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 :	17. Februar 2025

Inhaltsverzeichnis

Abbildungsverzeichnis **IV**

Tabellenverzeichnis **VI**

Abkürzungsverzeichnis **VII**

Symbolverzeichnis **VIII**

Glossar **IX**

1 Einleitung **1**

2 Grundlagen **2**

2.1 Neuronale Netze . . . 2

2.1.1 Aufbau und Arten neuronaler Netze . . . 2

2.1.2 Training neuronaler Netze mit Backpropagation und Gradient Descent 4

2.2 Analoge Computer . . . 5

2.2.1 Geschichte zu analogen Computern . . . 5

2.2.2 Typische Komponenten und Bauweisen analoger Computer . . . 5

2.2.3 Aufbau von Schaltkreisen zur Lösung von Differentialgleichungen . . 10

2.3 Energiebasierte Modelle . . . 12

2.3.1 Definition: energiebasierte Modelle und energiebasiertes Lernen . . 12

2.3.2 Das Hopfield-Netzwerk: Eine Herangehensweise an neuronale Netze 12

2.3.3 Energiebasiertes Lernen am Beispiel Equilibrium Propagation . . . 14

2.3.3.1 Mathematische Grundlagen . . . 14

2.3.3.2 Theoretische Anwendung am Beispiel eines Hopfield-Netzwerks . . . 15

2.4 Methodik . . . 17

2.4.1 Forschungsansatz: Konstruktion . . . 17

2.4.2 Auswahl und Grundlagen der Simulationssoftware . . . 17

2.4.3 Auswahl der Referenzarbeit für das Netzwerkdesign . . . 19

2.4.4 Erwartete Ergebnisse . . . 20

2.5 Konstruktion . . . 20

2.5.1 Simulationsumgebung . . . 20

2.5.1.1 Übernahme des Hopfield-Netzwerks . . . 20

2.5.1.2 Notwendige Modifikationen am Netzwerk für Equilibrium Propagation . . . 21

2.5.1.3 Simulation des angepassten Netzwerks 22
2.5.2 Konstruktion des Equilibrium Propagation Algorithmus 24
2.5.2.1 Umsetzung der theoretischen Modelle in der Simulationssoftware 24
2.5.2.2 Anwendung auf das Hopfield-Netzwerk 26
2.5.2.3 Herausforderungen bei der Übernahme und Anpassung . . 26
2.5.2.4 Strategien zur Fehlerbehebung und Optimierung 29
2.5.2.5 Validierung des Algorithmus durch Testläufe 30

Anhang **34**

Literaturverzeichnis **35**

Abbildungsverzeichnis

Abbildung 1: Operationsverstärker mit Rückkopplung. Quelle: *Ulmann*, *B.*, 2022, S. 76 6
Abbildung 2: Symbol des Summierers. Quelle: *Eigene Darstellung* 7
Abbildung 3: Symbol des Integrierers. Quelle: *Eigene Darstellung* 7
Abbildung 4: Symbol des Potentiometers. Quelle: *Eigene Darstellung* 7
Abbildung 5: Symbol des Funktionsgenerators. Quelle: *Eigene Darstellung* 8
Abbildung 6: The Analog Thing des Herstellers anabrid GmbH. Quelle: *GmbH*, *A.*, 2024b 9
Abbildung 7: Simulation der Euler-Spirale auf einem The Analog Thing (THAT). Quelle: *GmbH*, *A.*, 2024b 9
Abbildung 8: Anwendung der Kelvin Feedback Technik. Quelle: *Ulmann*, *B.*, 2022, S. 154 10
Abbildung 9: Masse-Feder-Dämpfer System ohne Rückkopplung. Quelle: *Ulmann*, *B.*, 2022, S. 170 11
Abbildung 10: Vollständiges Masse-Feder-Dämpfer System. Quelle: *Ulmann*, *B.*, 2022, S. 170 12
Abbildung 11: Simulation eines Masse-Feder-Dämpfer Systems in Simulink. Quelle: *Ulmann*, *B.*, 2022, S. 241 19
Abbildung 12: Masse-Feder-Dämpfer System in Simulink und Simscape. Quelle: *MathWorks*, 2024 20
Abbildung 13: Erste Simulation des Hopfield-Netzwerk. Quelle: *Eigene Darstellung* 23
Abbildung 14: Zweite Simulation des Hopfield-Netzwerk. Quelle: *Eigene Darstellung* 23
Abbildung 15: Dritte Simulation des Hopfield-Netzwerk. Quelle: *Eigene Darstellung* 23
Abbildung 16: Ausgabe des zweiphasigen Lernprozesses des Equilibrium Propagation im ersten vorgestellten Ansatz. Quelle: *Eigene Darstellung* . . . 25
Abbildung 17: Graph der Aktivierungsfunktion $\rho(x) = \frac{1}{1+e^{-x}}$ (rot) und ihrer Ableitung (grün). Quelle: *Eigene Darstellung* 27
Abbildung 18: Auswirkungen einer nicht-negativen Aktivierungsfunktion auf das Continual Equilibrium Propagation. Quelle: *Eigene Darstellung* . . . 27
Abbildung 19: Die Parameter des Modells gelangen zu keinem Fixpunkt und verfehlen damit das Minimum der Kostenfunktion. Quelle: *Eigene Darstellung* 28
Abbildung 20: Eine Annäherung an C-EP findet passende Parameter für das Netzwerk. Quelle: *Eigene Darstellung* 29
Abbildung 21: Graph der Aktivierungsfunktion (rot) und der skalierten Variante $\rho(x) = \frac{1}{1+e^{-4x}}$ (orange). Quelle: *Eigene Darstellung* 30

