



# **FOM Hochschule für Oekonomie & Management**

Hochschulzentrum Hamburg

## **Bachelor Thesis**

im Studiengang Informatik

zur Erlangung des Grades eines

**Bachelor of Science (B.Sc.)**

über das Thema

**Implementierung des Equilibrium Propagation Algorithmus auf analogen  
Computern für das Trainieren neuronaler Netze**

von

**Merlin Moelter**

Betreuer : Dipl.-Phys.Ing. Stefan Scharr

Matrikelnummer : 575141

Abgabedatum : 1. März 2025

## Inhaltsverzeichnis

<b>Abbildungsverzeichnis</b>	<b>IV</b>
<b>Abkürzungsverzeichnis</b>	<b>VI</b>
<b>1 Einleitung</b>	<b>1</b>
1.1 Relevanz des Themas . . . . .	1
1.2 Fragestellung . . . . .	1
1.3 Zielsetzung der Arbeit . . . . .	1
<b>2 Grundlagen</b>	<b>3</b>
2.1 Neuronale Netze . . . . .	3
2.1.1 Aufbau und Arten neuronaler Netze . . . . .	3
2.1.2 Training neuronaler Netze mit Backpropagation und Gradient Descent	5
2.2 Analoge Computer . . . . .	6
2.2.1 Geschichte zu analogen Computern . . . . .	6
2.2.2 Typische Komponenten und Bauweisen analoger Computer . . . . .	7
2.2.3 Aufbau von Schaltkreisen zur Lösung von Differentialgleichungen .	11
2.3 Energiebasierte Modelle . . . . .	13
2.3.1 Definition: energiebasierte Modelle und energiebasiertes Lernen .	13
2.3.2 Das Hopfield-Netzwerk: Eine Herangehensweise an neuronale Netze	13
2.3.3 Energiebasiertes Lernen am Beispiel Equilibrium Propagation .	15
2.3.3.1 Mathematische Grundlagen . . . . .	15
2.3.3.2 Theoretische Anwendung am Beispiel eines Hopfield-Netzwerks . . . . .	17
<b>3 Methodik</b>	<b>20</b>
3.1 Forschungsansatz: Konstruktion . . . . .	20
3.2 Auswahl und Grundlagen der Simulationssoftware . . . . .	20
3.3 Auswahl der Referenzarbeit für das Netzwerkdesign . . . . .	22
3.4 Erwartete Ergebnisse . . . . .	23
<b>4 Konstruktion</b>	<b>25</b>
4.1 Simulationsumgebung . . . . .	25
4.1.1 Übernahme des Hopfield-Netzwerks . . . . .	25
4.1.2 Notwendige Modifikationen am Netzwerk für Equilibrium Propagation	26
4.1.3 Simulation des angepassten Netzwerks . . . . .	26

4.2 Konstruktion des Equilibrium Propagation Algorithmus . . . . .	28
4.2.1 Umsetzung der theoretischen Modelle in der Simulationssoftware . .	28
4.2.2 Herausforderungen bei der Übernahme und Anpassung . . . . .	31
4.2.3 Strategien zur Fehlerbehebung und Optimierung . . . . .	33
4.2.4 Validierung des Algorithmus durch Testläufe . . . . .	35
<b>5 Diskussion</b>	<b>38</b>
5.1 Reflexion der Konstruktionsmethodik . . . . .	38
5.1.1 Bewertung der Simulationsumgebung und -werkzeuge . . . . .	38
5.1.2 Effektivität der Modifikationen am Netzwerk . . . . .	39
5.2 Diskussion der Herausforderungen . . . . .	39
5.2.1 Technische Herausforderungen und Lösungsansätze . . . . .	39
5.2.2 Umgang mit unerwarteten Ergebnissen . . . . .	40
5.3 Potenziale des Equilibrium Propagation in analogen Systemen . . . . .	40
5.4 Kritische Bewertung der Forschungsergebnisse . . . . .	41
5.4.1 Zusammenfassung der Erkenntnisse . . . . .	41
5.4.2 Vergleich mit den Erwartungen und Zielsetzung . . . . .	41
<b>6 Zusammenfassung</b>	<b>43</b>
6.1 Beantwortung der Forschungsfrage . . . . .	43
6.2 Bewertung der methodischen Vorgehensweise . . . . .	43
6.3 Herausforderungen und Lösungsansätze . . . . .	44
6.4 Empfehlungen für zukünftige Forschung . . . . .	45
<b>Anhang</b>	<b>46</b>
<b>Literaturverzeichnis</b>	<b>52</b>

## Abbildungsverzeichnis

Abbildung 1:	Operationsverstärker mit Rückkopplung . . . . .	7
Abbildung 2:	Symbole zur Darstellung eines analogen Computers . . . . .	9
Abbildung 3:	The Analog Thing des Herstellers anabrid GmbH . . . . .	10
Abbildung 4:	Simulation der Euler-Spirale auf einem The Analog Thing (THAT) . . . . .	10
Abbildung 5:	Masse-Feder-Dämpfer System ohne Rückkopplung . . . . .	12
Abbildung 6:	Vollständiges Masse-Feder-Dämpfer System . . . . .	13
Abbildung 7:	Simulation eines Masse-Feder-Dämpfer Systems in Simulink . . . . .	22
Abbildung 8:	Masse-Feder-Dämpfer System in Simulink und Simscape . . . . .	23
Abbildung 9:	Erste Simulation des Hopfield-Netzwerk (HNN) . . . . .	27
Abbildung 10:	Zweite Simulation des HNN . . . . .	27
Abbildung 11:	Dritte Simulation des HNN . . . . .	28
Abbildung 12:	Ausgabe des zweiphasigen Lernprozesses des Equilibrium Propagation (EP) im ersten vorgestellten Ansatz . . . . .	29
Abbildung 13:	Graph der Aktivierungsfunktion $\rho(x) = \frac{1}{1+e^{-x}}$ (rot) und ihrer Ableitung (grün) . . . . .	32
Abbildung 14:	Auswirkungen einer nicht-negativen Aktivierungsfunktion auf das Continual Equilibrium Propagation (C-EP) . . . . .	33
Abbildung 15:	Die Parameter des Modells gelangen zu keinem Fixpunkt und verfehlten damit das Minimum der Kostenfunktion . . . . .	33
Abbildung 16:	Eine Annäherung an C-EP findet passende Parameter für das Netzwerk	34
Abbildung 17:	Graph der Aktivierungsfunktion (rot) und der skalierten Variante $\rho(x) = \frac{1}{1+e^{-4x}}$ (orange) . . . . .	34
Abbildung 18:	Simulationen des C-EP mit $\beta = 1, \eta = 0.5$ . Dargestellt ist die Ausgabe des Netzwerks . . . . .	35
Abbildung 19:	Annäherungen des C-EP an den Grenzwert 0 . . . . .	36
Abbildung 20:	Simulationen des C-EP mit der Aktivierungsfunktion: $\rho(x) = \tanh(2x)$	36
Abbildung 21:	Simulation des C-EP mit der Aktivierungsfunktion „ReLU“ ( $R(x) = \max(x, 0)$ ). Gezeigt wird die Dynamik der Zustände des Netzwerks $\frac{ds}{dt}$ . . . . .	37
Abbildung 22:	Simulationen des C-EP mit 6 Neuronen . . . . .	37
Abbildung 23:	Umsetzung eines HNNs mit zwei Neuronen in Simulink . . . . .	46
Abbildung 24:	Umsetzung eines HNNs mit drei Neuronen in Simulink . . . . .	47
Abbildung 25:	Finales Modell des HNN in Simulink mit einem Funktionsblock zur Validierung der Funktionsweise . . . . .	48
Abbildung 26:	Umsetzung eines diskreten Ansatzes zur Lösung des EP in Simulink	50

Abbildung 27: Umsetzung einer Annäherung an C-EP in Simulink . . . . .	51
Abbildung 28: Größere Simulationen des C-EP können mithilfe der Matlab-Funktionsblöcke in Simulink durchgeführt werden, um Aufwand beim Modellieren zu sparen. . . . .	51

## Abkürzungsverzeichnis

<b>EBM</b>	Energiebasiertes Modell
<b>THAT</b>	The Analog Thing
<b>BPTT</b>	Backpropagation Through Time
<b>TLU</b>	Threshold Logic Unit
<b>MLP</b>	Multilayer-Perceptron
<b>FNN</b>	Feedforward Neural Network
<b>DNN</b>	Deep Neuronal Network
<b>CNN</b>	Convolutional Neural Network
<b>RNN</b>	Recurrent Neural Network
<b>HNN</b>	Hopfield-Netzwerk
<b>EP</b>	Equilibrium Propagation
<b>C-EP</b>	Continual Equilibrium Propagation
<b>SNN</b>	Spiking neural network
<b>EqSpike</b>	Spike-driven Equilibrium Propagation
<b>FPAA</b>	Field Programmable Analog Array

## 1 Einleitung

### 1.1 Relevanz des Themas

### 1.2 Fragestellung

Die vorliegende Arbeit untersucht die Implementierung des Equilibrium Propagation Algorithmus auf analogen Computern und die damit verbundenen Herausforderungen und Potenziale. Dabei stellt sich die Forschungsfrage:

Wie kann der Equilibrium Propagation Algorithmus effektiv auf analogen Computern implementiert werden und welche spezifischen Herausforderungen ergeben sich aus dieser Umsetzung?

Der Fokus liegt auf der praktischen Umsetzung des Algorithmus in einer Simulationsumgebung und der notwendigen Anpassung des HNN für die Anwendung von EP. Dabei werden insbesondere die Herausforderungen bei der Übernahme theoretischer Modelle in analoge Rechenstrukturen betrachtet. Zudem wird analysiert, inwiefern analoge Systeme durch ihre kontinuierliche Signalverarbeitung eine effiziente Alternative zu digitalen Implementierungen bieten können.

### 1.3 Zielsetzung der Arbeit

Ziel dieser Arbeit ist die Implementierung des EP Algorithmus auf einem analogen Computer und die Untersuchung der damit verbundenen Herausforderungen und Potenziale. Dabei wird analysiert, inwiefern sich der Algorithmus an analoge Rechenstrukturen anpassen lässt und welche Vorteile sich daraus für das Training neuronaler Netze ergeben.

Ein zentraler Aspekt ist die Modifikation eines HNN zur Nutzung von EP. Dazu wird das Netzwerk in einer Simulationsumgebung abgebildet und entsprechend angepasst. Im Fokus stehen dabei die mathematischen und technischen Anforderungen für eine stabile Implementierung sowie mögliche Einschränkungen durch die Eigenschaften analoger Systeme.

Besondere Herausforderungen ergeben sich aus der kontinuierlichen Signalverarbeitung analoger Rechner, wodurch klassische Optimierungsmethoden, wie sie in digitalen Systemen genutzt werden, nicht direkt übertragbar sind. Daher wird untersucht, welche strukturellen Anpassungen erforderlich sind und wie sich analoge Komponenten für die Implementierung des Lernverfahrens nutzen lassen.

Die Arbeit soll damit einen Beitrag zur Evaluierung analoger Rechenarchitekturen im Kontext maschinellen Lernens leisten und aufzeigen, inwiefern EP auf nicht-digitalen Systemen realisierbar ist.

## 2 Grundlagen

In diesem Kapitel werden die theoretischen und technischen Grundlagen dieser Arbeit erläutert. Zunächst wird ein Überblick über neuronale Netze gegeben, wobei ihr Aufbau, verschiedene Architekturen sowie das Training mittels Backpropagation und Gradient Descent beschrieben werden. Anschließend folgt eine Einführung in analoge Computer, ihre historische Entwicklung sowie ihre typischen Komponenten und Bauweisen. Schließlich werden energiebasierte Modelle behandelt, mit einem besonderen Fokus auf das Hopfield-Netzwerk und das Equilibrium Propagation, deren mathematische Grundlagen und theoretische Anwendung für diese Arbeit von zentraler Bedeutung sind.

### 2.1 Neuronale Netze

#### 2.1.1 Aufbau und Arten neuronaler Netze

„Birds inspired us to fly, burdock plants inspired Velcro, and nature has inspired countless more inventions. It seems only logical then, to look at the brain's architecture for inspiration on how to build an intelligent machine“ (Géron, A., 2019, S. 279)

Die Wissenschaftler Warren S. McCulloch und Walter Pitts führten neuronale Netze erstmals 1943 in ihrer gemeinsamen Arbeit „A logical calculus of the ideas immanent in nervous activity“ (McCulloch, W. S., Pitts, W., 1943) ein. Sie entwickelten ein vereinfachtes Modell eines künstlichen Neurons, das ausschließlich binäre Eingaben und eine binäre Ausgabe besitzt. Die Aktivierung der Ausgabe erfolgt, sobald eine festgelegte Anzahl an Eingabewerten aktiviert ist. McCulloch und Pitts konnten zeigen, dass bereits dieses einfache Modell ausreicht, um beliebige logische Ausdrücke in Form eines neuronalen Netzes darzustellen.

Heute sind neuronale Netze für ihre Vielseitigkeit, Leistungsfähigkeit und Skalierbarkeit bekannt, was wesentlich zur Entstehung des Forschungsfeldes „Deep Learning“ beigetragen hat. (Géron, A., 2019) Die mit Deep Learning verbundene Aufmerksamkeit führte zu innovativen Entwicklungen wie dem Sprachmodell „GPT-4“ von OpenAI oder der Bildgenerierungs-KI „Midjourney“. Diese Fortschritte zeigen, dass neuronale Netze in der Lage sind komplexe Probleme zu lösen und in manchen Fällen sogar Ergebnisse zu liefern, die mit denen eines Menschen vergleichbar sind. (OpenAI et al., 2024)

Eines der einfachsten neuronalen Netze ist das von Rosenblatt, F., 1958 vorgestellte Perceptron, dieses basiert auf dem Konzept der Threshold Logic Unit (TLU). Die Eingabe-

und Ausgabewerte einer TLU sind numerisch und den eingehenden Verbindungen ist jeweils ein Gewicht zugewiesen. Die TLU berechnet nun die gewichtete Summe der Eingabewerte  $z = w_1x_1 + w_2x_2 \dots + w_nx_n = xtw$ . Unter Anwendung einer Aktivierungsfunktion  $hw(x) = step(z)$  kann nun berechnet werden, ob die TLU den Wert 0 oder 1 ausgibt. Eine Aktivierungsfunktion für diese Art der Neuronen ist i. d. R. eine Heaviside-Funktion oder eine Vorzeichen-Funktion. (Géron, A., 2019, vgl. S. 284 ff.)

Aktivierungsfunktionen aus Buch einfügen

Abbildung aus Buch nachstellen

Eine alleinstehende TLU kann ausschließlich für lineare binäre Klassifikation genutzt werden, es berechnet eine gewichtete Summe anhand der Eingabewerte und gibt einen positiven oder negativen Ausgabewert, abhängig von der Überschreitung eines Schwellenwertes. Ein Perceptron stellt nun eine Sammlung dieser Einheiten auf einer einzelnen Ebene dar, wobei jede TLU mit jeder Eingabe verbunden ist. Dies wird als Dense Layer oder Fully Connected Layer bezeichnet. Die Eingabewerte des Perceptron werden durch Eingabe-Neuronen geschleust, wozu zusätzlich ein Bias-Neuron gezählt wird. Dieses Neuron gibt konstant den Wert 1 aus und dient dazu, jedem Neuron des Perceptron einen Bias-Wert zuzuweisen. Die Ausgabe eines Perceptron berechnet sich durch  $h_W, b(X) = \rho(XW + b)$  (ebd., vgl. S. 284 ff.)

Das Perceptron kann nun in mehreren Ebenen genutzt werden, um ein Multilayer-Perceptron (MLP) zu erzeugen. Dieses besteht aus einer Eingabe-Ebene, mindestens einer versteckten Ebene und einer Ausgabe-Ebene. Jede dieser Ebenen ist im MLP mit der jeweils nächsten Ebene vollständig verbunden. Das MLP kann durch die vorwärts gerichteten Verbindungen auch als Feedforward Neural Network (FNN) bezeichnet werden, eines mit vielen versteckten Ebenen wird Deep Neuronal Network (DNN) genannt. (ebd., vgl. S. 284 ff.)

Das bisher beschriebene MLP kann zur Klassifikation genutzt werden. Um dieses auch auf Regressionen anwenden zu können, muss die Aktivierungsfunktion entweder entfernt oder durch z.B. ReLU ausgetauscht werden, damit die Ausgabeneuronen willkürliche Werte annehmen können. Die Anzahl der Ausgabe-Neuronen muss in dem Fall angepasst werden, sodass jeder geforderte Ausgabewert durch ein Neuron abgebildet ist. (ebd., vgl. S. 292 ff.)

Eine weitere Art des neuronalen Netz ist das Convolutional Neural Network (CNN), dessen Hauptbestandteil die Convolutional-Ebenen sind. Diese Ebenen sind nur jeweils mit einem Ausschnitt der vorherigen Ebene verbunden, wodurch das daraus entstehende neuronale Netz abstrakte Eigenschaften der Eingabe-Neuronen erlernen kann. Da die einzelnen

Ebenen hier nicht, wie beim MLP, vollständig verbunden sind, erlaubt ein CNN eine große Anzahl an Neuronen pro Ebene, ohne den Rechenaufwand für Training und Inferenz exponentiell zu steigern. Durch diese Eigenschaften eignet sich das CNN besonders für die Bildbearbeitung, da hier als Eingabe die Pixel genutzt und darin Eigenschaften des Bildes gefunden werden können. (Géron, A., 2019, vgl. S. 447 f.)

Das Recurrent Neural Network (RNN) ähnelt vom Aufbau dem FNN unterscheidet sich aber durch rückwärts gerichtete Verbindungen. Die Ausgabe eines Neuron wird damit Teil seiner Eingabe, womit es sich Informationen „merken“ kann. Ein RNN kann als Sequenz-zu-Sequenz Netzwerk genutzt werden, es erhält also eine Sequenz an Eingabe-Werten und produziert eine Sequenz an Ausgabe-Werten. Genauso kann jeder Ausgabewert bis auf den letzten ignoriert werden, was als Sequenz-zu-Vektor Netzwerk bezeichnet wird. Andersherum kann ein RNN auch einen Vektor als Eingabe erhalten und eine Sequenz ausgeben, wodurch z.B. eine Beschreibung für ein Bild generiert werden könnte. Letztlich ist auch ein sog. Encoder-Decoder Modell möglich, wobei ein Sequenz-zu-Vektor Netzwerk zuerst eine Repräsentation der Eingabesequenz erzeugt, und ein Vektor-zu-Sequenz Netzwerk daraufhin eine Ausgabe aus dieser Repräsentation erzeugt. Ein Anwendungsfall dafür sind Sprachanwendungen wie ein Übersetzer, da jedes Wort eines Textes im Vornherein bekannt sein muss, um eine korrekte Ausgabe zu erzeugen. (ebd., vgl. S. 497 ff.)

### 2.1.2 Training neuronaler Netze mit Backpropagation und Gradient Descent

Das Gradienten-Verfahren ist ein Optimierungs-Algorithmus, der mithilfe einer Fehlerfunktion die Parameter eines Modells so anpasst, dass die Fehlerfunktion minimiert wird. Um das zu erreichen, muss das Verfahren erst die Steigung der Fehlerfunktion berechnen können, um dann iterativ die Parameter anzupassen. (ebd., vgl. S. 118)

„Neurons wire together if they fire together“ (Lowel, S., Singer, W., 1992)

Dieses Zitat prägt das sog. Hebbian learning, eine Regel die beschreibt wie sich die Gewichte zwischen Neuronen relativ zu deren Aktivierungen verändern. Ein Perceptron wird mit einer Abwandlung dieser Regel trainiert, die außerdem den Fehler der Ausgabe mit einbezieht und diejenigen Gewichte verstärkt, die zu einer Verringerung des Fehlers führen. (Géron, A., 2019, vgl. S. 289 ff.) Die Lernregel lautet damit:

#### Formel 1: Hebbian-Learning angewendet auf ein Perceptron

$$w_{i,j}^{(nextstep)} = w_{i,j} + \eta (y_j - \hat{y}_j) \quad (1)$$

Quelle: Géron, A., 2019, S. 289

Ein Verfahren zum Trainieren eines MLP stellt das von *Rumelhart, D. E., Hinton, G. E., Williams, R. J.*, 1986 vorgestellte Backpropagation dar. Dieses basiert auf dem Gradienten-Verfahren, mit der Eigenschaft die Gradienten der Gewichte in allen Ebenen des MLP mit Bezug auf jeden einzelnen Parameter effizient berechnen zu können. Die Trainingsdaten werden in mehreren Epochen im folgenden Ablauf durchlaufen:

Zuerst wird für jede Instanz der Trainingsdaten das MLP im Forward Pass durchlaufen. Die Ausgabe jedes Neurons der versteckten Ebenen wird dabei zwischengespeichert. Nun wird die Ausgabe des MLP anhand der Fehlerfunktion bestimmt. Die Gradienten aller Gewichte zwischen Neuronen der versteckten Ebenen werden im Reverse Pass bestimmt. Dazu wird der Beitrag jedes Ausgabe-Neuronen zum Fehler berechnet und gleiches rekursiv für die Neuronen der versteckten Ebenen wiederholt. Mit den berechneten Werten kann abschließend das Gradienten-Verfahren angewendet werden. (Géron, A., 2019, S. 286)

## 2.2 Analoge Computer

### 2.2.1 Geschichte zu analogen Computern

Die Geschichte des analogen Computers reicht bis in die Antike zurück. So wurde im Jahr 1900 der Antikythera-Mechanismus in einem Schiffswrack entdeckt, der aus etwa 100 v. Chr. stammt. Er gilt als eines der komplexesten mechanischen und mathematischen Geräte der Antike und konnte die Bewegungen von Himmelskörpern modellieren sowie Sonnen- und Mondfinsternisse vorhersagen. (Ullmann, B., 2022, vgl. S. 9 f.)

