuit Complexity - A Uniform Approach*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 1999. ISBN: 978-3-540-64310-4. DOI: 10.1007/978-3-662-03927-4. URL: https://doi.org/10.1007/978-3-662-03927-4.