Im Verlauf der technischen Entwicklung wurden Gleitkomma-Rechner konstruiert, mechanische Geräte, die zur Lösung komplexer mathematischer Probleme, wie der Berechnung von Bombenflugbahnen und Feuerleitsystemen, eingesetzt wurden. Diese fanden auch in friedlichen Anwendungen wie der Gezeiten-Berechnung Verwendung. (ebd., vgl. S. 9)

Mit dem Fortschritt der Technik entstanden die ersten elektronischen analogen Computer, die von Helmut Hoelzer in Deutschland und George A. Philbrick in den USA entwickelt wurden. Diese Computer wurden primär für militärische Anwendungen wie Flugbahn- und Feuerleitsysteme genutzt. Ein Beispiel für einen frühen elektronische Analogrechner ist Hoelzers Mischgerät, das während des zweiten Weltkriegs in Deutschland entwickelt wurde und Elektrohnenröhren zur Durchführung von Berechnungen verwendete. (ebd., vgl. S. 41 f.)

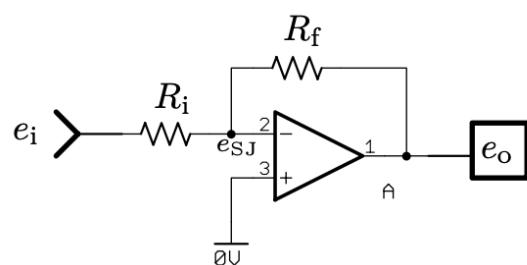
Ein weiteres bedeutendes Gerät war der Caltech-Computer, ein elektronischer Analogrechner, der zwischen 1946 und 1947 entwickelt wurde und etwa 15 Tonnen wog. Dieser Computer wurde zur Lösung komplexer mathematischer und wissenschaftlicher Probleme, einschließlich Differentialgleichungen, entwickelt. (Ulmann, B., 2022, vgl. S. 69)

George A. Philbrick spielte eine bedeutende Rolle in der Weiterentwicklung und Verbreitung elektronischer Analogrechner. Er führte kommerzielle Operationsverstärker ein und entwickelte in den 1950er Jahren modulare elektronische Analogrechner, was einen großen Einfluss auf die Standardisierung und Verbreitung dieser Technologie hatte. (ebd., vgl. S. 136)

## 2.2.2 Typische Komponenten und Bauweisen analoger Computer

Der Operationsverstärker gilt als zentraler Baustein für die meisten Komponenten eines analogen Rechners. Er bildet die Differenz aus zwei Eingangssignalen und verstärkt dieses Signal anschließend. Werden ein freies und ein geerdetes Eingangssignal genutzt, so wird nur das Freie verstärkt. Durch ein Rückkopplungs-Signal gelingt es dem Operationsverstärker einen Teil der Stromverstärkung aufzugeben, um Stabilität zu gewinnen. Dieses Signal verbindet die Ausgabe der Komponente mit seiner Eingabe, wobei diese beiden Werte miteinander summiert werden. Ein wichtiges Merkmal dieser Komponente ist außerdem, dass das Vorzeichen der Eingabe gedreht wird. (ebd., vgl. S. 73 f.)

**Abbildung 1: Operationsverstärker mit Rückkopplung**



Quelle: ebd., S. 76

Die Formel zu Berechnung der Ausgangsspannung lautet damit wie folgt:

### Formel 2: Ausgangsspannung eines Operationsverstärkers

$$e_o = \frac{R_f}{R_i} e_i \quad (2)$$

Quelle: Ebd., S. 76

Durch Anpassungen des Eingangssignals bzw. des Rückkopplungs-Signals kann die Funktion des Operationsverstärkers geändert werden. So sind hiermit Operationen wie Summieren, Subtrahieren, Invertieren, das Multiplizieren von Konstanten, Integration und (bedingt) die Differenzierung möglich.

Im Prozess der Verstärkung der Signale kann ein sog. Drift auftreten. Dadurch werden kleine Fehler in der Signalverarbeitung verstärkt und führen zu großen Fehlern am Ausgang. Dieses Problem kann durch Drift-Stabilisierung gelöst werden, wie beschrieben in einem Patent von *Goldberg, E. A., Jules, L.*, 1954. Das durch Drift verursachte Fehlersignal kann an der Summe des Eingangs- und Rückkopplungs-Signals ausgelesen werden. Es wird verstärkt und als weiteres Eingangssignal genutzt. Somit wird der Drift ausgereglicht. (*Ulmann, B.*, 2022, vgl. S. 80)

Um einen Summierer zu bilden, muss lediglich ein weiteres Eingangssignal zum Operationsverstärker hinzugefügt werden. Die Widerstände  $R_i$  bilden die Koeffizienten der Eingangssignale. Der Summierer bildet also die gewichtete Summe der Eingangssignale und gibt diese invertiert aus. (ebd., vgl. S. 86)

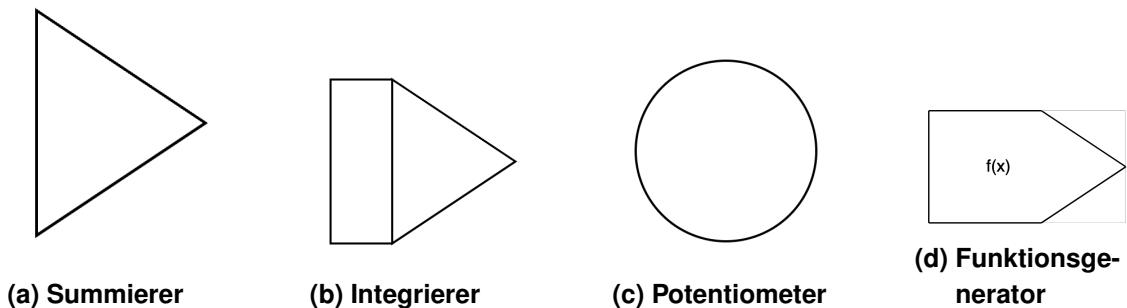
Wird der Feedback-Widerstand  $R_f$  durch einen Kondensator  $C$  ersetzt, wird die Summe der Eingangssignale über Zeit integriert. Man erhält also einen Integrierer. Durch ein Steuerelement  $IC$  wird ein Initialwert festgelegt, der so lange ausgegeben wird, bis ein weiteres Steuerelement  $OP$  betätigt wird. Im aktivierte Zustand wird so lange das Integral der Eingangssignale ausgegeben, bis der  $OP$  Schalter erneut betätigt und der letzte gespeicherte Wert ausgegeben wird. (ebd., vgl. S. 89 ff.)

Die Anwendung eines Koeffizienten am Operationsverstärker kann durch ein Potentiometer erfolgen, welches am Eingangssignal angeschlossen ist. Das somit erhaltene Koeffizient-Potentiometer lässt sich über einen speziellen Zustand des Analogrechners, dem *Potentiometer Set* (kurz: *Pot Set*) setzen. In diesem Zustand werden alle Rechen-einheiten des Analogrechners deaktiviert, sodass das rohe Eingangssignal weitergeleitet wird. In großen analogen Rechnern werden Potentiometer durch Servomotoren oder in hybriden Rechnern digital gesteuert. (ebd., vgl. S. 92 ff.)

Ein Funktionsgenerator kann genutzt werden, um eine beliebige Funktion  $f(x, \dots)$  abzubilden. Für analoge Rechner ist das aber eine komplexe Aufgabe, weshalb sich für diesen Anwendungsfall hybride Systeme besonders eignen. Als mögliche analoge Ansätze bieten sich der servogetriebene Funktionsgenerator, der Kurvenfolger oder der Fotoformer an (ebd., vgl. S. 97 ff.).

Die im ersten Augenblick einfache Multiplizierung ist auf analogen Rechnern komplex zu implementieren (ebd., vgl. S. 105). Moderne analoge Rechner greifen typischerweise auf

**Abbildung 2: Symbole zur Darstellung eines analogen Computers**



Quelle: Eigene Darstellung

die Gilbert Zelle zurück. Die Berechnung einer Division bzw. Quadratwurzel kann durch modifizierte Multiplikatoren erfolgen. (*Ullmann, B., 2022, S. 114 f.*)

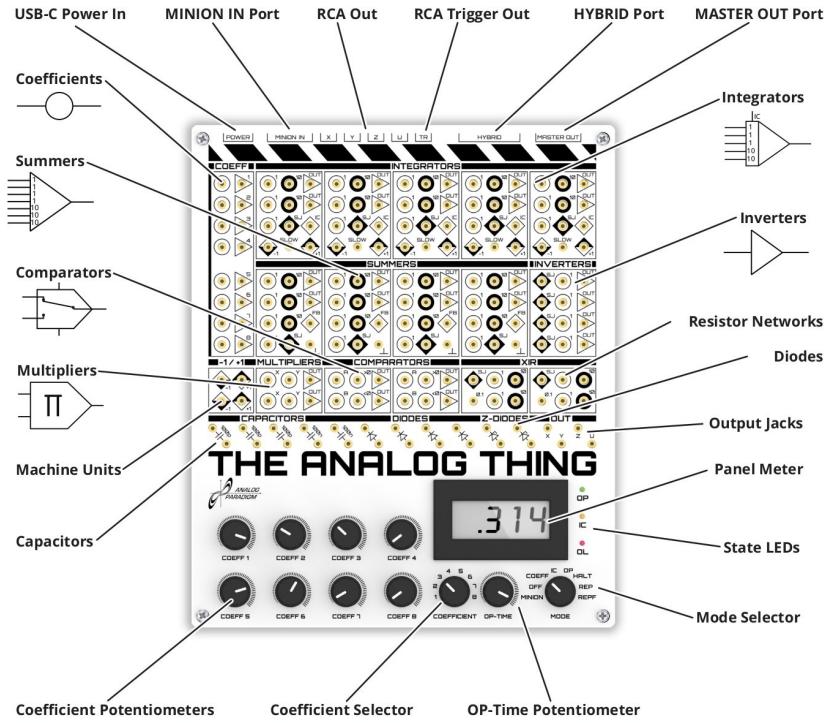
Ein Komparator kann z. B. zur Implementierung einer Schrittfunktion oder zum Tauschen von Variablen genutzt werden. Um das zu erreichen, reicht es schon aus, Dioden in der Feedback-Schleife eines Operationsverstärkers zu platzieren. Nach einem ähnlichen Prinzip kann auch ein Begrenzer erzeugt werden, dessen Eingangssignal zwischen einer Ober- und Untergrenze limitiert wird. (ebd., S. 116)

Die im Jahr 2020 gegründete anabrid GmbH hat es sich zum Ziel gesetzt, analoge und digitale Technologien zu kombinieren, um die technologischen Herausforderungen der Zukunft effizient lösen zu können. Dazu entwickelt und vertreibt das Unternehmen rekonfigurierbare analoge Rechner, wie den 2024 erschienenen „lucidac“ oder das rudimentäre „The Analog Thing“ (*GmbH, A., 2024a*).

„THE ANALOG THING is a high-quality, low-cost, open-source, and not-for-profit cutting-edge analog computer. You can think of it as a kind of Raspberry Pi that computes with continuous voltages rather than with zeroes and ones.“  
 (GmbH, A., 2024b)

Das THAT orientiert sich am historischen Design der analogen Computer und arbeitet demnach mit Patchkabeln, mit denen Rechenelemente verschaltet werden (siehe Abbildung 3). Dabei arbeitet das Gerät mit 10 Volt und in einem Wertebereich von  $\pm 1$ . Fällt eine Spannung außerhalb dieses Wertebereichs, fällt das Gerät in den sog. „Overload“-Modus und signalisiert dies über eine LED. Außerdem bietet das Gerät Anschlüsse für einige Variablen, welche als Ein- oder Ausgaben zur Auswertung oder Verschaltung mit weiteren analogen Rechnern genutzt werden können (ebd.).

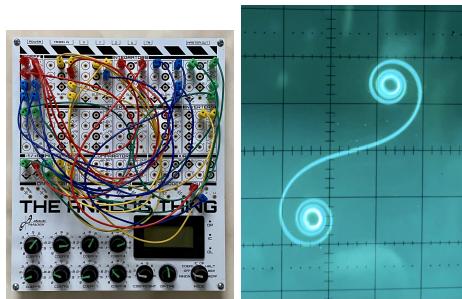
**Abbildung 3: The Analog Thing des Herstellers anabrid GmbH**



Quelle: *GmbH, A., 2024b*

Wie in Abbildung 3 zu sehen, bietet das THAT die grundlegenden Komponenten wie Potentiometer, Summierer, Integrierer, Multiplizierer und Komparatoren an. Diese Komponenten reichen bereits aus, um Probleme wie die Euler-Spirale zu lösen (Siehe Abbildung 4)

**Abbildung 4: Simulation der Euler-Spirale auf einem THAT**



Quelle: ebd.

Das neueste Modell des Herstellers Anabrid, der „lucidac“, lässt sich digital über Software programmieren, wodurch der manuelle Aufwand durch die Verkabelung des Rechners wegfällt (*GmbH, A., 2025, vgl.*). Dafür wurde eine Python-Bibliothek entwickelt, die durch Befehle wie „connect(a, b)“ Verbindungen zwischen Komponenten herstellen kann (*GmbH, A., 2024c, vgl.*). Um die software-gesteuerte Programmierung zu ermöglichen, verwendet der lucidac eine Verbindungs-Matrix, welche der Verbindung aller Komponenten ein Ge-

wicht zuweist. Wird ein Wert innerhalb der Matrix von null auf z. B. eins geändert, so wird eine neue (gewichtete) Verbindung hergestellt (*GmbH, A., 2025*). Seien  $C^{in}$  bzw.  $C^{out}$  die Ein- bzw. Ausgaben der Komponenten und  $M$  die gewichteten Verbindungen zwischen Komponenten, so gilt:

#### **Formel 3: Beziehung zwischen Eingangs- und Ausgangssignalen im lucidac**

$$C_i^{in} = \sum_{j=0}^{16} M_{ij} C_j^{out}, i, j \in [0, 16] \quad (3)$$

Quelle: Ebd., S. 11

#### **2.2.3 Aufbau von Schaltkreisen zur Lösung von Differentialgleichungen**

Im Folgenden werden zwei wesentliche Ansätze zum Ableiten eines analogen Computers von einer Reihe an Differentialgleichungen vorgestellt: die Kelvin Feedback Technik sowie die Substitutionsmethode.

Um einen analogen Rechner mithilfe der Kelvin Feedback Technik zu konstruieren, muss die vorliegende Differentialgleichungen so umgestellt werden, dass die höchsten Ableitungen alleine auf der linken Seite stehen. Die Ableitungen niedrigerem Grades können durch Integration der höchsten Ableitung gebildet werden, weshalb mithilfe der höchsten Ableitung die rechte Seite der Gleichung gebildet werden kann. Da die rechte Seite der Gleichung die höchste Ableitung darstellt, kann eine Rückkopplung hergestellt werden. (*Ulmann, B., 2022*, vgl. S. 153 ff.)

Die Substitutionsmethode substituiert Teile einer Gleichung, um Gleichungen ersten Grades zu erhalten. Die Schaltungen der substituierten Gleichungen werden verbunden, um das Ergebnis zu erhalten. Die Anwendung dieser Methode führt aber zu Schwierigkeiten beim Umgang mit Initialwerten ungleich null und ist umständlich für reale Anwendungen. (ebd., vgl. S. 155 ff.)

Der Wertebereich eines analogen Computers liegt i.d.R. bei  $\pm 1$ , weshalb Skalierung notwendig ist, um die berechneten Werte zu realen Werten umzuwandeln. Hierzu kann eine Zuordnung von maschinellen Variablen zu realen Variablen zum Einsatz kommen, jede Zuordnung ist dabei mit einem Skalierungsfaktor versehen. Die Skalierung von Zeit kann auch sinnvoll sein, um die Berechnungen zu beschleunigen. (ebd., vgl. S. 162 ff.)

Als Beispiel wird hier, wie von ebd., S. 168 ff. beschrieben, ein Masse-Feder-Dämpfer System aufgeführt:

Ein Gewicht mit einer Masse  $m$  ist mit einer Feder mit Starrheit  $s$  und einem Dämpfer mit Dämpfungsfaktor  $d$  verbunden. Sei  $y$  die Position des Gewichts, so wirkt durch das Gewicht die Kraft  $my''$ , durch den Dämpfer  $dy'$  und durch die Feder  $sy$ . Die Summe aller Kräfte in einem geschlossenem physikalischen System sind gleich 0, deshalb gilt:

**Formel 4: Differentialgleichungen des Masse-Feder-Dämpfer Systems**

$$my'' + dy' + sy = 0 \quad (4)$$

Quelle: *Ulmann, B.*, 2022, S. 168

Unter Anwendung der Kelvin Feedback Technik ergibt sich folgende Gleichung:

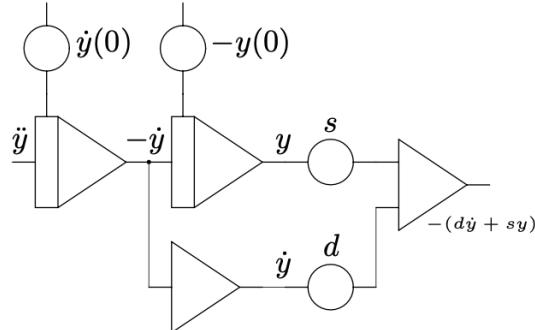
**Formel 5: Masse-Feder-Dämpfer System umgestellt mithilfe der Kelvin Feedback Technik**

$$y'' = \frac{-(dy' + sy)}{m} \quad (5)$$

Quelle: Ebd., S. 168

Der Schaltkreis zur Lösung von  $-(dy' + sy)$  kann wie in Abbildung 5 gezeigt gebildet werden. Die Rückkopplung zu  $y''$  ist in Abbildung 6 dargestellt.

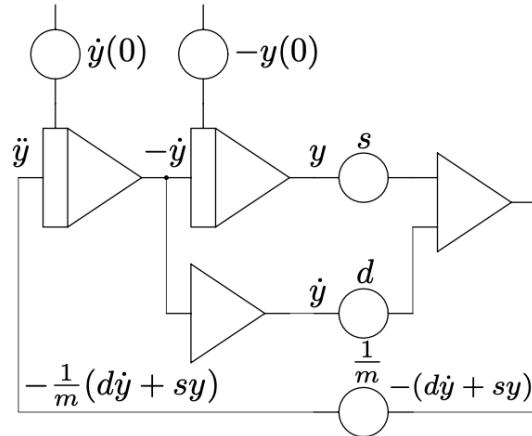
**Abbildung 5: Masse-Feder-Dämpfer System ohne Rückkopplung**



Quelle: ebd., S. 170

Zur Berechnung der realen Werte müssen die berechneten Werte entsprechend skaliert werden. Die Informationen über Maximalwerte von  $y$ ,  $y'$  und  $y''$  ist dafür notwendig.

**Abbildung 6: Vollständiges Masse-Feder-Dämpfer System**



Quelle: Ullmann, B., 2022, S. 170

## 2.3 Energiebasierte Modelle

### 2.3.1 Definition: energiebasierte Modelle und energiebasiertes Lernen

Energiebasierte Modelle (EBM) werden im maschinellen Lernen eingesetzt und nutzen eine Energiefunktion, die jeder möglichen Variablenkonfiguration einen skalaren Energiewert zuweist. In einem neuronalen Netz könnten diese Variablen beispielsweise die Eingabegrößen, die Parameter, versteckte Variablen sowie die Ausgabewerte umfassen. Bei der Inferenz des Modells werden zunächst die EingabevARIABLEN festgelegt, bevor ein Minimum der Energiefunktion bestimmt wird. Sobald dieses Minimum gefunden ist, lässt sich die Ausgabe des Modells anhand der AusgabevARIABLEN ablesen.

Das Training eines EBM erfolgt durch Anpassung seiner Energiefunktion, sodass korrekte Werte mit geringeren Energien und falsche Werte mit höheren Energien bewertet werden (LeCun, Y. et al., 2006). Zu den ersten vorgestellten EBM zählen das HNN (Hopfield, J. J., 1984) sowie die darauf basierende Boltzmann-Maschine (Ackley, D. H., Hinton, G. E., Sejnowski, T. J., 1985).

### 2.3.2 Das Hopfield-Netzwerk: Eine Herangehensweise an neuronale Netze

In den frühen 1980er Jahren stellte *Hopfield* ein neuronales Netzwerkmodell vor, das als bedeutender Meilenstein in der Erforschung kollektiver Berechnungsfähigkeiten gilt. Ziel dieses Modells war es, die Funktionsweise stark vernetzter neuronaler Systeme zu verstehen und ihre Potenziale für Mustererkennung, fehlerresistente Speicherung und assoziatives Gedächtnis zu untersuchen. Diese Netzwerke dienen auch als Modelle für biologisch

inspirierte Rechner, die parallele und robuste Berechnungen ermöglichen sollen (*Hopfield, J. J., 1982*, vgl. S. 2554) (*Hopfield, J. J., 1984*, vgl. S. 3088).

Die Netzwerktopologie basiert auf vollständig miteinander verbundenen Neuronen, die durch eine symmetrische Gewichtsmatrix  $T_{ij}$  miteinander verknüpft sind. Diese Symmetrie ist entscheidend, um stabile Zustände zu gewährleisten und chaotisches Verhalten zu vermeiden. Ursprünglich wurden die Neuronen mit binären Zuständen modelliert, bei denen jedes Neuron entweder „aktiv“ (1) oder „inaktiv“ (0) ist, was an das McCulloch-Pitts-Modell angelehnt ist (*Hopfield, J. J., 1982*, vgl. S. 2555). Spätere Arbeiten erweiterten das Modell durch die Einführung von Neuronen mit kontinuierlichen, sigmoidalen Ausgabefunktionen, um die biologischen Realitäten besser abzubilden (*Hopfield, J. J., 1984*, vgl. S. 3088).

Ein zentrales Konzept des HNN ist die Minimierung einer Energiefunktion  $E$ , die den Zustand des Netzwerks beschreibt. Diese Funktion ist so definiert, dass sie durch asynchrone Aktualisierungen der Neuronen monoton abnimmt, bis ein lokales Minimum erreicht ist. Diese Minima entsprechen stabilen Zuständen des Netzwerks, die gespeicherte Erinnerungen repräsentieren. Diese Eigenschaft erlaubt es dem Netzwerk, unvollständige oder verrauchte Eingaben zu vervollständigen, wodurch es robust gegenüber Störungen und Ausfällen einzelner Neuronen ist. Die Fähigkeit, Informationen anhand von Teilinformationen abzurufen, macht das HNN zu einem echten inhaltsadressierbaren Speicher (*Hopfield, J. J., 1982*, vgl. S. 2554 f.).

Das ursprüngliche Modell mit binären Zuständen ist besonders für digitale Berechnungen geeignet und lässt sich effizient simulieren. Das kontinuierliche Modell mit sigmoidalen Neuronen hingegen berücksichtigt biologische Eigenschaften wie graduelle Reaktionskurven und Verzögerungen durch synaptische Übertragungen. Diese Modelle zeigen, dass dieselben kollektiven Eigenschaften, wie sie im ursprünglichen binären Modell beobachtet wurden, auch in biologisch realistischeren Systemen auftreten können. Dies stärkt die These, dass solche kollektiven Eigenschaften tatsächlich in natürlichen neuronalen Netzwerken vorkommen (*Hopfield, J. J., 1984*, vgl. S. 3089).

Ein weiterer Aspekt ist die Kapazität des Netzwerks, mehrere stabile Zustände oder Erinnerungen gleichzeitig zu speichern. Studien zeigen, dass ein Netzwerk mit  $N$  Neuronen etwa  $0,15N$  stabile Zustände speichern kann, bevor die Fehlerrate signifikant ansteigt. Die Speicherkapazität ist somit begrenzt, kann jedoch durch gezielte Anpassungen der Schwellenwerte und Gewichte erhöht werden. Darüber hinaus bietet das Netzwerk eine hohe Fehlertoleranz und kann ähnliche Muster in Kategorien zusammenfassen, was es zu einem effektiven Werkzeug für Mustererkennungsaufgaben macht (*Hopfield, J. J., 1982*, vgl. S. 2556) (*Hopfield, J. J., 1984*, vgl. S. 3091).

Das HNN hat nicht nur Bedeutung für die Neurobiologie, sondern auch für technische Anwendungen. Seine analoge Implementierung in integrierten Schaltkreisen ermöglicht die Entwicklung von robusten, fehlertoleranten Speichern und Prozessoren. Dieses Netzwerk ist insbesondere für parallele Berechnungen und selbstorganisierende Systeme nützlich, was es für Anwendungen im maschinellen Lernen und in autonomen Systemen relevant macht. Durch seine Fähigkeit, kollektive Berechnungen durchzuführen, bietet es eine Grundlage für fortschrittliche Algorithmen in der künstlichen Intelligenz (*Hopfield, J. J., 1982*, vgl. S. 2554 ff.).

Neuere Forschungen untersuchen die Auswirkungen von Asymmetrien in der Gewichtsmatrix und die Einbeziehung von nichtlinearen Dynamiken, um zeitabhängige Sequenzen und komplexere Berechnungen zu ermöglichen. Diese Erweiterungen zeigen, dass das HNN flexibel genug ist, um als Grundlage für vielfältige Anwendungen zu dienen. Seine Robustheit gegenüber Rauschen und Störungen macht es besonders geeignet für reale Umgebungen, in denen Perfektion selten ist (ebd., vgl. S. 2557) (*Hopfield, J. J., 1984*, vgl. S. 3092).

### **2.3.3 Energiebasiertes Lernen am Beispiel Equilibrium Propagation**

#### **2.3.3.1 Mathematische Grundlagen**

##### **TODO: Seitenzahlen für Quellenangaben**

Ein Verfahren zum Trainieren EBM stellt das EP dar. Dabei wird der Gradient der Energiefunktion berechnet, um die Modellparameter zu optimieren. Anstatt Vorhersagen explizit zu definieren, werden diese implizit aus den Datenpunkten und Parametern abgeleitet, wodurch sich das Verfahren besonders für analoge Computer eignet. (*Scellier, B., Bengio, Y., 2017*)

Der Zustand des Modells wird durch den Vektor  $s$  repräsentiert, wobei  $v$  die EingabevARIABLEN und  $\theta$  die Modellparameter bezeichnet. Die Zustandsvariable  $s$  entwickelt sich über die Zeit hinweg so, dass die Energiefunktion  $E(\theta, v, s)$  minimiert wird. Neben dieser Funktion wird auch eine Kostenfunktion  $C(\theta, v, s)$  definiert, welche die Abweichung der Modellausgabe von den Zielwerten beschreibt. Eine geringere Energie für eine bestimmte Variablenkonfiguration sollte mit einer entsprechend niedrigeren Kostenfunktion einhergehen. Zudem wird der Einflussparameter  $\beta$  eingeführt, der als Skalierungsfaktor für die Kostenfunktion dient. Die Gesamtenergiefunktion  $F$  im Rahmen des EP lautet damit:

---

**Formel 6: Gesamtenergiefunktion im EP**

$$F(\theta, v, \beta, s) := E(\theta, v, s) + \beta C(\theta, v, s) \quad (6)$$

Quelle: *Scellier, B., Bengio, Y.*, 2017, S. 5

Die Fixpunkte des Modells werden als  $s_{(\theta, v)}^\beta$  bezeichnet und entsprechen jeweils lokalen Minima der Gesamtenergiefunktion  $F$ . Für  $\beta = 0$  ergibt sich  $s_{(\theta, v)}^0$ , das Minimum der Energiefunktion  $E$ , welches die Vorhersage des Modells repräsentiert. (ebd.)

Die Zielfunktion  $J$ , die im Equilibrium Propagation-Ansatz optimiert wird, lautet:

**Formel 7: Zielfunktion im EP**

$$J(\theta, v) = C(\theta, v, s_{(\theta, v)})^0 \quad (7)$$

Quelle: Ebd., S. 5

Während die Kostenfunktion  $C$  die Modellqualität für beliebige Werte von  $\beta$  beschreibt, betrachtet  $J$  diese ausschließlich für  $\beta = 0$ . Der Gradient der Zielfunktion  $J$  nach den Parametern  $\theta$  ergibt sich durch:

**Formel 8: Gradient der Zielfunktion über die Parameter des Modells im EP**

$$\frac{\partial J}{\partial \theta}(\theta, v) = \lim_{\beta \rightarrow 0} \frac{1}{\beta} \left( \frac{\partial F}{\partial \theta}(\theta, v, \beta, s_{(\theta, v)}^\beta) - \frac{\partial F}{\partial \theta}(\theta, v, \beta, s_{(\theta, v)}^0) \right) \quad (8)$$

Quelle: Ebd., S. 6

Daraus ergibt sich die Änderungsrate von  $\theta$ :

**Formel 9: Änderungsrate der Parameter im EP**

$$\Delta \theta \propto -\frac{1}{\beta} \left( \frac{\partial F}{\partial \theta}(\theta, v, \beta, s_{(\theta, v)}^\beta) - \frac{\partial F}{\partial \theta}(\theta, v, \beta, s_{(\theta, v)}^0) \right) \quad (9)$$

Quelle: Ebd., S. 6

Hieraus resultiert in der praktischen Anwendung auf das HNN ein zweistufiger Lernprozess. In der ersten Phase wird eine Inferenz durchgeführt, bei der ein Minimum der Energiefunktion ermittelt und die Ausgabe des Netzwerks ausgelesen wird. Während dieser Phase gilt  $\beta = 0$ . In der zweiten Phase wird  $\beta > 0$  gesetzt, wodurch die Ausgabe des Netzwerks in Richtung des Zielwertes gesteuert wird. Zu Beginn dieser Phase befinden sich die versteckten Variablen des Netzwerks noch im Gleichgewicht, jedoch breitet sich

die Störung der Ausgabevervariablen im Laufe der Zeit auf diese aus. Betrachtet man ein mehrschichtiges Netzwerk, so propagiert die Störung rückwärts durch die Struktur des Netzwerks, ein Prinzip, das auch als Backpropagation bekannt ist. (Scellier, B., Bengio, Y., 2017)

### 2.3.3.2 Theoretische Anwendung am Beispiel eines Hopfield-Netzwerks

Wie von ebd. bereits beschrieben, ist EP auf das HNN anwendbar. Für diese Anwendung muss zuerst die Energiefunktion des Modells bestimmt werden. Seien  $u$  die Neuronen des Netzwerks,  $W_{i,j}$  die Gewichte der Verbindung zweier Neuronen und  $b_i$  der Bias eines Neurons, so können die Parameter des Modells als  $\theta = (W, b)$  definiert werden. Die Aktivierungsfunktion ist definiert als  $\rho(u_i)$ . Zusätzlich werden die Neuronen aufgeteilt in Eingabeneuronen  $x$ , verdeckte Neuronen  $h$  und Ausgabeneuronen  $y$ , die Gesamtheit der Neuronen im Netzwerk ist damit  $u = \{x, h, y\}$ . Mit diesen Variablen kann nun die Hopfield-Energiefunktion aufgestellt werden:

#### Formel 10: Energiefunktion in Anlehnung an das HNN

$$E(u) := \frac{1}{2} \sum_i u_i^2 - \frac{1}{2} \sum_{i \neq j} W_{ij} \rho(u_i) \rho(u_j) - \frac{1}{2} \sum_i b_i \rho(u_i) \quad (10)$$

Quelle: In Anlehnung an Bengio, Y., Fischer, A., 2015, S. 2

Um die Kostenfunktion  $C$  aufzustellen, müssen noch die Zielwerte  $d$  definiert werden, welche die für die Eingabewerte korrekten Ausgabewerte beinhalten. Damit lautet die Kostenfunktion:

#### Formel 11: Kostenfunktion im EP für ein HNN

$$C := \frac{1}{2} \|y - d\|^2 \quad (11)$$

Quelle: Scellier, B., Bengio, Y., 2017, S. 3

Diese Funktion kann genutzt werden, um die Ausgabeneuronen in Richtung der Zielwerte zu schieben. Aus der Energiefunktion kann zusammen mit der Kostenfunktion die Gesamtenergiefunktion  $F$  gebildet werden:

#### Formel 12: Gesamtenergiefunktion im EP für ein HNN

$$F := E + \beta C \quad (12)$$

Quelle: Ebd., S. 3

Die Zustandsvariable  $s$  ist definiert als  $s = \{h, y\}$ . Sie besteht aus den versteckten und den Ausgabeneuronen und beinhaltet nicht die Eingabeneuronen, da diese immer festgelegt sind. Der Gradient dieser Zustandvariable über Zeit ist gegeben durch  $\frac{ds}{dt} = -\frac{\partial F}{\partial s}$ , wodurch die Energiefunktion minimiert und somit ein Fixpunkt des Netzwerks gefunden wird. Dieser Gradient kann so betrachtet werden, dass zwei Kräfte auf ihn wirken:

**Formel 13: Dynamik der Zustände eines HNN im EP**

$$\frac{ds}{dt} = -\frac{\partial E}{\partial s} - \beta \frac{\partial C}{\partial s} \quad (13)$$

Quelle: Scellier, B., Bengio, Y., 2017, S. 3

Hierbei ist die durch die Hopfield-Energiefunktion ausgeübte Kraft:

**Formel 14: Auswirkung der Energiefunktion auf die Zustände des Netzwerks**

$$-\frac{\partial E}{\partial s_i} = \rho'(s_i) \left( \sum_{i \neq j} W_{ij} \rho(u_j) + b_i \right) - s_i \quad (14)$$

Quelle: Ebd., S. 3

Durch die Kostenfunktion wirken folgende Kräfte:

**Formel 15: Auswirkung der Kostenfunktion auf die Zustände des Netzwerks**

$$-\beta \frac{\partial C}{\partial h_i} = 0 \quad (15)$$

$$-\beta \frac{\partial C}{\partial y_i} = \beta(d_i - y_i) \quad (16)$$

Quelle: Ebd., S. 3

Im zweiphasigen Lernprozess, bestehend aus der freien und der festen Phase, wird das Netzwerk trainiert. In der freien Phase mit  $\beta = 0$  wird Inferenz durchgeführt, wodurch das Netzwerk zum freien Fixpunkt  $u^0$  konvergiert. Die feste Phase mit  $\beta > 0$  bringt den festen Fixpunkt  $u^\beta$  hervor. Daraus lässt sich die Lernregel für das HNN ableiten:

**Formel 17,18: Lernregel des EP für das HNN**

$$\Delta W_{ij} \propto \frac{1}{\beta} \left( \rho(u_i^\beta) \rho(u_j^\beta) - \rho(u_i^0) \rho(u_j^0) \right) \quad (17)$$

$$\Delta b_i \propto \frac{1}{\beta} \left( \rho(u_i^\beta) - \rho(u_i^0) \right) \quad (18)$$

Quelle: *Scellier, B., Bengio, Y.*, 2017, S. 3

### 3 Methodik

Im Folgenden wird der methodische Ansatz zur Umsetzung der Equilibrium Propagation auf einem analogen Computer beschrieben. Dafür wird die Forschungsstrategie erläutert, die auf der Konstruktion eines Simulationsmodells basiert. Anschließend erfolgt die Auswahl einer geeigneten Simulationsumgebung, wobei verschiedene Softwarelösungen hinsichtlich ihrer Eignung für die Modellierung analoger Rechensysteme verglichen werden. Zudem wird die Referenzarbeit für das Netzwerkdesign vorgestellt, auf deren theoretische Grundlagen sich die Implementierung stützt. Abschließend werden die erwarteten Ergebnisse der Simulation formuliert, um die Zielsetzung der Arbeit klar zu definieren.

#### 3.1 Forschungsansatz: Konstruktion

Der Forschungsansatz dieser Arbeit ist die Konstruktion eines Simulationsmodells zur Umsetzung des EP Algorithmus zur Anwendung auf ein HNN. Ziel ist es, eine funktionale Implementierung in einer geeigneten Simulationsumgebung zu realisieren und anhand dieser die Übertragbarkeit des Algorithmus auf analoge Computer zu untersuchen.

#### 3.2 Auswahl und Grundlagen der Simulationssoftware

Zur Simulation elektrischer Schaltkreise kann die branchenübliche Software SPICE (Simulation Program with Integrated Circuit Emphasis) zum Einsatz kommen. Auf Basis von SPICE werden von verschiedenen Herstellern Programme entwickelt, welche das Arbeiten durch z. B. ein grafisches Benutzerinterface vereinfachen oder die Software erweitern. Zu diesen Herstellern gehören u. a. National Instruments mit ihrem Produkt „Multisim“ *Instruments, N.*, 2024, Cadence Design Systems mit „PSpice“ (*Systems, C. D.*, 2024) oder die Analog Devices Inc. mit „LTSpice“ (*Devices, A.*, 2024).

Eine Software, die hier näher betrachtet wird, ist LTSpice. Dieses Produkt zeichnet sich in erster Linie dadurch aus, dass es im Gegensatz zu z. B. Multisim kostenfrei nutzbar ist. Es kommt mit einer umfangreichen Bibliothek an vorgefertigten Makromodellen daher, welche das Konstruieren einer Schaltung vereinfachen und beschleunigen können. LTSpice erlaubt es auch, Schaltungen beliebiger Größe zu simulieren, wobei der limitierende Faktor die Rechenleistung des Systems ist (*Alonso, G.*, 2019).

Ein Punkt gegen die Nutzung dieser Software ist aber, dass es keine fertigen Implementierungen der Modelle eines analogen Computers in der mitgelieferten Bibliothek gibt. So

sind zwar grundlegende Bausteine wie der Operationsverstärker, Widerstände und Kondensatoren vorhanden, aber Modelle wie eine Gilbert Zelle zum Multiplizieren oder ein Funktionsgenerator fehlen. Auch öffentlich zugängliche externe Bibliotheken bieten nicht den für diese Arbeit geforderten Umfang an Modellen (*Maffei, P.*, 2024, vgl.). Die Entwicklung einer geeigneten Bibliothek passt nicht in den Umfang dieser Arbeit und erfordert zusätzlich ein nicht vorhandenes Maß an Vorwissen im Bereich Elektrotechnik. Aufgrund dessen eignet sich LTSpice nicht für den Einsatz in dieser Arbeit.

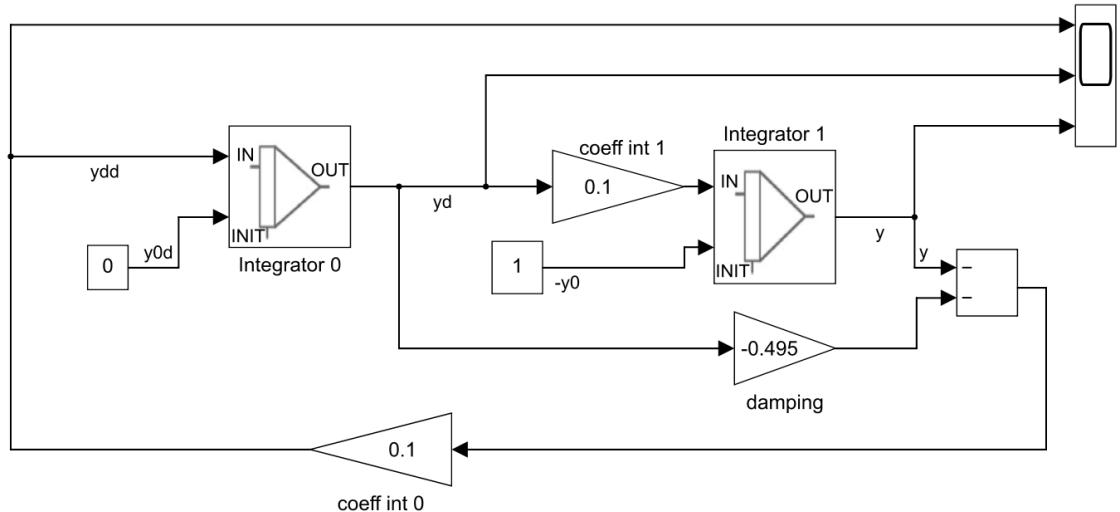
Als Erweiterung zu Matlab kommt Simulink für diese Arbeit infrage, eine grafische Entwicklungsumgebung für modellbasierte Systementwicklung. Simulink zeichnet sich durch die Modellierung von Programmen durch Blockdiagramme aus und stellt Funktionen zur Simulation und Analyse bereit. Des Weiteren lässt sich Source-Code für z. B. C++ aus den Programmen generieren. Ein Modell in Simulink besteht aus Blöcken, Signalen und Kommentaren. Blöcke stellen mathematische Funktionen bereit, Signale verbinden diese Blöcke und können verschiedene Datentypen wie Skalare, Vektoren oder Matrizen darstellen. Kommentare erlauben das beliebige Einfügen von Text und Bildern in die grafische Oberfläche (*Peasley, E., Mear, I. F.*, 2018).

Die von Simulink vorgegebenen Blöcke sind in Bibliotheken aufgeteilt. In „Sources“ befinden sich diverse Blöcke wie Konstanten, Sinus-Kurven oder Puls-Generatoren zur Eingabe in das Modell. „Sinks“ bietet Blöcke zur Darstellung oder dem Export von Signalen, wie dem „Scope“ zur Darstellung eines Graphen oder „Display“ zur einfachen Anzeige von Werten, an. Auch alle für diese Arbeit relevanten mathematischen Operationen sind in Simulink innerhalb der Bibliothek „Math Operations“ vorhanden. Dazu zählen Funktionen wie Summierung, Multiplizierung und Koeffizienten. Die Bibliothek „Continuous“ beinhaltet Funktionen speziell für kontinuierliche Signale, wie sie in analogen Rechner genutzt werden. Dazu zählen primär die Integration bzw. die Ableitung eines Signals. Unter „Discrete“ befinden sich demnach Blöcke für diskrete Signale, die in dieser Arbeit aber nicht zum Einsatz kommen können. Letztlich bietet Simulink eine nützliche Funktion zur Vereinfachung und Validierung von Modellen mit den „User Defined Functions“ an, womit Matlab-Code einfach in ein Simulink-Modell integriert werden kann. (ebd., vgl.)

Wie mithilfe der genannten Blöcke das Masse-Pfeder-Dämpfer System aus Abbildung 6 modelliert werden kann, wurde bereits von *Ullmann* gezeigt:

Zusätzlich zur Standard-Bibliothek lässt sich Simulink durch weitere Produkte erweitern. Zur Modellierung physikalischer Systeme wird z. B. „Simscape“ angeboten, womit Simulink um die Simulation mechanischer, elektrischer, hydraulischer und thermaler Systeme erweitert wird. Durch die Installation dieser Erweiterung werden auch Komponenten der Elektrotechnik wie der Operationsverstärker, Widerstand oder Kondensator zur Bibliothek

Abbildung 7: Simulation eines Masse-Feder-Dämpfer Systems in Simulink



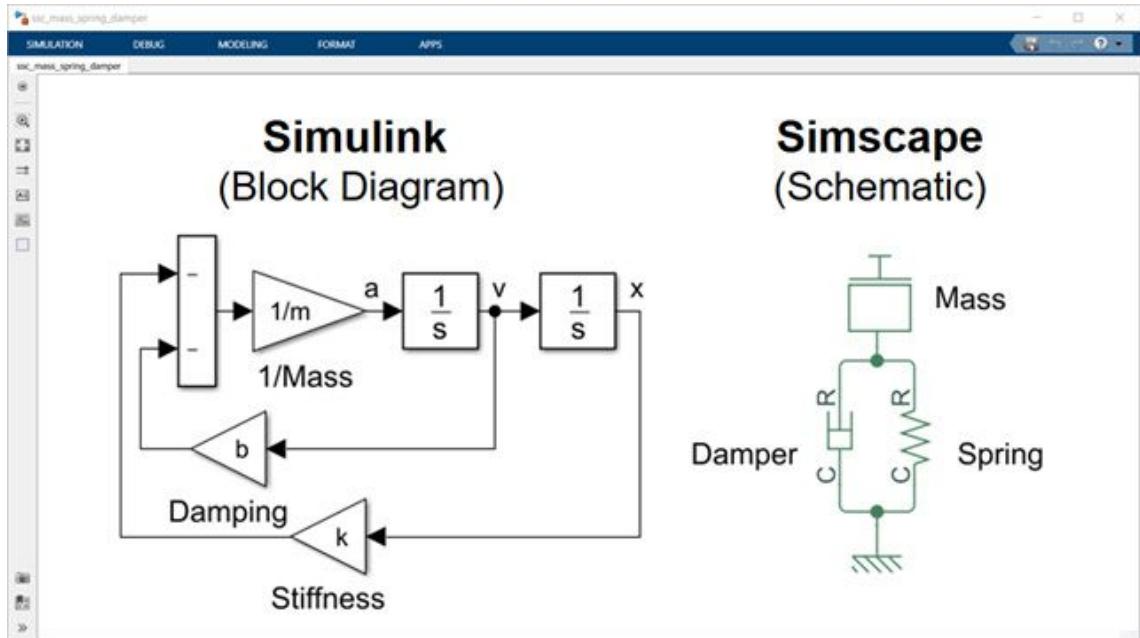
Quelle: Ullmann, B., 2022, S. 241

hinzugefügt, wodurch analoge Rechenelemente modelliert werden können. Eine Implementierung des Masse-Feder-Dämpfer Systems anhand dieser Komponenten ist in Abbildung 8 dargestellt. Physikalische Signale können mit den herkömmlichen Simulink-Blöcken über Signal-Konverter verbunden werden, wodurch Simulink- und Simscape-Modelle miteinander kompatibel werden (MathWorks, 2024). Für diese Arbeit wird jedoch nur Simulink betrachtet, die theoretischen Konzepte werden also nicht physikalisch modelliert.

### 3.3 Auswahl der Referenzarbeit für das Netzwerkdesign

Es haben sich bereits einige Arbeiten mit der Umsetzung eines HNN auf Basis elektrotechnischer Schaltungen beschäftigt. So hat z. B. Guo et al. einen Analog-Digital-Wandler mithilfe von Memristoren zur Implementierung der Gewichte vorgestellt (Guo, X. et al., 2015), welcher aber aufgrund der Komplexität der verwendeten Neuronen hier nicht zum Einsatz kommt. Mathews, Hasler stellte 2023 eine Implementierung eines HNN auf einem Field Programmable Analog Array (FPAA) vor, womit der maximale Schnitt eines Graphen gefunden werden konnte. Diese Umsetzung eignet sich aber aufgrund des Einsatzes eines FPAA auch nicht für diese Arbeit. Ein weiterer Einsatz von Memristoren als Gewichte konnte von Hu et al. gezeigt werden, die in seiner Arbeit vorgestellten Neuronen arbeiten aber mit diskreten Werten (eins oder null) und das vorgestellte Netzwerk weicht vom ursprünglichen Design durch Hopfield ab (Hu, S. G. et al., 2015). Als vierte und letzte Möglichkeit wurde wieder ein mithilfe von Memristoren implementiertes HNN betrachtet, welches 2020

Abbildung 8: Masse-Feder-Dämpfer System in Simulink und Simscape



Quelle: [MathWorks Simscape](#)

von *Hong, Li, Wang* vorgestellt wurde und zur Bildwiederherstellung genutzt werden kann. Da diese Arbeit aber wieder, wie die von *Guo, X. et al., 2015*, komplexe Neuronen vorstellt und einiges an Vorwissen im Bereich Elektrotechnik zur Umsetzung und Verarbeitung in Simulink benötigt, ist sie auch ungeeignet.

*Scellier, Bengio* haben in Ihrer Arbeit über EP bereits die Energiefunktion eines leicht abgewandelten Hopfield-Netzwerks beschrieben und anhand dessen eine Gleichung zur Dynamik des Systems aufgestellt (siehe Kapitel 2.3.3.2). Um den Fokus dieser Arbeit auf die Implementierung des Lern-Algorithmus zu setzen und die Komplexität nicht zu übersteigen, wird ebendiese Dynamik in Simulink implementiert und anhand dessen das EP validiert.

### 3.4 Erwartete Ergebnisse

Die Implementierung des EP in der Simulationsumgebung soll zeigen, dass der Algorithmus auf analoge Systeme übertragbar ist. Es wird erwartet, dass die Simulation die korrekte Funktionsweise des zweiphasigen Lernprozesses bestätigt und stabile Fixpunkte im Netzwerk erreicht werden. Die Modellierung des HNN in Simulink soll eine kontinuierliche Gewichtsaktualisierung ermöglichen und die Wirksamkeit der Lernregel nachweisen. Zudem wird untersucht, ob die gewählte Aktivierungsfunktion und Netzwerkarchitektur die

gewünschte Dynamik erzeugen und inwiefern sich die physikalischen Eigenschaften analoger Rechner in der Simulation realitätsnah abbilden lassen.

## 4 Konstruktion

Hier wird die praktische Umsetzung der Equilibrium Propagation in einer Simulationsumgebung beschrieben. Das HNN wird als Grundlage für das Modell übernommen und in Simulink abgebildet. Darauf aufbauend werden die notwendigen Anpassungen zur Implementierung des EP vorgenommen, insbesondere durch die Integration der Kostenfunktion und der Lernregel. Es folgt eine detaillierte Beschreibung der Simulationen des Netzwerks, einschließlich der Evaluierung der Fixpunkte und der Stabilität der Implementierung. Durch diesen Ansatz wird die Übertragbarkeit des Algorithmus auf analoge Systeme überprüft und eine Grundlage für die anschließende Analyse der Ergebnisse geschaffen.

### 4.1 Simulationsumgebung

#### 4.1.1 Übernahme des Hopfield-Netzwerks

Im Folgenden soll die von *Scellier, Bengio* vorgestellte Dynamik  $\frac{ds}{dt}$  eines HNN (wie bereits im Kapitel 2.3.3.2 gezeigt) in Simulink implementiert werden.

Um die Zustandsvariablen  $s$  darzustellen, können Integratoren genutzt werden, welche standardmäßig über Zeit integrieren. Über die Initialwerte dieser Integratoren wird die Eingabe in das HNN definiert, welche dann über die Dynamik zu einem Fixpunkt gelangen (vgl. Formel "Dynamik der Zustände eines HNN im EP").

Um diesen Term abzubilden werden die Gewichte benötigt, diese und ihre Initialwerte werden vorerst über Konstanten implementiert. Das hier untersuchte Netzwerk soll im Wertebereich  $[0; 1]$  arbeiten, da sich dieser für die Anwendung auf analogen Computern eignet und für z. B. Bildverarbeitung mit 0 für Schwarz bzw. 1 für Weiß genutzt werden kann. Als Aktivierungsfunktion kommt daher eine Sigmoidfunktion, die logistische Funktion  $\rho(x) = \frac{1}{1+e^{-x}}$ , zum Einsatz, welche diesen Wertebereich ausgibt. Diese Funktion kann praktisch über einen Funktionsgenerator und in Simulink über einen Matlab-Funktionsblock (Siehe Anhang Anhang 1) erzeugt werden. Zusätzlich zur Aktivierungsfunktion wird auch ihre Ableitung  $\rho'(x) = \rho(x)(1 - \rho(x))$  benötigt, welche auf selbe Weise implementiert wird.

Diese Komponenten, die Zustände, Gewichte (und Bias) und die Aktivierungsfunktion werden nun gemäß der Formel ?? mit Summierern und Multiplizierern miteinander verschaltet. Ein solches Netzwerk mit zwei Neuronen ist im Anhang Anhang 1 zu sehen. Die gezeigte Schaltung implementiert noch nicht die Kostenfunktion, erreicht aber bereits mit zwei

Neuronen eine hohe Komplexität. Ein Netzwerk mit drei Neuronen, welches zusätzlich die Kostenfunktion implementiert, ist im Anhang Anhang 1 dargestellt, wobei der Bias zur Übersichtlichkeit weggelassen wurde. Aus dieser Schaltung wird ersichtlich, dass die Implementierung eines HNN mit analogen Rechenelementen bereits mit wenigen Neuronen sehr komplex und unübersichtlich wird, weshalb im Weiteren von Vektoren und Matrizen innerhalb von Simulink Gebrauch gemacht wird, siehe Anhang Anhang 1. Durch die Eingabe verschieden großer Vektoren in den „Training“-Block, kann so die Anzahl der Neuronen des Netzwerks geändert werden. Diese Implementierung bietet potenziell auch die Möglichkeit, Trainingsdaten dynamisch aus der Matlab-Umgebung zu laden.

#### 4.1.2 Notwendige Modifikationen am Netzwerk für Equilibrium Propagation

Das EP definiert mit  $C := \frac{1}{2} \|y - d\|^2$  eine Kostenfunktion des HNN, welche mit  $-\beta \frac{\partial C}{\partial y_i} = \beta(d_i - y_i)$  auf die Dynamik der Zustände wirkt. Wie dieser Term in Simulink abgebildet werden kann, ist bereits im Anhang Anhang 1 dargestellt.

Zusätzlich werden die Energiefunktion sowie die Kostenfunktion mithilfe von Matlab-Funktionsblöcken an das Netzwerk angeschlossen, und zur Auswertung bereitgestellt. Mithilfe dieser Auswertungen kann die Funktionsweise des Netzwerks hinsichtlich der Annahme  $\frac{dF}{dt} \leq 0$  (*Scellier, B., Bengio, Y., 2017*, vgl. S. 3) validiert werden.

Die Gewichte wurden bisher als konstanten implementiert, müssen aber für die Anwendung eines Lernalgorithmus dynamisch anpassbar sein. Die von *Scellier, Bengio* vorgestellte Lernregel  $\Delta W_{ij}$  kann auch als Integration der Lernregel  $\frac{dW_{ij}}{dt}$  interpretiert werden (ebd., vgl. S. 5), weshalb zur Implementierung der Gewichte Integratoren zum Einsatz kommen können (gleiches gilt für die Bias-Werte). Typischerweise werden Gewichte in neuronalen Netzen mit Zufallswerten initialisiert, was in Simulink durch den Block „Random Number“ möglich ist. Da das hier implementierte HNN mit einem Vektor als Repräsentation der Zustände arbeitet, müssen die Gewichte durch einen Matlab-Funktionsblock zu einer Matrix zusammengesetzt werden (siehe Anhang Anhang 1). Das angepasste Netzwerk ist im Anhang Anhang 1 abgebildet.

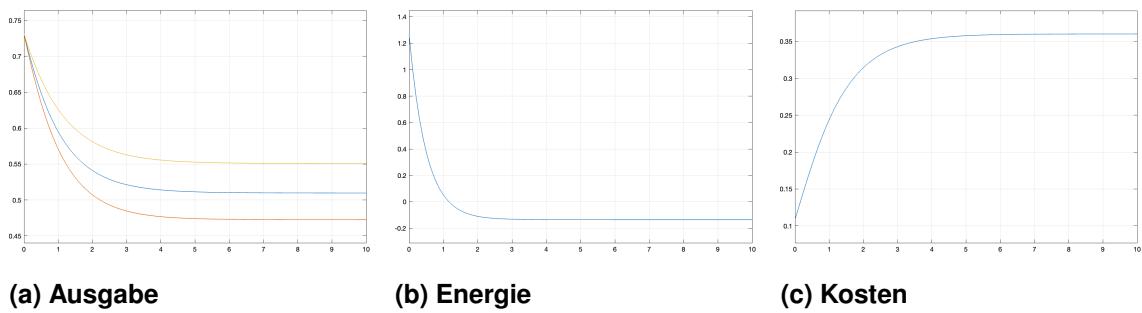
#### 4.1.3 Simulation des angepassten Netzwerks

Im Folgenden werden einige Simulationen des in Simulink implementierten HNN beschrieben, eine Validierung der Funktionsweise des Netzwerkes ist im Anhang Anhang 1 zu finden. Die Gewichte werden einmalig zufällig generiert und mit diesen Werten in allen

weiteren Simulationen weitergearbeitet. Es sollen also keine sinnvollen Anwendungen getestet, sondern ausschließlich die Formen der Ausgaben überprüft werden.

Die erste Simulation wird mit  $\beta = 0$  und  $\vec{d} = (1, 1, 1)$  durchgeführt. Der Vektor  $\vec{d}$  steht dabei für die Eingabe in das Netzwerk bzw. die erwartete Ausgabe. Die Gewichte werden mit  $(-1.21, 1.319, 0.3204)$  zufällig erzeugt. Wie in Abbildung 9 zu sehen, gibt das Netzwerk zu Beginn der Simulation für alle Zustände einen Wert zwischen 0.7 und 0.75 aus, da für alle Zustände der Initialwert 1 gesetzt wurde und  $\rho(1) \approx 0.7311$  gilt. Das Netzwerk findet einen Fixpunkt bei  $y \approx (0.51, 0.47, 0.55)$ , bei dem auch die Energiefunktion ihren niedrigsten Wert erreicht. Die Kostenfunktion steigt proportional zur Energiefunktion, da sich die Ausgabe des Modells immer weiter von der Eingabe  $\vec{d} = (1, 1, 1)$  entfernt.

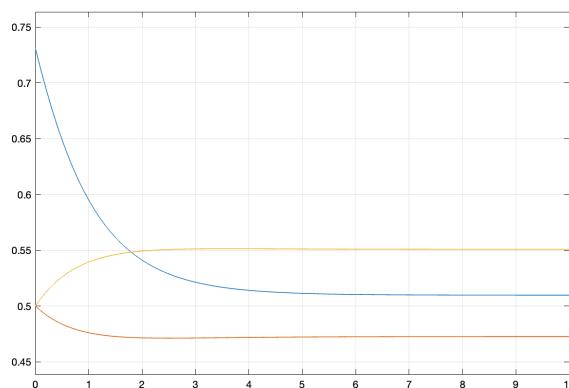
**Abbildung 9: Erste Simulation des HNN**



Quelle: Eigene Darstellung

Für die zweite Simulation wird die Eingabe zu  $\vec{d} = (1, 0, 0)$  geändert. Da das HNN die Eigenschaften eines assoziativen Speichers besitzt (vgl. Kapitel 2.3.2), ist zu erwarten, dass das Netzwerk die gleiche Ausgabe wie aus der vorherigen Simulation liefert. Diese These lässt sich, wie in Abbildung 10 gezeigt, bestätigen.

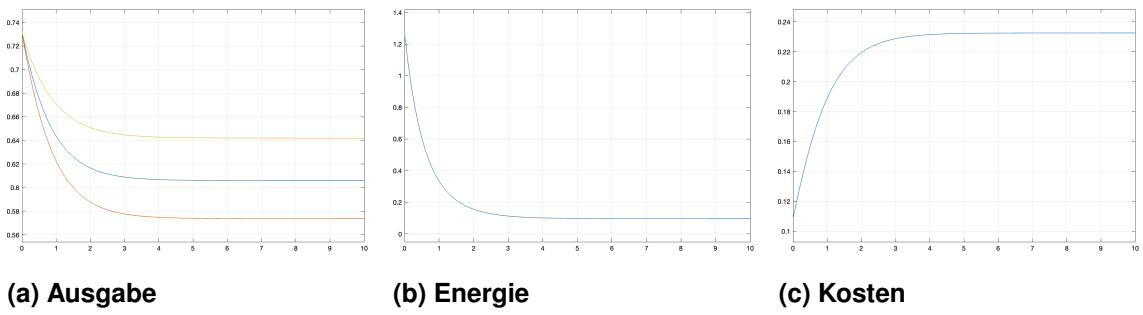
**Abbildung 10: Zweite Simulation des HNN**



Quelle: Eigene Darstellung

Die dritte Simulation wird mit  $\beta = 1$  durchgeführt, wodurch die Zustände des Netzwerks höhere Werte annehmen und somit die Kostenfunktion kleiner sein sollte. Dementsprechend sollte die Energiefunktion auch einen höheren Wert annehmen, da die Ausgabe des Netzwerks vom optimalen Ergebnis aus der ersten Simulation abweicht. Die Ergebnisse der Simulation (siehe Abbildung 11) bestätigen diese Annahmen.

**Abbildung 11: Dritte Simulation des HNN**



Quelle: Eigene Darstellung

## 4.2 Konstruktion des Equilibrium Propagation Algorithmus

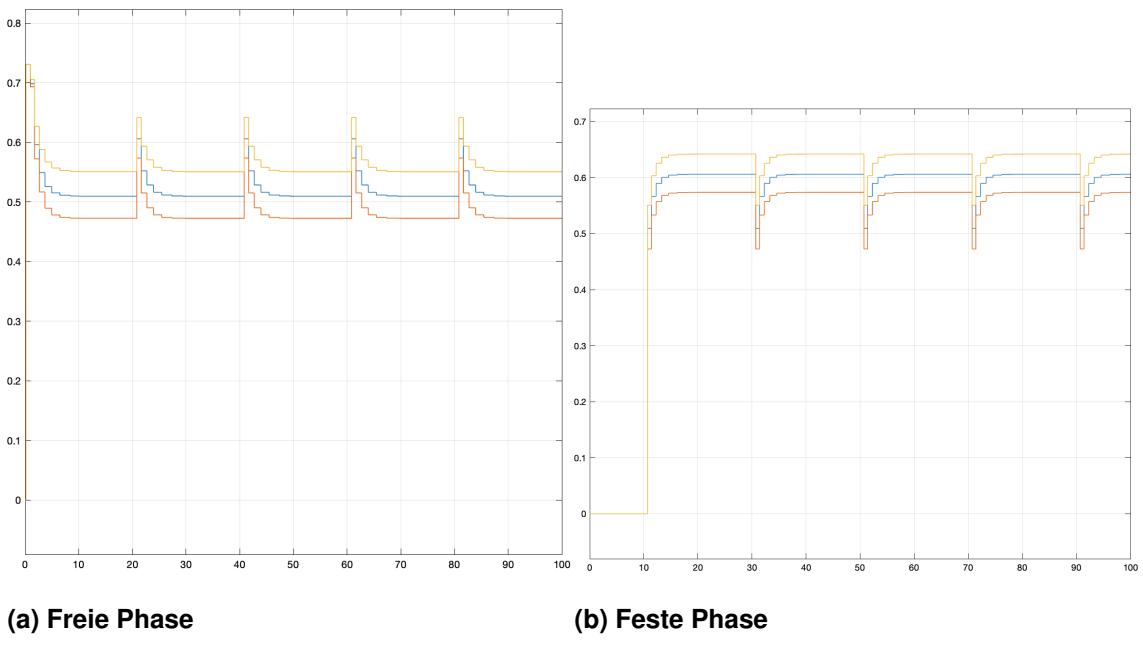
### 4.2.1 Umsetzung der theoretischen Modelle in der Simulationssoftware

Ein erster, naiver Ansatz zur Implementierung des EP beinhaltet die Aufteilung des zweiphasigen Lernprozesses in zeitlich getrennte Abläufe. Damit wird ein einziges HNN benötigt, um Inferenz mit  $\beta = 0$  und  $\beta > 0$  durchzuführen.

Die Integratoren zur Repräsentation der Gewichte müssen dafür zurücksetzbar sein, was in Simulink durch den Parameter „External reset“ am Integrator-Block möglich ist. Auf einem analogen Computer kann dies durch den Wechsel der Modi „IC“ bzw. „OP“ erfolgen (vgl. Kapitel 2.2.2). Der „Pulse Generator“-Block aus Simulink kann so konfiguriert werden, dass er innerhalb einer gesetzten Periodendauer zwischen den Werten 0 und 1 wechselt und somit die Zeitsteuerung abbildet. Dafür muss für die praktische Umsetzung ein externes Bauteil zum Einsatz kommen, da, basierend auf den in Kapitel 2.2.2 genannten Plattformen, Pulsgeneratoren nicht auf analogen Computern bereitgestellt werden. Als weitere Komponente wird der Komparator benötigt, welcher anhand des Signals des Pulsgenerators die aktive Phase des Lernprozesses bestimmt und in Simulink als „Switch“-Block abgebildet wird. Dieser Block findet Anwendung am Lernparameter  $\beta$ , am Zurücksetzen der Integratoren und als Schranke zur Weiterverarbeitung der Zustände des Netzwerks.

Durch die Zeitsteuerung des Lernprozesses ist zu jeder Zeit nur eine der beiden Phasen aktiv. Aus diesem Grund müssen die Zustände des Netzwerks nach der freien Phase zwischengespeichert werden, um sie nach Abschluss der festen Phase weiterverarbeiten zu können. Im hier vorgestellten Modell (siehe Anhang Anhang 2), ist dieser Mechanismus über den „Memory“-Block in Simulink zusammen mit einer Rückkopplung seiner Ausgabe zum vorgeschalteten Komparator gelöst. Die damit erzeugte Ausgabe aus Abbildung 12 zeigt die Wirksamkeit der Zeitsteuerung, da alle zwanzig Sekunden für beide Phasen ein Fixpunkt des Netzwerks gefunden wird. Aus den Graphen wird aber auch deutlich, dass die Simulation durch die Nutzung der „Memory“-Blöcke mit diskreten Werten arbeitet, wodurch sich dieser Ansatz nicht für die Umsetzung auf analogen Computern eignet. Die Dokumentation des THAT beschreibt zwar den „HALT“-Modus, wodurch die Integratoren ihren letzten Wert behalten und damit scheinbar für diesen Prozess genutzt werden können, dieser Modus soll aber laut der Dokumentation primär für Diagnostik und Fehlerbehebung genutzt werden u. a. aus dem Grund, dass die gespeicherten Werte über Zeit an Genauigkeit verlieren (GmbH, A., 2024b, vgl.).

**Abbildung 12: Ausgabe des zweiphasigen Lernprozesses des EP im ersten vorgestellten Ansatz**



Quelle: Eigene Darstellung

In einer Arbeit von *Kendall et al.* wurde bereits gezeigt, dass EP in einer analogen Simulation in Zusammenarbeit mit herkömmlicher Software implementiert werden kann. *Kendall et al.* stellte dafür einen Ansatz mithilfe der Simulationssoftware SPICE sowie Python vor, wobei innerhalb von SPICE das neuronale Netzwerk implementiert und damit die Fixpunkte der beiden Phasen gefunden wurden. Für die restlichen Bestandteile des Lernprozesses,

wie die Berechnung der Fehlerfunktion oder die Anwendung der Lernregel, kam Python als digitaler Co-Prozessor zum Einsatz (*Kendall, J.* et al., 2020, vgl. S. 27). Die Entwicklung eines hybriden Rechners ist aber nicht Teil dieser Arbeit, weshalb dieser Ansatz nicht weiter verfolgt wird.

Damit das EP auf analoger Hardware realisierbar wird, muss eine kontinuierliche Lernregel aufgestellt werden. Einen Ansatz hierfür bietet das 2020 von *Ernoult* et al. vorgestellte C-EP. Diese Abwandlung des EP bietet einerseits seine räumliche Lokalität, welche durch die Eigenschaft der Lernregel  $\Delta W_{ij}$ , nur mit den beiden Zuständen  $u_i$  und  $u_j$  auszukommen, gegeben ist (siehe Kapitel 2.3.3.2). Zusätzlich arbeitet C-EP lokal in Zeit, da der Zugriff auf das Ergebnis der freien Phase nach der festen Phase entfällt (*Ernoult, M.* et al., 2020, vgl. S. 3 f.). Im C-EP werden die Parameter  $\theta$  des Modells nicht durch eine Lernregel wie  $\Delta W$  aktualisiert, sondern ähnlich wie die Zustände des Modells, durch eine Dynamik beschrieben.

#### **Formel 20: Anwendung des C-EP auf ein RNN**

$$\theta_{t+1}^{\beta, \eta} = \theta_t^{\beta, \eta} + \frac{\eta}{\beta} \left( \frac{\partial \Phi}{\partial \theta}(x, s_{t+1}^{\beta, \eta}, \theta_t^{\beta, \eta}) - \frac{\partial \Phi}{\partial \theta}(x, s_t^{\beta, \eta}, \theta_t^{\beta, \eta}) \right) \quad (19)$$

Quelle: Ebd., S. 4

Angewandt auf ein RNN kann die Lernregel, wie von *Ernoult* et al. beschrieben, wie folgt aussehen:

#### **Formel 20: Anwendung des C-EP auf ein RNN**

$$\Delta_W^{C-EP}(\beta, \eta, t) = \frac{1}{\beta} (s_{t+1}^{\beta, \eta} \cdot s_{t+1}^{\beta, \eta T} - s_t^{\beta, \eta} \cdot s_t^{\beta, \eta T}) \quad (20)$$

Quelle: Ebd., S. 24

Auf Grundlage des C-EP stellte *Martin* et al. das Spike-driven Equilibrium Propagation (EqSpike), eine weitere Abwandlung des EP zur Anwendung auf neuromorphe Systeme, vor. Die Arbeit stellt neben dem angepassten Lernalgorithmus auch eine Dynamik  $\frac{dW_{ij}}{dt}$ , angepasst für das von *Scellier, Bengio* vorgestellte HNN, vor (*Martin, E.* et al., 2020, vgl. S. 3), mit derer hier weitergearbeitet wird.

#### **Formel 21: Kontinuierliche Anpassung der Gewichte eines RNN**

$$\frac{dW_{ij}}{dt} \propto \dot{\rho}_j \rho_i + \dot{\rho}_i \rho_j \quad (21)$$

Quelle: Ebd., S. 2

Zur Umsetzung dieses angepassten Lernprozesses kommt, wie im vorherigen Ansatz auch, ein zeitgesteuerter Pulsgenerator zum Einsatz. Der Einflussparameter  $\beta$  wird weiterhin über einen Komparator verbunden mit zwei Konstanten implementiert, zusätzlich wird nun die Lernrate  $\eta$  benötigt, welche einen konstanten Wert über den gesamten Lernprozess annimmt. Die Implementierung des HNN berechnet bereits die Aktivierungen der Neuronen bzw. die Änderungsraten dieser (siehe Kapitel 4.1.1), weshalb die entsprechenden Signale aus dem Netzwerk für die Berechnung von  $\frac{dW_{ij}}{dt}$  genutzt werden können.

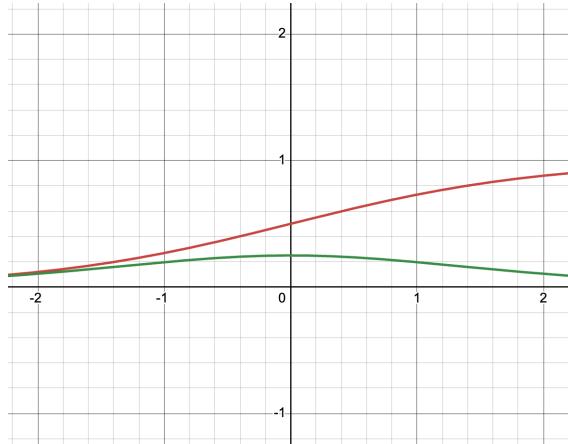
Damit die Integratoren zur Darstellung der Gewichte ihren aktualisierten Wert während der freien Phase behalten, kommt ein weiterer Komparator zum Einsatz, welcher während der freien Phase einen konstanten Wert 0 an die entsprechenden Integratoren weitergibt. Dieser Ansatz kann in der Praxis zu Problemen führen, da analoge Rechner mit sog. Leaky-Integratoren arbeiten, welche über Zeit ihren Zustand verlieren, mögliche Lösungen werden in Kapitel 5.2.1 beschrieben. Die Konstruktion des C-EP in Simulink wird im Anhang Anhang 2 gezeigt.

#### 4.2.2 Herausforderungen bei der Übernahme und Anpassung

Eine Herausforderung bei der Umsetzung des C-EP ist das Finden einer passenden Aktivierungsfunktion für die Zusammenarbeit mit der Lernregel  $\frac{dW_{ij}}{dt}$  und einer passenden Lernrate  $\eta$ , welche eine effiziente Konvergenz der Gewichte gewährleistet. Für die bisher gewählte Aktivierungsfunktion, die logistische Funktion wie dargestellt in Abbildung 13, gilt  $\rho(x) > 0$ , weshalb auch die Lernregel immer positiv ist. Die Auswirkungen dessen sind in Abbildung 14 dargestellt, wobei als Zielwert für das Netzwerk  $\vec{d} = (0, 0, 0)$  gewählt wurde. Die Ausgabe des Netzwerks richtet sich zu Beginn der zweiten Phase offensichtlich in Richtung der Zielwerte, steigt aber nach einer kurzen Zeit stetig an, da der positive Einfluss der Gewichte den negativen der Kostenfunktion übersteigt.

Die Dauer der beiden Phasen des C-EP wurde durch die Konvergenz dieser definiert. So soll die erste Phase durchgeführt werden bis das Netzwerk einen Fixpunkt findet, für die zweite Phase müssen Fixpunkte des Netzwerkes und der Parameter gefunden werden (Ernoult, M. et al., 2020, vgl. S. 3). Die Bestimmung sinnvoller Werte für die Dauer dieser Phasen stellt die Umsetzung des Lernprozesses vor eine weitere Herausforderung. Wie bereits in 4.1.3 gezeigt, propagiert das implementierte HNN mit beliebigem  $\beta$  zuverlässig innerhalb von weniger als zehn Zeiteinheiten zu einem Fixpunkt. Gleiches sollte auf die feste Phase zutreffen, dies ist aber, wie in Abbildung 15 gezeigt, nicht der Fall. Eine mögliche Ursache hierfür stellt erneut die Auswahl der Aktivierungsfunktion dar, da, wie aus Abbildung 13 abzulesen,  $\rho(x) > 0$  bzw.  $\dot{\rho}(x) > 0$  und somit  $\frac{dW_{ij}}{dt} > 0$  gilt. Eine Simulation

**Abbildung 13: Graph der Aktivierungsfunktion  $\rho(x) = \frac{1}{1+e^{-x}}$  (rot) und ihrer Ableitung (grün)**



Quelle: Eigene Darstellung

mit  $\vec{d} = (0.8, 0.8, 0.8)$  und  $\eta = 0.1$  zeigt die valide Anpassung der Gewichte bis zu einem Zeitpunkt  $t \approx 250$  aber die anschließende fehlende Konvergenz der Parameter.

Um Fehler bei der Auswahl der Aktivierungsfunktion oder der Übernahme der Lernregel auszuschließen, wird im Folgenden die von *Ernoult et al.* ursprünglich für C-EP vorgestellte Lernregel verwendet (siehe Formel „Anwendung des C-EP auf ein RNN“). Angewandt auf das hier genutzte HNN und unter Beachtung der Lernrate  $\eta$  ergibt sich:

### Formel 22,23: Anwendung des C-EP auf hier genutzte HNN

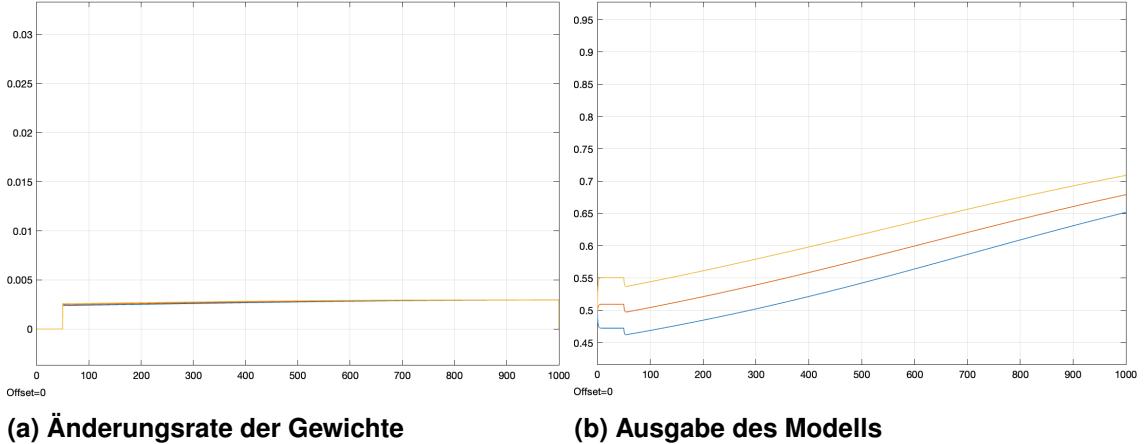
$$\Delta_{W_{ij}}^{C-EP}(\eta, t) \propto \eta \left( \rho(u_{t+1,i}^\beta) \rho(u_{t+1,j}^\beta) - \rho(u_{t,i}^\beta) \rho(u_{t,j}^\beta) \right) \quad (22)$$

$$\Delta_{b_i}^{C-EP}(\eta, t) \propto \eta \left( \rho(u_{t+1,i}^\beta) - \rho(u_{t,i}^\beta) \right) \quad (23)$$

Quelle: In Anlehnung an *Ernoult, M. et al.*, 2020, S. 24

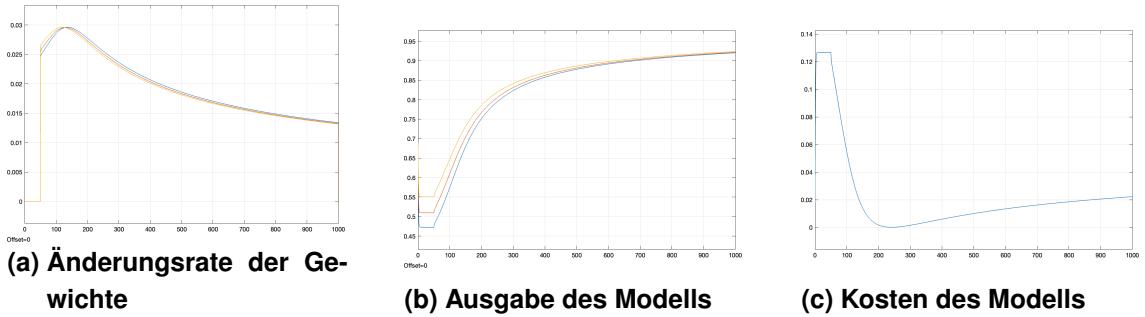
Anhand dieser Lernregel kann die Dynamik  $\frac{dW_{ij}}{dt}$  approximiert werden, indem praktisch mit einer Zeitverschiebung gearbeitet wird. Simulink bietet hierfür den Block „Transport Delay“, welcher ein kontinuierliches Eingangssignal um einen beliebigen Zeitraum verschiebt. Die Zeitverschiebung ist kein grundlegender Bestandteil herkömmlicher analoger Computer (siehe Kapitel 2.2.2) und für diese Aufgabe eignet sich auch wieder ein hybrider Ansatz, da Informationen (auch wenn nur über einen sehr kurzen Zeitraum) gespeichert werden müssen. Es existieren aber auch Ansätze zur Annäherung einer Zeitverschiebung auf analoger Hardware (*Ulmann, B.*, 2022, vgl. S. 117 ff.). Negative Gewichtsanpassungen sind mit diesem Ansatz ohne weiteres möglich und der beispielhafte Durchlauf des Netzwerks

**Abbildung 14: Auswirkungen einer nicht-negativen Aktivierungsfunktion auf das C-EP**



Quelle: Eigene Darstellung

**Abbildung 15: Die Parameter des Modells gelangen zu keinem Fixpunkt und verfehlten damit das Minimum der Kostenfunktion**



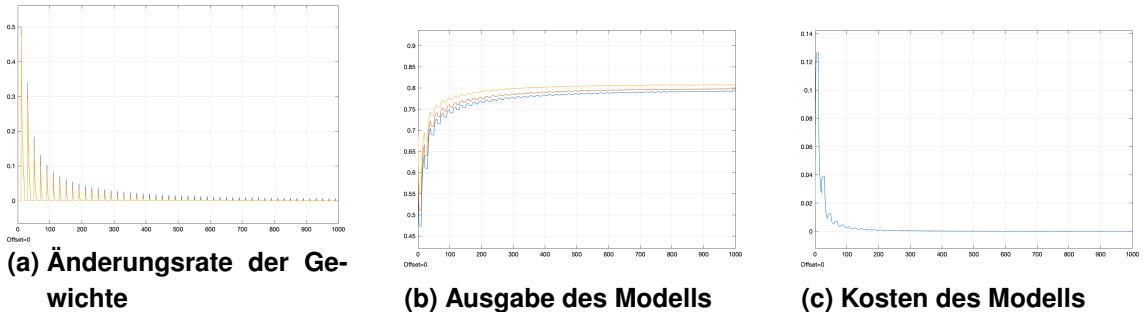
Quelle: Eigene Darstellung

mit  $\vec{d} = (0.8, 0.8, 0.8)$  (jetzt mit  $\eta = 10$ ) konvergiert mit diesem Ansatz zu sinnvollen Gewichten (siehe Abbildung 16). Im Anhang Anhang 2 wird das Simulink-Modell zu diesem Ansatz gezeigt.

#### 4.2.3 Strategien zur Fehlerbehebung und Optimierung

Im vorherigen Kapitel 4.2.2 wurde bereits gezeigt, dass die Implementierung des C-EP sinnvolle Gewichte für  $\vec{d} = (0.8, 0.8, 0.8)$  findet. Da die Aktivierungsfunktion in Annäherung an  $\rho(x) = 0$  bzw.  $\rho(x) = 1$  eher große Werte  $x$  benötigt ( $\rho(4.5952) \approx 0.99$ ), müssen die Zustände des Netzwerks bzw. die Gewichte entsprechend große Werte annehmen, um die Zielwerte 0 bzw. 1 abzubilden. Die Konvergenz der Gewichte ist für diese Zielwerte auch nicht möglich, weshalb das C-EP in diesem Fall die Gewichte endlos vergrößert

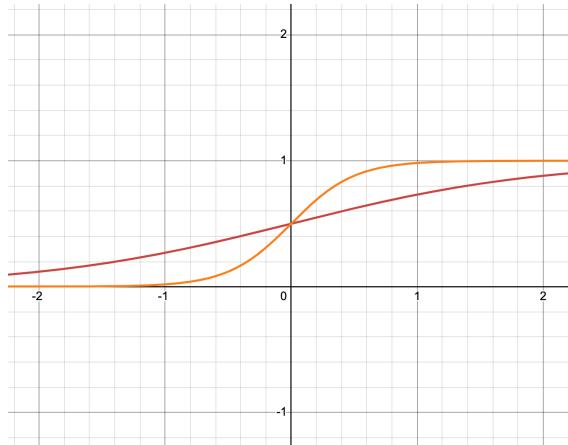
**Abbildung 16: Eine Annäherung an C-EP findet passende Parameter für das Netzwerk**



Quelle: Eigene Darstellung

bzw. verkleinert. Eine mögliche Lösung hierfür könnte das Austauschen der Aktivierungsfunktion gegen die sog. „ReLU“-Funktion ( $R(x) = \max(x, 0)$ ) sein, welche linear im Wertebereich  $x > 0$  verläuft. Beim Anwenden auf das HNN führt ReLU aber zu Problemen, da das Netzwerk damit nicht konvergiert und somit unbrauchbare Werte ausgibt. In Anlehnung an *Ernoult, M. et al., 2020*, vgl. S. 31 kann die bereits genutzte Aktivierungsfunktion  $\rho(x) = \frac{1}{1+e^{-x}}$  skaliert werden, um ihre S-Kurve zu stauchen und damit die Annäherung an 0 bzw. 1 zu vereinfachen. Die abgewandelte Funktion  $\rho(x) = \frac{1}{1+e^{-4x}}$  ist in Abbildung 17 dargestellt.

**Abbildung 17: Graph der Aktivierungsfunktion (rot) und der skalierten Variante  $\rho(x) = \frac{1}{1+e^{-4x}}$  (orange)**



Quelle: Eigene Darstellung

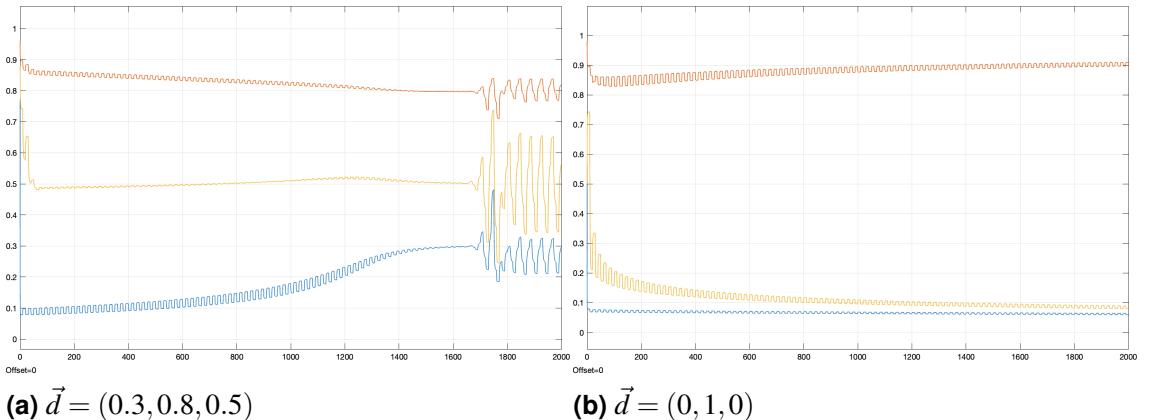
Mit den Parametern  $\beta = 1$  und  $\eta = 1$  findet das C-EP mit dieser skalierten Aktivierungsfunktion und den Zielwerten  $\vec{d} = (0, 0, 0)$  Gewichte, welche die Ausgabe  $\vec{y} \approx (0.14, 0.14, 0.16)$  erzeugen, damit zwar in die Nähe der Zielwerte gelangen aber diese noch immer drastisch verfehlten. Werden aber Zielwerte abseits der Grenzwerte 0 bzw. 1 beispielsweise  $\vec{d} = (0.2, 0.5, 0.8)$  betrachtet, so werden Gewichte gefunden, mit de-

nen ein Fehler von  $C(y) < 0.001$  erreicht wird. Ein weiterer Durchlauf mit den Zielwerten  $\vec{d} = (0.3, 0.4, 0.7)$  führt zu einem Fehler der Größenordnung  $10^{-6}$  in unter zehn Epochen. Aufgrund dieser Verbesserungen wird mit der skalierten Aktivierungsfunktion weitergearbeitet.

#### 4.2.4 Validierung des Algorithmus durch Testläufe

Zuerst werden einige Durchläufe des bereits implementierten Netzwerks mit drei Neuronen durchgeführt. Als Hyperparameter werden  $\beta = 1$  und  $\eta = 0.5$  definiert, jeweils die Ausgaben zweier Durchläufe des C-EP sind in Abbildung 18 dargestellt. Alle hier aufgeführten Simulationen werden mit 10 Zeiteinheiten für die freie und 10 für die feste Phase durchgeführt, was einer Dauer von 20 Zeiteinheiten pro Epoche entspricht. In Abbildung 18 sind zwei Durchläufe des Lernprozesses mit den genannten Hyperparametern dargestellt. Die Zielwerte  $\vec{d} = (0.3, 0.8, 0.5)$  findet das C-EP innerhalb von ca. 1700 Zeiteinheiten, was 85 Epochen entspricht. Kurz nachdem die Zielwerte erreicht und die Kostenfunktion damit minimiert wurde, wurden die Gewichtsanpassungen chaotisch. Die zweite Simulation mit  $\vec{d} = (0, 1, 0)$  zeigt, dass sich das Netzwerk an die Grenzwerte 0, 1 zwar annähert, diese aber nicht ohne größeren Fehler erreicht.

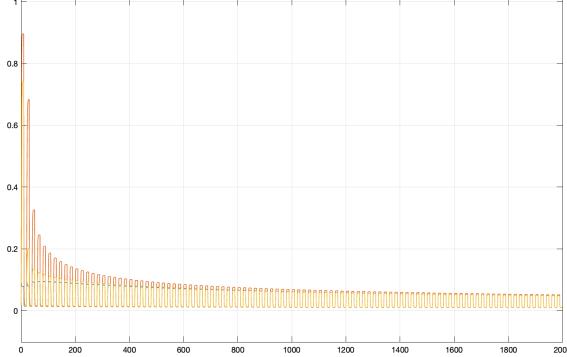
**Abbildung 18: Simulationen des C-EP mit  $\beta = 1, \eta = 0.5$ . Dargestellt ist die Ausgabe des Netzwerks**



Quelle: Eigene Darstellung

In Abbildung 19 ist diese Annäherung erneut dargestellt, diesmal mit den extremen Hyperparametern  $\eta = 10, \beta = 1$ . Zu sehen ist, dass sich das Netzwerk auf nahezu 0 annähert, dies aber in einer nicht trivialen Weise, da ein so groß gewähltes  $\eta$  die Konvergenz für Zielwerte  $0 < d < 1$  erschwert. Der Lernprozess wurde auch für  $\vec{d} = (-1, -1, -1)$  und  $\vec{d} = (2, 2, 2)$  durchgeführt, mit diesen Zielwerten nähert sich die Ausgabe des Netzwerk 0 bzw. 1.

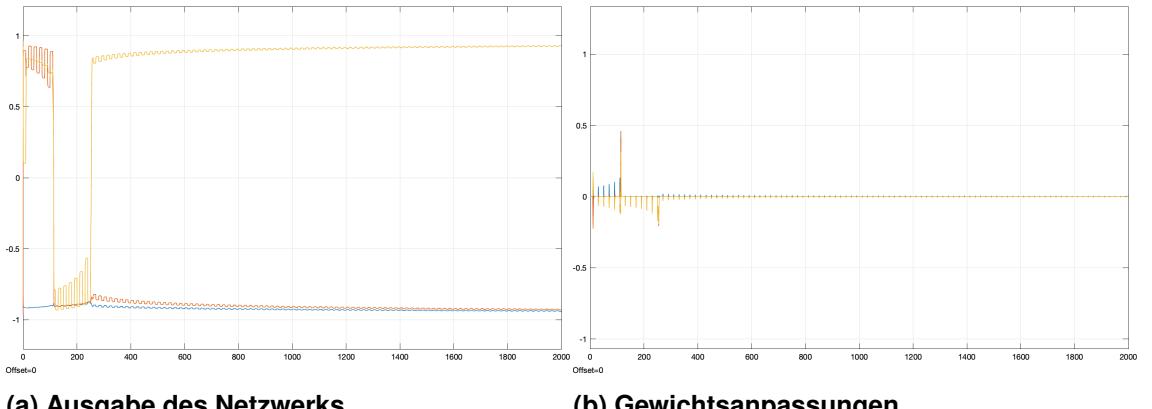
**Abbildung 19: Annäherungen des C-EP an den Grenzwert 0**



Quelle: Eigene Darstellung

Das C-EP wurde auch mit einer alternativen Aktivierungsfunktion  $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$  und den Zielwerten  $(-1, -1, 1)$  simuliert. Die Besonderheit dieser Funktion im Vergleich zur bisher genutzten logistischen Funktion ist der Wertebereich  $[-1; 1]$ . Damit lassen sich, wie in Abbildung 20 gezeigt, auch negative Zielwerte mit einem Fehler von  $C(y) \approx 0.0067$  abbilden, wodurch andere Anwendungsfälle bedient werden können. Die gezeigten Simulationen wurden mit einer skalierten Variante der genannten Funktion durchgeführt  $\rho(x) = \tanh(2x)$ .

**Abbildung 20: Simulationen des C-EP mit der Aktivierungsfunktion:  $\rho(x) = \tanh(2x)$**

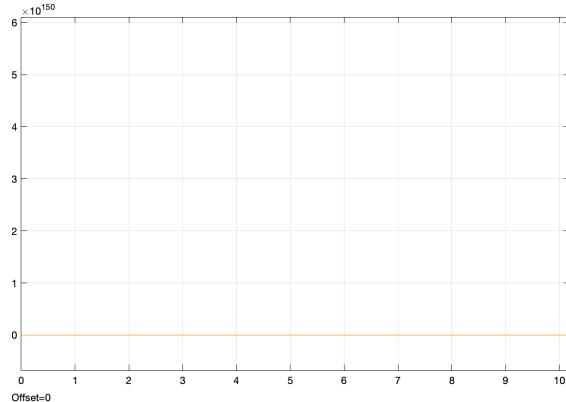


Quelle: Eigene Darstellung

Wie bereits im Kapitel 4.2.3 beschrieben, führt die Aktivierungsfunktion „ReLU“ im Zusammenhang mit dem hier implementierten HNN zu unerwarteten Fehlern. Der schlagartige Anstieg der Zustände des Netzwerks ist in Abbildung 21 dargestellt.

Ein fehlerfreies Speichern von  $N$  Mustern im HNN erfordert eine Mindestanzahl von  $\frac{N}{0.15}$  Neuronen, was einer Anzahl von 14 Neuronen zum Speichern von zwei Mustern, 20 für drei Muster usw. entspricht (Hopfield, J. J., 1982, vgl. S. 2556). In dieser Arbeit konnte

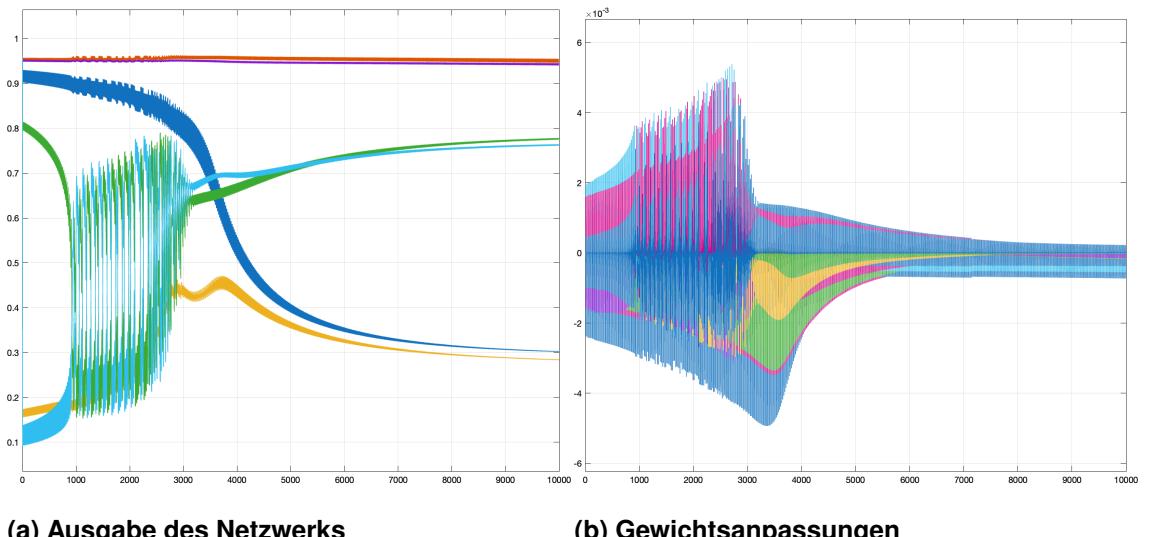
**Abbildung 21: Simulation des C-EP mit der Aktivierungsfunktion „ReLU“ ( $R(x) = \max(x, 0)$ ). Gezeigt wird die Dynamik der Zustände des Netzwerks  $\frac{ds}{dt}$**



Quelle: Eigene Darstellung

bisher gezeigt werden, dass ein einziges Muster mit einem Netzwerk aus drei Neuronen mit einem Fehler unter 0.1 gespeichert werden kann. Die Konstruktion eines Netzwerks mit 14 (oder mehr) Neuronen erfordert aber eine überproportional komplexe Suche nach Hyperparametern für den Lernprozess, in Verbindung mit dem erforderlichen Rechenaufwand ist eine größere Simulation sehr unattraktiv. Aus diesem Grund wurde das Modell nur auf eine Größe von sechs Neuronen erweitert und so auf die Zielwerte  $\vec{d} = (0.3, 0.6, 0.3, 0.8, 0.8, 0.8)$  trainiert, siehe Abbildung 22. Die für die Simulation gefundenen Hyperparameter lauten  $\eta = 0.05, \beta = 0.5$ , die Simulation wurde für 10.000 Zeiteinheiten, also 500 Epochen, durchgeführt. Damit erreicht das Netzwerk einen Fehler von  $C(y) \approx 0.0709$ .

**Abbildung 22: Simulationen des C-EP mit 6 Neuronen**



Quelle: Eigene Darstellung

## 5 Diskussion

In diesem Kapitel werden die Ergebnisse der Implementierung und Simulation des EP kritisch reflektiert. Es erfolgt eine Bewertung der gewählten Methodik und Simulationsumgebung hinsichtlich ihrer Effektivität und Praxistauglichkeit. Die technischen und konzeptuellen Herausforderungen bei der Umsetzung werden analysiert, einschließlich möglicher Abweichungen von theoretischen Erwartungen. Zudem wird das Potenzial des EP in analogen Rechensystemen diskutiert, auch im Vergleich zu digitalen Implementierungen. Die Forschungsergebnisse werden im Kontext bestehender Arbeiten kritisch eingeordnet, um mögliche Implikationen für zukünftige Entwicklungen aufzuzeigen.

### 5.1 Reflexion der Konstruktionsmethodik

#### 5.1.1 Bewertung der Simulationsumgebung und -werkzeuge

Die bereits von *Ulmann* empfohlene Software Simulink stellt alle nötigen Komponenten bereit, um analoge Schaltungen simulieren zu können (*Ulmann, B.*, 2022, vgl. S. 240). Darüber hinaus bietet Simulink auch eine Vielzahl weiterer Komponenten an, welche kein Äquivalent auf analoger Hardware besitzen, so z. B. der „Memory“-Block. Es muss also beachtet werden, welche Komponenten genutzt werden können, um aus dem Modell später eine analoge Schaltung ableiten zu können. Grundsätzlich kann sich aber frei an den Bibliotheken „Math Operations“ und „Continuous“ bedient werden. Es muss auch die Abweichung der Blöcke aus Simulink von den analogen Rechenelementen beachtet werden. So fehlt z. B. den Summierern aus Simulink der für Operationsverstärker typische Vorzeichenwechsel (vgl. Kapitel 2.2.2) und die Blöcke aus Simulink geben grundsätzlich eine fehler- und störungsfreie Ausgabe, was nicht den realen Bedingungen entspricht.

Als Erweiterung der Software Matlab hat Simulink den Vorteil der Integration weiterer Komponenten, wie dem Matlab-Funktionsblock, siehe Kapitel 3.2. Dieser Block hat in dieser Arbeit einige Anwendungsfälle gefunden, da somit z. B. die Funktionsweise des HNN validiert oder die Gewichte zu einer Matrix umgewandelt und somit zur Verwendung im HNN vorbereitet werden konnten. Die Fähigkeit von Simulink, mit Vektoren und Matrizen zu arbeiten, hat sich hier auch bewährt, da somit nur ein einziges Neuron des HNN modelliert werden muss, woraus ein Netzwerk mit beliebig vielen Neuronen erzeugt werden kann.

Ein für diese Arbeit schwerwiegendes Problem mit Simulink ist die Performance der Software in Verbindung mit dem Betriebssystem macOS. Die Konstruktion größerer Schaltungen (siehe z. B. Anhang Anhang 2) führt zu einer trügen Benutzererfahrung und zieht

somit den Prozess in die Länge. Bei der Ausführung komplexer Simulationen, wie hier ein Netzwerk mit mehr als drei Neuronen (siehe Kapitel 4.2.4), kommt das System auch an seine Grenzen, da diese bereits einige Sekunden bis wenige Minuten dauern. Das erschwert z. B. eine effiziente Suche nach Hyperparametern zum Trainieren des Netzwerks, da diese bei steigender Komplexität des Netzwerks präzise gewählt werden müssen.

### 5.1.2 Effektivität der Modifikationen am Netzwerk

## 5.2 Diskussion der Herausforderungen

### 5.2.1 Technische Herausforderungen und Lösungsansätze

Zur Umsetzung der Gewichte wurden bisher Integratoren genutzt, welche während der festen Phase mit einem konstanten Eingabewert 0 verschaltet werden und somit ihren Wert behalten. Nach *Ullmann* ist die Nutzung von Integratoren als temporäre Speicherzellen auch ein gängiges Vorgehen (*Ullmann, B.*, 2022, vgl. S. 92), bei der Anwendung auf größere Netzwerke und damit längeren Inferenz-Phasen kann dieser Ansatz praktisch aber anhand der Natur von Kondensatoren, durch Leckströme ihre Ladung zu verlieren, zu Problemen führen. Wie in Kapitel 3.3 beschrieben, befassten sich bereits einige Arbeiten mit der Implementierung eines HNN mit analogen Bauteilen. Eine häufig auftretende Bauweise beinhaltet den Memristor als zentrale Komponente, weshalb dieser auch hier zum Einsatz kommen könnte. So kann möglicherweise der akkumulierte Wert der Integratoren genutzt werden, um Memristoren zu programmieren, welche die Gewichte für die freie Phase speichern.

Eine Herausforderung bei der Konstruktion analoger Computer ist die notwendige Skalierung der einzelnen Werte, da diese typischerweise im Wertebereich  $[-1; 1]$  liegen müssen (siehe Kapitel 2.2.2). Die Aktivierungsfunktion wurde unter diesen Voraussetzungen bereits passend gewählt, die Zustände des Netzwerks bzw. die Gewichte überschreiten dieses Limit aber. Um die Zustände auf  $[-1; 1]$  zu limitieren, muss eine Aktivierungsfunktion gewählt werden, welche im Bereich  $-1 \leq x \leq 1$  in der Lage ist, alle Zielwerte abzubilden. Die im Kapitel 4.2.3 gezeigte ReLU Funktion könnte dieses Problem für Zielwerte im Bereich  $[0; 1]$  lösen. Die Gewichte könnten z. B. um einen Faktor 2 skaliert und die Werte der Integratoren dadurch auf  $[-1; 1]$  beschränkt werden. In der Umsetzung muss dann nur noch sichergestellt werden, dass das Ergebnis der Multiplikation der Gewichte mit den Zuständen den Wertebereich  $[-1; 1]$  nicht überschreitet.

Das in dieser Arbeit implementierte C-EP wurde auf nicht mehr als einen Zielwert pro Durchlauf angewandt. Soll das HNN aber eine praktische Aufgabe lösen, muss im Lernprozess über eine Vielzahl an Zielwerten iteriert werden. Mithilfe des „MNIST“ Datensatzes, bestehend aus 60.000 Bildern zu je 28x28 Pixeln, könnte das Netzwerk z. B. auf die Erkennung handgeschriebener Ziffern trainiert werden (*Deng, L., 2012*). Hierfür benötigt es einen Prozess zur effizienten Eingabe der Zielwerte, welcher im einfachsten Fall durch einen hybriden Ansatz gelöst wird. Der Lernprozess pro Iteration könnte so aber weiterhin vollständig auf der analogen Recheneinheit durchgeführt werden.

### 5.2.2 Umgang mit unerwarteten Ergebnissen

Die ursprüngliche Arbeit über EP beschreibt neben der Lernregel  $\Delta W_{ij}$  auch eine kontinuierliche Interpretation dieser im Zusammenhang mit der sog. Spike-Timing Dependent Plasticity *Scellier, B., Bengio, Y., 2017*. Dafür wird  $\Delta W_{ij}$  als kontinuierliche Dynamik der Gewichte beschrieben, worauf die Konstruktion des EP in dieser Arbeit aufbaut. Zusammen mit dem EqSpike stellte *Martin et al.* eine Lernregel  $\frac{dW_{ij}}{dt}$  auf, welche als Variation des EP lokal in Ort und Zeit arbeitet und sich somit für analoge Computer eignet. Die Modellierung dieser Lernregel in Simulink führt aber, wie in Kapitel 4.2.1 gezeigt, zu Problemen, da die Gewichte nicht konvergieren und die Ergebnisse damit unbrauchbar werden. Aufgrund dessen wurde auf diese kontinuierliche Lernregel verzichtet, und das diskrete Modell, vorgestellt von *Ernoult et al.* in ihrer Arbeit über C-EP, verwendet.

Diese Annäherung hat sich, wie in Kapitel 4.2.4 gezeigt, für Netzwerke mit wenigen Neuronen als funktionsfähig erwiesen, kann aber bei größeren Modellen an Genauigkeit verlieren. Möglicherweise kann die Genauigkeit durch eine Verringerung der Zeitdifferenz der verglichenen Zustände des Modells mit  $t$  und  $t + 1$  verbessert werden, eine mathematisch fehlerfreie Lösung ist aber nur mit einer Lösung von  $\frac{dW_{ij}}{dt}$  für die von *Scellier, Bengio* beschrieben Dynamik des HNN möglich. Da die aktuelle Forschung dafür aber noch keine Lösung bereithält, bleibt dieses Problem hier ungelöst.

## 5.3 Potenziale des Equilibrium Propagation in analogen Systemen

Einige Arbeiten befassen sich bereits mit dem Trainieren neuronaler Netze auf analoger Hardware. So finden Algorithmen wie das „Stochastic Hamiltonian Descent“ (*Onen, M. et al., 2022*) oder „Analog Gradient Accumulation with Dynamic Reference“ (*Rasch, M. J. et al., 2024*) Anwendung auf asymmetrische Netzwerke, implementiert auf resistiven

Crossbar-Arrays (ähnlich z. B. dem lucidac, siehe Kapitel 2.2.2), benötigen aber Informationen über das gesamte Netzwerk und vorheriger Zustände dessen. Nennenswert sind auch das „Multiplexed Gradient Descent“ (*McCaughan, A. N. et al., 2023*) sowie das „Physical Neuronal Network“ (*Sakemi, Y. et al., 2024*), denen aber die biologische Plausibilität des EP fehlt.

Als Alternative zum Backpropagation Through Time (BPTT) und Annäherung an die Funktionsweise biologischer Neuronen findet das C-EP Anwendung an energiebasierten Modellen (*Ernoult, M. et al., 2020*). Diese Arbeit befasst sich in Bezug auf das BPTT mit einem unpassenden Netzwerk, da sich das HNN durch eine einzige Ebene an Neuronen auszeichnet und dadurch die räumliche Lokalität des C-EP nicht vollkommen ausnutzen kann. Das C-EP hat Potenzial für die Anwendung auf ein mehrlagiges HNN oder MLP, da hier die Schwierigkeit in der Anwendung des Fehlers des Netzwerks über mehrere Ebenen liegt (vgl. Kapitel 2.1.2). Wie *Martin, E. et al., 2020* auch zeigte, eignet sich EP als Basis für das Trainieren von Spiking neural network (SNN) und ist damit eine vielversprechende Alternative zu herkömmlichen Gradienten-basierten Methoden, insbesondere für energieeffiziente und hardwarefreundliche Implementierungen in neuromorphen Systemen.

## 5.4 Kritische Bewertung der Forschungsergebnisse

### 5.4.1 Zusammenfassung der Erkenntnisse

Das C-EP konnte als Abwandlung des EP in der Simulationsumgebung umgesetzt werden, wodurch das C-EP als analoger Computer realisierbar ist. Der zweiphasige Lernprozess wurde dafür korrekt abgebildet, und es konnten stabile Fixpunkte im HNN gefunden werden. Die kontinuierliche Gewichtsanpassung durch C-EP wurde als umsetzbar erwiesen, wobei die Anpassung der Gewichte über die Zeit hinweg durch die Simulation nachvollzogen werden konnte.

### 5.4.2 Vergleich mit den Erwartungen und Zielsetzung

Da für die kontinuierliche Gewichtsanpassung im C-EP eine Differentialfunktion notwendig ist, welche für die von *Scellier, Bengio* vorgestellte Energiefunktion nicht aufgestellt wurde, wurde als Alternative eine Annäherung an diesen Lernprozess mithilfe diskreter Zeitschritte versucht. Dies erfolgte durch eine zeitliche Verschiebung der Ausgabe des Netzwerks, um eine kontinuierliche Dynamik näherungsweise zu simulieren. Zur vollständigen Umsetzung des C-EP ist jedoch eine mathematische Lösung erforderlich, die eine exakte

Differentialgleichung für die Gewichtsanpassung liefert. Da in der aktuellen Literatur keine entsprechende Formulierung gefunden wurde, bleibt dieses Problem ungelöst.

Neben der fehlenden mathematischen Definition für die kontinuierliche Gewichtsanpassung zeigte sich eine weitere Herausforderung in der Wahl einer geeigneten Aktivierungsfunktion. Während in der Simulation eine Sigmoid-Funktion verwendet wurde, bleibt offen, ob diese die optimale Wahl für eine physikalische Implementierung darstellt. Eine geeignete Aktivierungsfunktion muss sowohl die Eigenschaften des Netzwerks als auch die physikalischen Grenzen analoger Rechner berücksichtigen und sollte eine effiziente Verarbeitung der Trainingsdaten ermöglichen.

Auch die Konvergenzgeschwindigkeit stellte eine relevante Fragestellung dar. Zwar konnten Fixpunkte des Netzwerks erreicht werden, jedoch variierte die Geschwindigkeit der Anpassung in Abhängigkeit von den gewählten Parametern. Zu hohe Lernraten führten zu Instabilitäten, während zu niedrige Lernraten den Lernprozess erheblich verlangsamt. Eine optimale Wahl der Lernrate sowie der Dauer der beiden Phasen ist entscheidend für die praktische Anwendbarkeit des C-EP auf analoge Hardware.

Schließlich bleibt die Frage nach der Skalierbarkeit offen. Während die Implementierung in einem einfachen HNN funktionierte, ist unklar, wie sich das C-EP auf tiefere Netzwerke übertragen lässt. Analoge Hardware könnte dabei sowohl Einschränkungen als auch Vorteile bieten, die noch weiter untersucht werden müssen.

## 6 Zusammenfassung

Im Folgenden wird die Forschungsfrage reflektiert und beantwortet, indem die wesentlichen Erkenntnisse aus der Implementierung und Simulation des EP auf analogen Computern zusammengefasst werden. Es erfolgt eine Bewertung der methodischen Vorgehensweise sowie eine Analyse der identifizierten Herausforderungen und deren Bewältigung. Abschließend werden Potenziale für zukünftige Forschungsarbeiten aufgezeigt und Empfehlungen für weiterführende Untersuchungen vorgestellt.

### 6.1 Beantwortung der Forschungsfrage

Die Forschungsfrage dieser Arbeit lautet: Wie kann der EP Algorithmus effektiv auf analogen Computern implementiert werden und welche spezifischen Herausforderungen ergeben sich aus dieser Umsetzung?

Die durchgeführte Konstruktion zeigt, dass das C-EP als Abwandlung des EP grundsätzlich als analoger Rechner realisierbar ist. In der Simulationsumgebung kann der zweiphasige Lernprozess erfolgreich nachgebildet werden, wobei stabile Fixpunkte im HNN erreicht werden. Die kontinuierliche Anpassung der Gewichte erweist sich als umsetzbar, obwohl sie eine diskrete Annäherung erfordert, da eine mathematische Beschreibung der Differentialgleichung für die Gewichtsanpassung bislang nicht vorliegt.

Herausforderungen ergeben sich insbesondere bei der Wahl der Aktivierungsfunktion. Während die Sigmoid-Funktion in der Simulation verwendet wurde, bleibt offen, ob sie sich für eine physikalische Implementierung optimal eignet. Zudem beeinflusst die Wahl des Einflussparameters  $\beta$  und der Lernrate  $\eta$  maßgeblich die Stabilität und Geschwindigkeit der Konvergenz, wobei zu hohe Werte Instabilitäten verursacht und zu niedrige die Effizienz reduziert.

Die Skalierbarkeit des Ansatzes auf tiefere Netzwerke bleibt eine offene Fragestellung. Während die Implementierung für ein einfaches HNN funktioniert, sind weitere Untersuchungen erforderlich, um die Tragfähigkeit des C-EP für größere Netzwerke zu bewerten. Analoge Hardware könnte dabei sowohl Einschränkungen als auch Vorteile bieten, die noch nicht abschließend erforscht wurden.

### 6.2 Bewertung der methodischen Vorgehensweise

Die methodische Vorgehensweise dieser Arbeit basiert auf der Konstruktion eines Simulationsmodells zur Untersuchung der Implementierung des EP Algorithmus auf analogen

Computern. Die gewählte Simulationsumgebung erweist sich als geeignet, um die theoretischen Konzepte funktional umzusetzen und ermöglicht eine Analyse der Netzwerkodynamik sowie der kontinuierlichen Gewichtsanpassung. Es zeigen sich Herausforderungen bei der Modellierung von realistischen analogen Systemen, insbesondere durch Abweichungen zwischen Simulations- und Hardwareeigenschaften. Die Wahl der Aktivierungsfunktion und der Hyperparameter des Lernprozesses beeinflussen die Konvergenz und Stabilität des Modells. Trotz dieser Einschränkungen bietet die methodische Vorgehensweise eine fundierte Grundlage für die Bewertung des Ansatzes und liefert Erkenntnisse für zukünftige Implementierungen.

### 6.3 Herausforderungen und Lösungsansätze

Die Implementierung des C-EP Algorithmus auf analogen Computern bringt eine Reihe technischer und konzeptioneller Herausforderungen mit sich. Ein Problem stellt die Wahl einer geeigneten Aktivierungsfunktion dar. Die ursprünglich eingesetzte Sigmoid-Funktion nähert sich den Grenzwerten 0 bzw. 1, was zu einer fehlerhaften Darstellung dieser Werte im Netzwerk führt. Als Alternative kann die tanh-Funktion zum Einsatz kommen, welche zwar 0 abbilden kann, das Problem aber zu  $-1$  verlagert. Die ReLU-Funktion konnte nicht erfolgreich in das HNN integriert werden, sodass dieses Problem vorerst ungelöst bleibt.

Ein weiteres Problem ergibt sich aus der kontinuierlichen Natur analoger Systeme. Während digitale Systeme mit diskreten Werten und stabilen Speicherzellen arbeiten, nutzen analoge Rechner kontinuierlichen Signale. Insbesondere die Speicherung der Gewichte stellt eine Herausforderung dar, da herkömmliche Integratoren mit der Zeit ihre Werte verlieren. Eine potenzielle Lösung wäre der Einsatz von Memristoren, welche sich bereits zur Umsetzung von Gewichten in neuronalen Netzen bewährt haben.

Zusätzlich gestaltet sich die Skalierung des C-EP auf größere Netzwerke schwierig. Während das Verfahren für einfache HNNs funktioniert, bleibt unklar, wie es sich auf tiefere Strukturen übertragen lässt. Analoge Systeme bringen sowohl Einschränkungen als auch Potenziale mit sich, die eine weiterführende Untersuchung erfordern. Auch die Wahl der Hyperparameter wie der Lernrate und des Einflussparameters hat einen entscheidenden Einfluss auf die Stabilität und Konvergenz des Algorithmus. Zu hohe Werte können zu Instabilitäten führen, während zu niedrige Werte den Lernprozess verlangsamen.

## 6.4 Empfehlungen für zukünftige Forschung

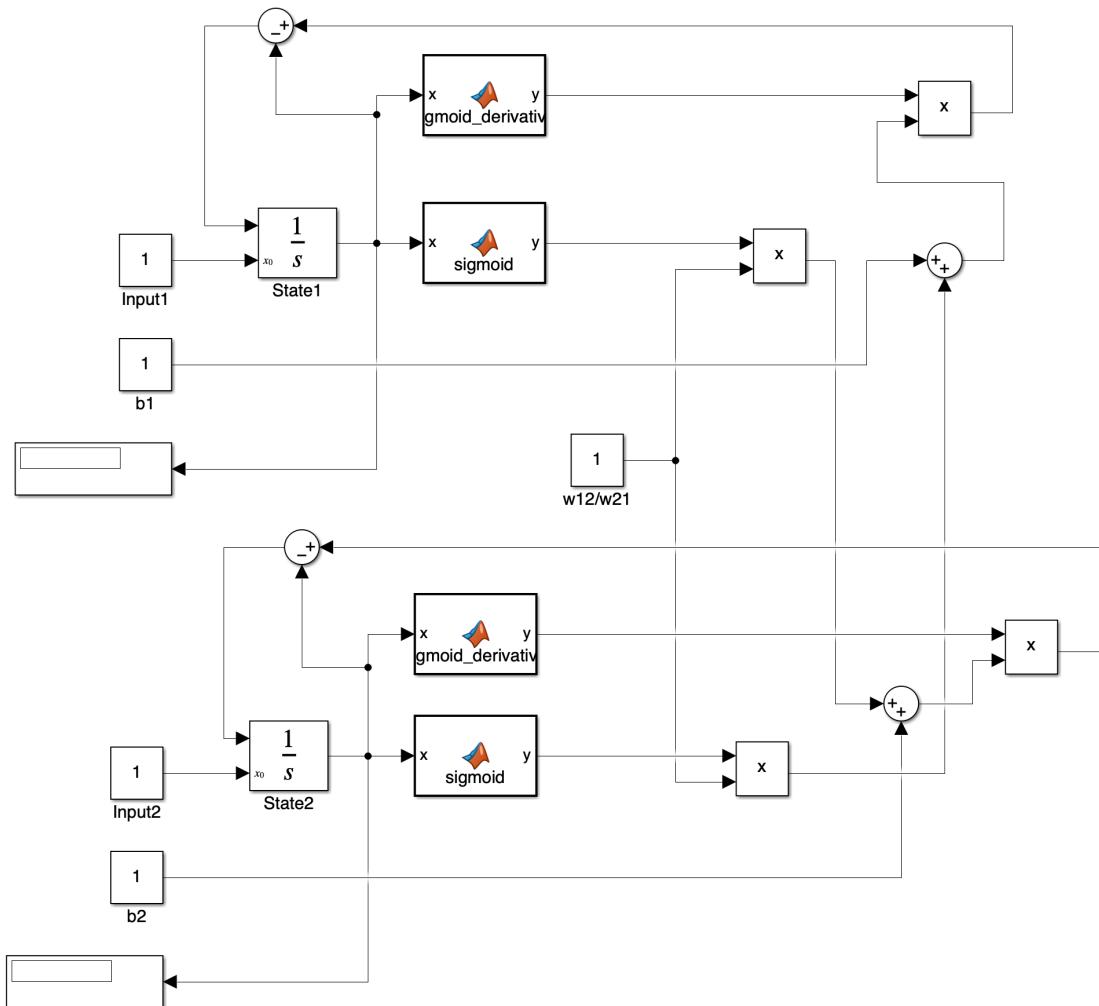
Zukünftige Arbeiten können sich mit der Skalierung des C-EP auf tiefere Netzwerke befassen, z. B. auf mehrschichtige HNNs oder MLP als Alternative zu BPTT. Die mathematische Beschreibung der kontinuierlichen Gewichtsanpassung bleibt offen und erfordert eine Formulierung der zugrundeliegenden Differentialgleichungen. Zudem ist die Wahl der Aktivierungsfunktion weiter zu untersuchen, um die physikalischen Eigenschaften analoger Rechner optimal zu nutzen. Eine experimentelle Implementierung auf analoger Hardware könnte Aufschluss über Stabilitäts- und Konvergenzverhalten geben und den Energieverbrauch realistisch bewerten.

## Anhang

### Anhang 1: Umsetzung des Hopfield-Netzwerks in Simulink

Ein HNN mit zwei Neuronen wurde nach der von *Scellier, B., Bengio, Y., 2017* vorgestellten Dynamik in Simulink modelliert und ist in Abbildung 23 dargestellt.

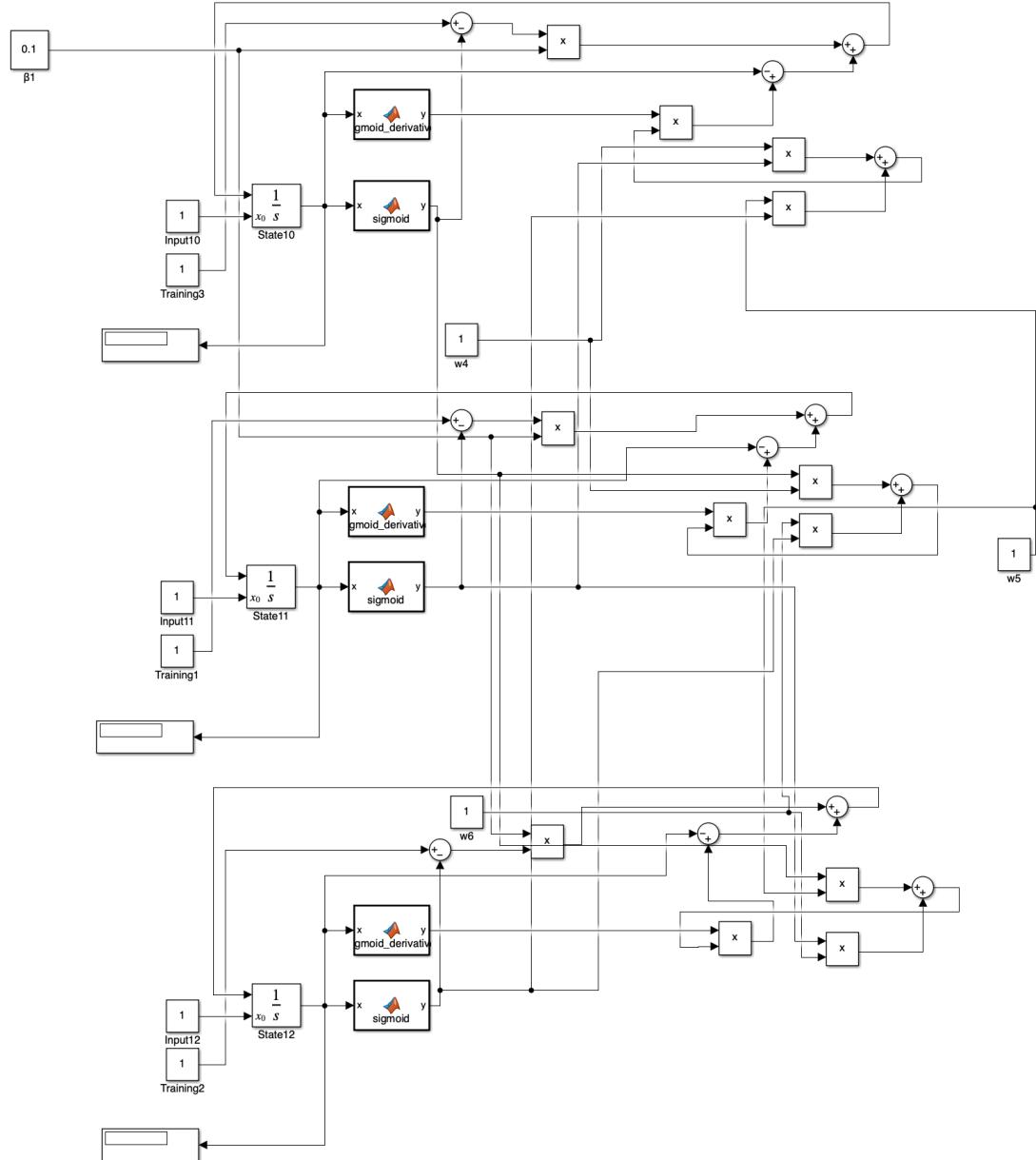
**Abbildung 23: Umsetzung eines HNNs mit zwei Neuronen in Simulink**



Quelle: Eigene Darstellung

Ein auf drei Neuronen skaliertes Modell ist in Abbildung 24 dargestellt. Hier wurde auch die Kostenfunktion des EP umgesetzt, nur der Bias wurde zur Übersichtlichkeit weggelassen.

Abbildung 24: Umsetzung eines HNNs mit drei Neuronen in Simulink

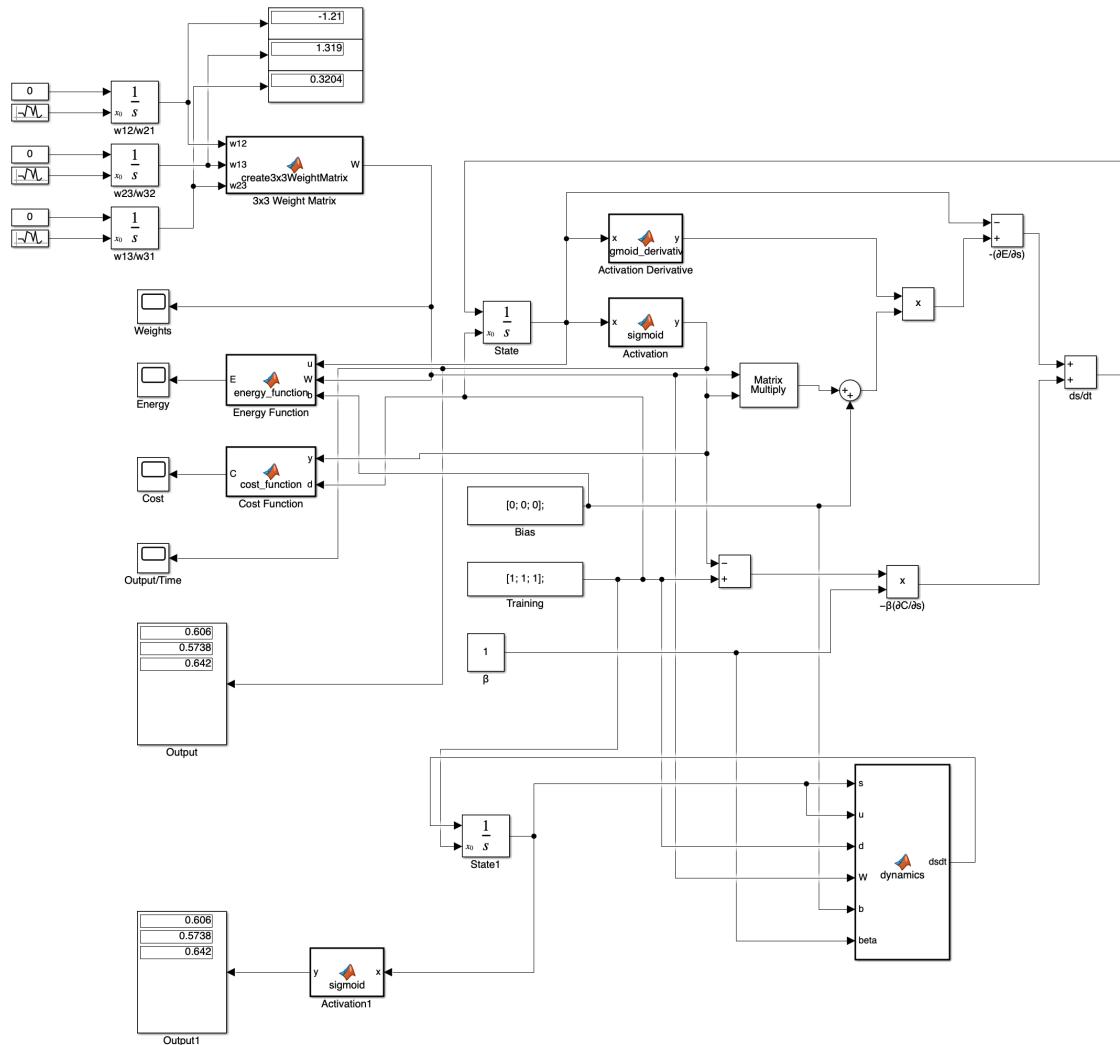


Quelle: Eigene Darstellung

Zur Vereinfachung der Arbeit mit dem HNN in Simulink wurde dieses für die Nutzung von Vektoren und Matrizen umgeformt, wie in Abbildung 25 dargestellt. Die Gewichte sind als Integratoren für skalare Werte vorhanden und werden über einen Matlab-Funktionsblock zu einer Gewichtsmatrix umgeformt.

Anhand des Funktionsblocks „dynamics“ unterhalb des Netzwerks kann die Funktionsweise dessen validiert werden. Da die Ausgaben „Output“ und „Output1“ übereinstimmen, wird gezeigt, dass das HNN eine korrekte Analogie zur Dynamik  $\frac{ds}{dt}$  darstellt.

**Abbildung 25: Finales Modell des HNN in Simulink mit einem Funktionsblock zur Validierung der Funktionsweise**



Quelle: Eigene Darstellung

Es folgt der Quellcode für die Funktionsblöcke „dynamics“, „sigmoid“ und „sigmoid derivative“ sowie zur Erstellung der Gewichtsmatrix.

```

1 function dsdt = dynamics(s, u, d, W, b, beta)
2 rho      = @(x) 1./(1 + exp(-x));
3 rho_prime = @(x) rho(x) .* (1 - rho(x));
4 W_eff = W - diag(diag(W));
5 energy_term = rho_prime(s) .* (W_eff * rho(u) + b);
6 cost_term = beta * (d - rho(u));
7 dsdt = energy_term - s + cost_term;
8 end

1 function y = sigmoid(x)
2 y = 1 ./ (1 + exp(-x));
3 end

```

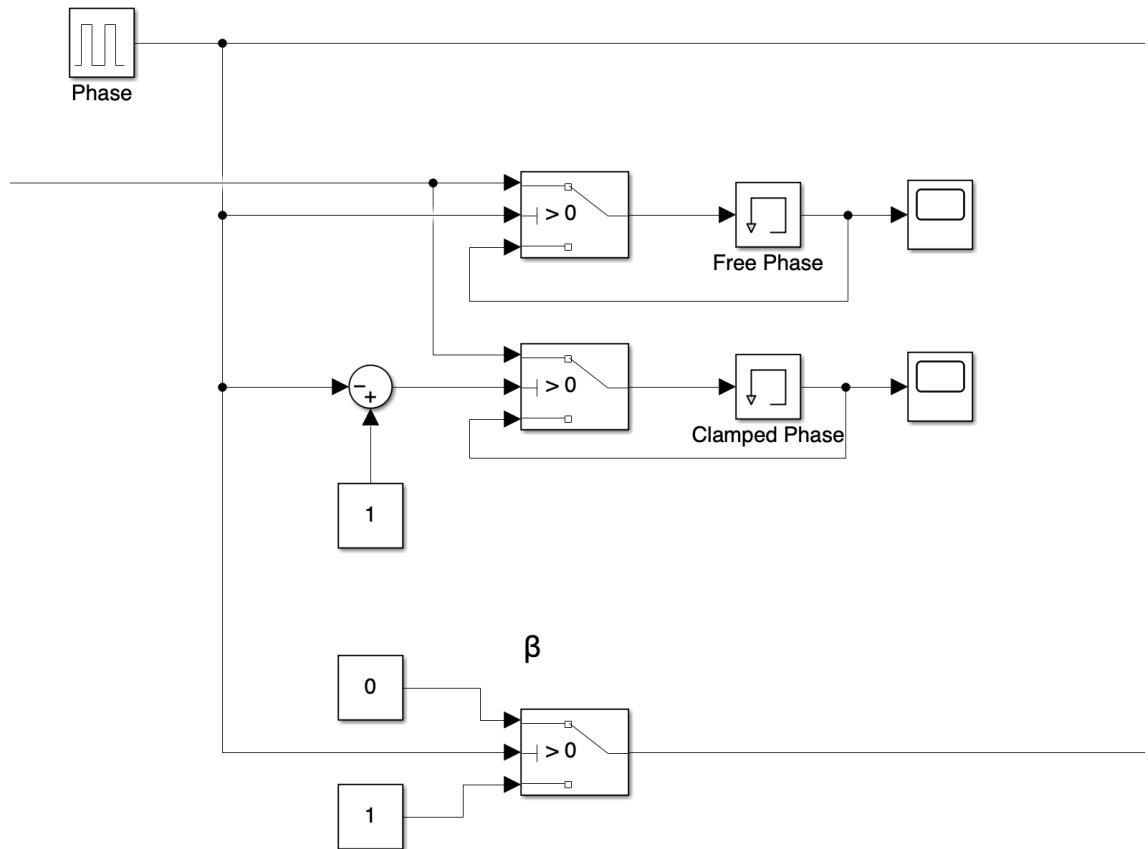
```
1 function y = sigmoid_derivative(x)
2 sigmoid = 1 ./ (1 + exp(-x));
3 y = sigmoid .* (1 - sigmoid);
4 end

1 function W = create3x3WeightMatrix(w12, w13, w23)
2 W = [0,      w12,   w13;
3       w12,    0,     w23;
4       w13,   w23,    0];
5 end
```

## Anhang 2: Umsetzung des Equilibrium-Propagation in Simulink

In Abbildung 26 ist ein diskreter Ansatz in Simulink für EP zu sehen, der sich durch den „Memory“-Block (dargestellt durch den rotierenden Pfeil) auszeichnet. Da dieser Block den jeweils letzten Eingabewert speichert und diese Funktion nicht auf analoge Rechner übertragbar ist, kann dieser Ansatz praktisch nicht umgesetzt werden.

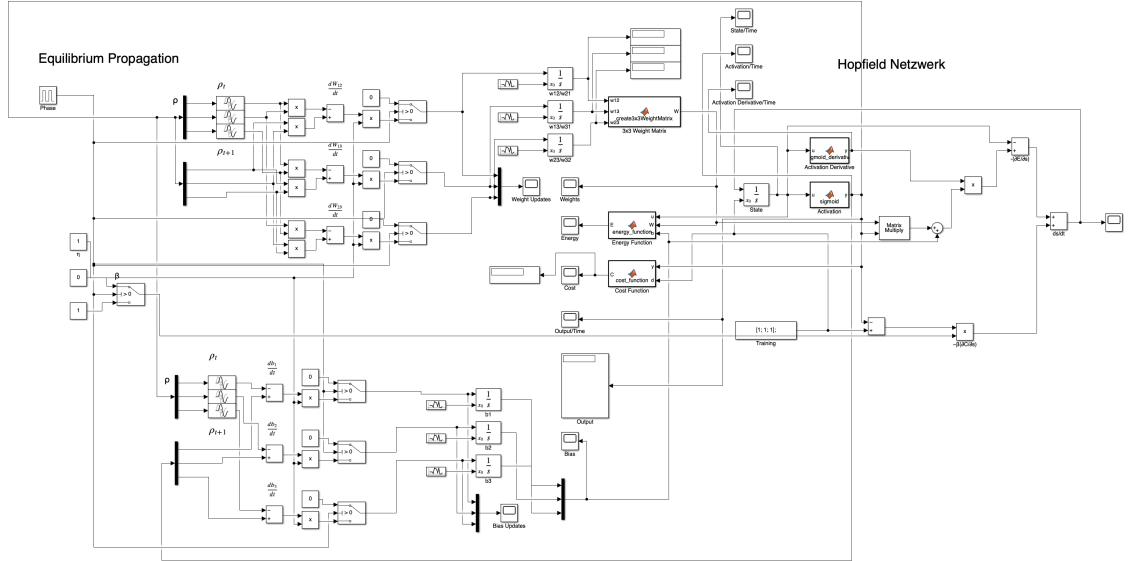
Abbildung 26: Umsetzung eines diskreten Ansatzes zur Lösung des EP in Simulink



Quelle: Eigene Darstellung

Das C-EP wurde in Simulink für ein HNN mit drei Neuronen und damit drei Gewichten modelliert und ist in Abbildung 27 dargestellt.

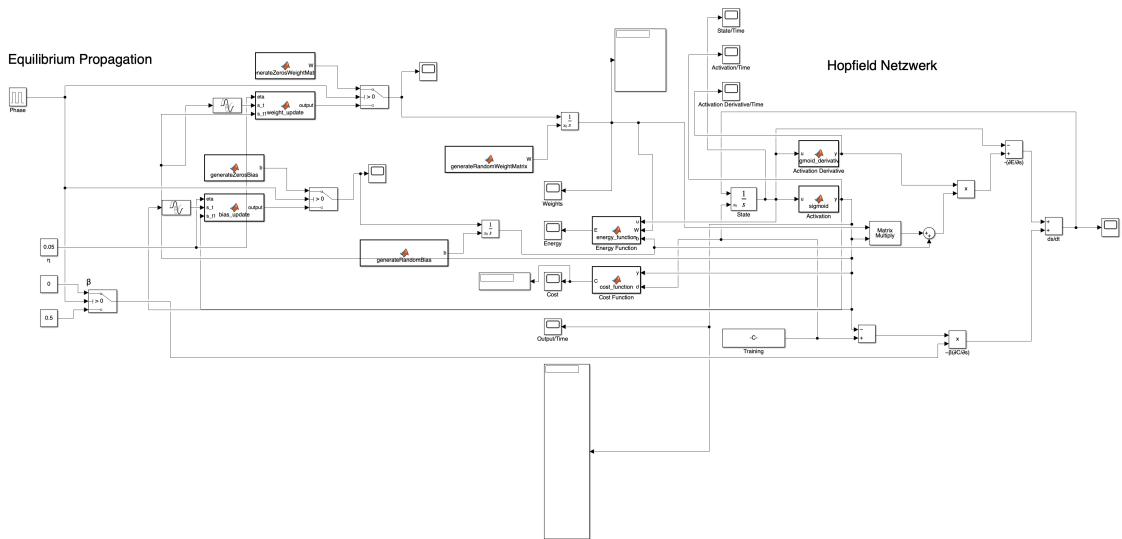
**Abbildung 27: Umsetzung einer Annäherung an C-EP in Simulink**



Quelle: Eigene Darstellung

Um den Zeitaufwand durch die Konstruktion eines größeren Modells des C-EP für größere Netzwerke zu sparen, wurde die Simulation mit sechs Neuronen aus 4.2.4 mithilfe des in Abbildung 28 durchgeführt. Dieses Modell nutzt Matlab Funktionsblöcke, um Eingabewerte in Form von Vektoren bzw. Matrizen beliebiger Größe verarbeiten zu können.

**Abbildung 28: Größere Simulationen des C-EP können mithilfe der Matlab-Funktionsblöcke in Simulink durchgeführt werden, um Aufwand beim Modellieren zu sparen.**



Quelle: Eigene Darstellung

---

## Literaturverzeichnis

- Ackley, David H., Hinton, Geoffrey E., Sejnowski, Terrence J.* (1985): A learning algorithm for boltzmann machines, in: Cognitive Science, 9 (1985), Nr. 1, S. 147–169
- Alonso, Gabino* (2019): Get Up and Running with LTspice, in: Analog Dialogue, 53 (2019), Nr. 4
- Bengio, Yoshua, Fischer, Asja* (2015): Early inference in energy-based models approximates back-propagation, in (2015)
- Deng, Li* (2012): The MNIST database of handwritten digit images for machine learning research [best of the web], in: IEEE Signal Process. Mag. 29 (2012), Nr. 6, S. 141–142
- Ernoult, Maxence, Grollier, Julie, Querlioz, Damien, Bengio, Yoshua, Scellier, Benjamin* (2020): Equilibrium Propagation with continual weight updates, in (2020)
- Géron, Aurélien* (2019): Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2. Aufl., o. O.: O'Reilly Media, 2019
- GmbH, Anabrid* (2025): LUCIDAC User Manual, 3.0 (third edition), o. O., 2025
- Goldberg, E. A., Jules, L.* (1954): U.S. Patent No. 2,684,999, (o. O.), 1954
- Guo, Xinjie, Merrikh-Bayat, Farnood, Gao, Ligang, Hoskins, Brian D, Alibart, Fabien, Linares-Barranco, Bernabe, Theagarajan, Luke, Teuscher, Christof, Strukov, Dmitri B* (2015): Modeling and experimental demonstration of a Hopfield network analog-to-digital converter with hybrid CMOS/memristor circuits, in: Front. Neurosci. 9 (2015), S. 488
- Hong, Qinghui, Li, Ya, Wang, Xiaoping* (2020): Memristive continuous Hopfield neural network circuit for image restoration, in: Neural Comput. Appl. 32 (2020), Nr. 12, S. 8175–8185
- Hopfield, J J* (1982): Neural networks and physical systems with emergent collective computational abilities. In: Proceedings of the National Academy of Sciences, 79 (1982), Nr. 8, S. 2554–2558
- Hopfield, J J* (1984): Neurons with graded response have collective computational properties like those of two-state neurons. In: Proceedings of the National Academy of Sciences, 81 (1984), Nr. 10, S. 3088–3092
- Hu, S G, Liu, Y, Liu, Z, Chen, T P, Wang, J J, Yu, Q, Deng, L J, Yin, Y, Hosaka, Sumio* (2015): Associative memory realized by a reconfigurable memristive Hopfield neural network, in: Nat. Commun. 6 (2015), Nr. 1, S. 7522

---

*Kendall, Jack, Pantone, Ross, Manickavasagam, Kalpana, Bengio, Yoshua, Scellier, Benjamin* (2020): Training end-to-end analog neural networks with Equilibrium Propagation, in (2020)

*LeCun, Yann, Chopra, Sumit, Hadsell, Raia, Ranzato, Marc'Aurelio, Huang, Fu Jie* (2006): A Tutorial on Energy-Based Learning, in: *Bakir, G., Hofman, T., Schölkopf, B., Smola, A., Taskar, B.* (Hrsg.), Predicting Structured Data, v1.0, August 19, 2006, o. O.: MIT Press, 2006

*Lowel, Siegrid, Singer, Wolf* (1992): Selection of intrinsic horizontal connections in the visual cortex by correlated neuronal activity, in: Science, 255 (1992), Nr. 5041, S. 211

*Martin, Erwann, Ernoult, Maxence, Laydevant, Jérémie, Li, Shuai, Querliz, Damien, Petrisor, Teodora, Grollier, Julie* (2020): EqSpike: Spike-driven Equilibrium Propagation for neuromorphic implementations, in (2020)

*Mathews, Pranav O., Hasler, Jennifer O.* (2023): Physical Computing for Hopfield Networks on a Reconfigurable Analog IC, in: 2023 IEEE International Symposium on Circuits and Systems (ISCAS), o. O., 2023, S. 1–5

*McCaughan, Adam N, Oripov, Bakhrom G, Ganesh, Natesh, Nam, Sae Woo, Dienstfrey, Andrew, Buckley, Sonia M* (2023): Multiplexed gradient descent: Fast online training of modern datasets on hardware neural networks without backpropagation, en, in: APL Machine Learning, 1 (2023), Nr. 2

*McCulloch, Warren S., Pitts, Walter* (1943): A logical calculus of the ideas immanent in nervous activity, in: The Bulletin of Mathematical Biophysics, 5 (1943), Nr. 4, S. 115–133

*Onen, Murat, Gokmen, Tayfun, Todorov, Teodor K, Nowicki, Tomasz, del Alamo, Jesus A, Rozen, John, Haensch, Wilfried, Kim, Seyoung* (2022): Neural network training with asymmetric crosspoint elements, in (2022)

*OpenAI, Achiam, Josh, Adler, Steven, Agarwal, Sandhini, Ahmad, Lama, Akkaya, Ilge, Aleman, Florencia Leoni, Almeida, Diogo, Altenschmidt, Janko, Altman, Sam, Anadkat, Shyamal, Avila, Red, Babuschkin, Igor, Balaji, Suchir, Balcom, Valerie, Baltescu, Paul, Bao, Haiming, Bavarian, Mohammad, Belgum, Jeff, Bello, Irwan, Berdine, Jake, Bernadett-Shapiro, Gabriel, Berner, Christopher, Bogdonoff, Lenny, Boiko, Oleg, Boyd, Madelaine, Brakman, Anna-Luisa, Brockman, Greg, Brooks, Tim, Brundage, Miles, Button, Kevin, Cai, Trevor, Campbell, Rosie, Cann, Andrew, Carey, Brittany, Carlson, Chelsea, Carmichael, Rory, Chan, Brooke, Chang, Che, Chantzis, Fotis, Chen, Derek, Chen, Sully, Chen, Ruby, Chen, Jason, Chen, Mark, Chess, Ben, Cho, Chester, Chu, Casey, Chung, Hyung Won, Cummings, Dave, Currier, Jeremiah, Dai, Yunxing, Decareaux, Cory, Degry, Thomas, Deutscher, Noah, Deville, Damien, Dhar, Arka, Dohan, David, Dowling, Steve, Dunning, Sheila, Ecoffet, Adrien, Eleti, Atty, Eloundou, Tyna, Farhi, David, Fedus, Liam, Felix, Niko, Fishman, Simón Posada, Forte, Juston, Fulford, Isabella, Gao, Leo, Georges, Elie, Gibson, Christian, Goel, Vik, Gogineni, Tarun, Goh, Gabriel, Gontijo-Lopes, Rapha, Gordon, Jonathan, Grafstein, Morgan, Gray, Scott, Greene, Ryan, Gross, Joshua, Gu, Shixiang Shane, Guo, Yufei, Hallacy, Chris, Han, Jesse, Harris, Jeff, He, Yuchen, Heaton, Mike, Heidecke,*

---

*Johannes, Hesse, Chris, Hickey, Alan, Hickey, Wade, Hoeschele, Peter, Houghton, Brandon, Hsu, Kenny, Hu, Shengli, Hu, Xin, Huizinga, Joost, Jain, Shantanu, Jain, Shawn, Jang, Joanne, Jiang, Angela, Jiang, Roger, Jin, Haozhun, Jin, Denny, Jomoto, Shino, Jonn, Billie, Jun, Heewoo, Kaftan, Tomer, Kaiser, Łukasz, Kamali, Ali, Kanitscheider, Ingmar, Keskar, Nitish Shirish, Khan, Tabarak, Kilpatrick, Logan, Kim, Jong Wook, Kim, Christina, Kim, Yongjik, Kirchner, Jan Hendrik, Kirov, Jamie, Knight, Matt, Kokotajlo, Daniel, Kondraciuk, Łukasz, Kondrich, Andrew, Konstantinidis, Aris, Kosic, Kyle, Krueger, Gretchen, Kuo, Vishal, Lampe, Michael, Lan, Ikai, Lee, Teddy, Leike, Jan, Leung, Jade, Levy, Daniel, Li, Chak Ming, Lim, Rachel, Lin, Molly, Lin, Stephanie, Litwin, Mateusz, Lopez, Theresa, Lowe, Ryan, Lue, Patricia, Makanju, Anna, Malfacini, Kim, Manning, Sam, Markov, Todor, Markovski, Yaniv, Martin, Bianca, Mayer, Katie, Mayne, Andrew, McGrew, Bob, McKinney, Scott Mayer, McLeavey, Christine, McMillan, Paul, McNeil, Jake, Medina, David, Mehta, Alok, Menick, Jacob, Metz, Luke, Mishchenko, Andrey, Mishkin, Pamela, Monaco, Vinnie, Morikawa, Evan, Mossing, Daniel, Mu, Tong, Murati, Mira, Murk, Oleg, Mély, David, Nair, Ashvin, Nakano, Reiichiro, Nayak, Rajeev, Neelakantan, Arvind, Ngo, Richard, Noh, Hyeonwoo, Ouyang, Long, O'Keefe, Cullen, Pachocki, Jakub, Paino, Alex, Palermo, Joe, Pantuliano, Ashley, Parascandolo, Giambattista, Parish, Joel, Parparita, Emy, Passos, Alex, Pavlov, Mikhail, Peng, Andrew, Perelman, Adam, de Avila Belbute Peres, Filipe, Petrov, Michael, de Oliveira Pinto, Henrique Ponde, Michael, Pokorny, Pokrass, Michelle, Pong, Vitchyr H., Powell, Tolly, Power, Alethea, Power, Boris, Proehl, Elizabeth, Puri, Raul, Radford, Alec, Rae, Jack, Ramesh, Aditya, Raymond, Cameron, Real, Francis, Rimbach, Kendra, Ross, Carl, Rotsted, Bob, Roussez, Henri, Ryder, Nick, Saltarelli, Mario, Sanders, Ted, Santurkar, Shibani, Sastry, Girish, Schmidt, Heather, Schnurr, David, Schulman, John, Selsam, Daniel, Sheppard, Kyla, Sherbakov, Toki, Shieh, Jessica, Shoker, Sarah, Shyam, Pranav, Sidor, Szymon, Sigler, Eric, Simens, Maddie, Sitkin, Jordan, Slama, Katarina, Sohl, Ian, Sokolowsky, Benjamin, Song, Yang, Staudacher, Natalie, Such, Felipe Petroski, Summers, Natalie, Sutskever, Ilya, Tang, Jie, Tezak, Nikolas, Thompson, Madeleine B., Tillet, Phil, Toootoonchian, Amin, Tseng, Elizabeth, Tuggle, Preston, Turley, Nick, Tworek, Jerry, Uribe, Juan Felipe Cerón, Vallone, Andrea, Vijayvergiya, Arun, Voss, Chelsea, Wainwright, Carroll, Wang, Justin Jay, Wang, Alvin, Wang, Ben, Ward, Jonathan, Wei, Jason, Weinmann, CJ, Welihinda, Akila, Welinder, Peter, Weng, Jiayi, Weng, Lilian, Wiethoff, Matt, Willner, Dave, Winter, Clemens, Wolrich, Samuel, Wong, Hannah, Workman, Lauren, Wu, Sherwin, Wu, Jeff, Wu, Michael, Xiao, Kai, Xu, Tao, Yoo, Sarah, Yu, Kevin, Yuan, Qiming, Zaremba, Wojciech, Zellers, Rowan, Zhang, Chong, Zhang, Marvin, Zhao, Shengjia, Zheng, Tianhao, Zhuang, Juntang, Zhuk, William, Zoph, Barret (2024): GPT-4 Technical Report, o. O., 2024, arXiv: 2303.08774 [cs.CL], URL: <https://arxiv.org/abs/2303.08774>*

*Peasley, Eric, Mear, I. F. (2018): An Introduction to Using Simulink, Version 5.0, o. O., 2018*

*Rasch, Malte J, Carta, Fabio, Fagbohungbe, Omobayode, Gokmen, Tayfun (2024): Fast and robust analog in-memory deep neural network training, en, in: Nat. Commun. 15 (2024), Nr. 1, S. 7133*

- Rosenblatt, F.* (1958): The perceptron: A probabilistic model for information storage and organization in the brain. In: *Psychological Review*, 65 (1958), Nr. 6, S. 386–408
- Rumelhart, D. E., Hinton, G. E., Williams, R. J.* (1986): Learning internal representations by error propagation, in: *Rumelhart, D. E., McClelland, J. L.* (Hrsg.), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Bd. 1, o. O.: MIT Press, 1986, S. 318–362
- Sakemi, Yusuke, Okamoto, Yuji, Morie, Takashi, Nobukawa, Sou, Hosomi, Takeo, Aihara, Kazuyuki* (2024): Training physical neural networks for analog in-memory computing, in (2024)
- Scellier, Benjamin, Bengio, Yoshua* (2017): Equilibrium Propagation: Bridging the Gap between Energy-Based Models and Backpropagation, in: *Frontiers in Computational Neuroscience*, 11 (2017)
- Ullmann, Bernd* (2022): *Analog Computing*, o. O.: DeGruyter, 2022

## Internetquellen

*Devices, Analog* (2024): LTspice Simulator - Design Tools and Calculators, <<https://www.analog.com/en/resources/design-tools-and-calculators/ltpice-simulator.html>> (2024) [Zugriff: 2024-02-01]

*GmbH, Anabrid* (2024a): Anabrid - Analog and Hybrid Computing Solutions, <<https://anabrid.com/>> (2024) [Zugriff: 2024-02-01]

*GmbH, Anabrid* (2024b): Documentation - The Analog Thing, <<https://the-analog-thing.org/docs/dirhtml/>> (2024) [Zugriff: 2024-02-01]

*GmbH, Anabrid* (2024c): Lucipy - Python Client for LUCIDAC, <<https://github.com/anabrid/lucipy>> (2024) [Zugriff: 2024-02-01]

*Instruments, National* (2024): What is Multisim? - Application Software for Electronic Test and Instrumentation, <<https://www.ni.com/de/shop/electronic-test-instrumentation/application-software-for-electronic-test-and-instrumentation-category/what-is-multisim.html>> (2024) [Zugriff: 2024-02-01]

*Maffei, Paolo* (2024): LTSpice Analog Computer, <<https://github.com/Paolo-Maffei/LTSpice-Analog-Computer>> (2024) [Zugriff: 2024-02-01]

*MathWorks* (2024): Simscape - Physical Modeling Environment, <<https://de.mathworks.com/products/simscape.html>> (2024) [Zugriff: 2024-02-01]

*Systems, Cadence Design* (2024): PSpice - Analog and Mixed-Signal Simulation, <[https://www.cadence.com/en\\_US/home/tools/pcb-design-and-analysis/analog-mixed-signal-simulation/pspice.html](https://www.cadence.com/en_US/home/tools/pcb-design-and-analysis/analog-mixed-signal-simulation/pspice.html)> (2024) [Zugriff: 2024-02-01]