Abbildung 22: Simulationen des Continual Equilibrium Propagation mit $\beta = 1, \eta = 0.5$. Dargestellt ist die Ausgabe des Netzwerks. Quelle: *Eigene Darstellung* 31
Abbildung 23: Annäherungen des Continual Equilibrium Propagation an den Grenzwert 0. Quelle: *Eigene Darstellung* 31
Abbildung 24: Simulationen des Continual Equilibrium Propagation mit der Aktivierungsfunktion: $\rho(x) = tanh(2x)$. Quelle: *Eigene Darstellung* 32
Abbildung 25: Simulation des Continual Equilibrium Propagation mit der Aktivierungsfunktion „ReLU“ ($R(x) = max(x, 0)$). Gezeigt wird die Dynamik der Zustände des Netzwerks $\frac{ds}{dt}$. Quelle: *Eigene Darstellung* 32
Abbildung 26: Simulationen des Continual Equilibrium Propagation mit 6 Neuronen. Quelle: *Eigene Darstellung* 33

Tabellenverzeichnis

Abkürzungsverzeichnis

EBM Energiebasiertes Modell

THAT The Analog Thing

Symbolverzeichnis

Glossar

Backpropagation Lernalgorithmus für MLP basierend auf dem Gradienten-Verfahren. IX, X, 4, 15

Bias-Neuron Zusätzliches Neuron, welches Konstant den Wert 1 ausgibt. Durch gewichtete Verbindungen kann jedem Neuron der nachfolgenden Ebene ein Bias-Wert zugewiesen werden. 3

CNN Convolutional Neural Network; Neuronales Netz mit Convolutional-Ebenen, welche nur mit jeweils einem Ausschnitt der vorherigen Ebene verbunden sind. 3

Continual Equilibrium Propagation Eine Abwandlung des Equilibrium Propagation mit einer kontinuierlichen Anpassung der Gewichte. IV, V, X, 25, 26, 27, 28, 29, 30, 31, 32, 33

Dense Layer Ebene eines neuronalen Netzes, in der jedes Neuron mit jedem Neuron der vorherigen Ebene verbunden ist. 3

DNN Deep Neuronal Network; Neuronales Netz mit vielen versteckten Ebenen. 3

Drift Kleine Fehler in der Signalverarbeitung führen zu großen Fehlern am Ausgang. 6

Energiebasiertes Modell Ein Energiebasiertes Modell (EBM) definiert eine Energiefunktion, die jeder beliebigen Konfiguration an Variablen eine skalare Energie zuweist. 12, 14

Equilibrium Propagation Ein Lernalgorithmus für neuronale Netzwerke, der auf Energie-Minimierung basiert und Gradienten durch Gleichgewichtszustände berechnet. IV, IX, 14, 15, 20, 21, 24, 25, 26

Field Programmable Analog Array Eine rekonfigurierbare Schaltung, die analaoge Signalverarbeitung durch analoge Blöcke und programmierbare Verbindungen ermöglicht. 19

FNN Feedforward Neural Network; Neuronales Netz, in dem Signal nur in eine Richtung geleitet werden. 3

Forward Pass Erster Schritt im Backpropagation, in dem die Trainingsdaten das Modell durchlaufen. 4

Fully Connected Layer Siehe "'Dense Layer"'. 3

Gradienten-Verfahren Gradient Descent; Ein generischer Optimierungs-Algorithmus. IX, X, 4

Hebbian learning Eine Lernregel die beschreibt, dass sich Gewichtungen zwischen Neuronen verstärken, die gleichzeitige Aktivierungen aufweisen. 4

Hopfield-Netzwerk Ein künstliches neuronales Netzwerk, das als Modell für assoziatives Gedächtnis dient und auf vollständig verbundenen Neuronen mit symmetrischen Gewichtungen basiert. IV, 12, 13, 14, 15, 17, 19, 20, 21, 22, 23, 24, 26, 28, 29, 32

Kelvin Feedback Technik Die Kelvin Feedback Technik nutzt negative Rückkopplungsschleifen in analogen Rechnern, um Differentialgleichungen zu lösen. IV, 10, 11

MLP Multilayer-Perceptron; Eine Zusammensetzung an Perceptrons mit einer Eingabe-Ebene, mindestens einer versteckten Ebene und einer Ausgabe-Ebene. IX, 3, 4

Perceptron Sammlung an TLUs auf einer einzelnen Ebene. 2, 3, 4

Reverse Pass Zweiter Schritt im Backpropagation, in dem das Gradienten-Verfahren auf die Gewichtungen zwischen Neuronen angewendet wird. 4

RNN Recurrent Neural Network; Neuronales Netz mit rückwärtigen Verbindungen, Hauptbestandteil ist hier die Speicherzelle. 3, 26

Spike-driven Equilibrium Propagation Eine Abwandlung des Continual Equilibrium Propagation optimiert für neuronale "'Spiking"'-Netzwerke. 26

Substitutionsmethode Die Substitutionsmethode löst Differentialgleichungen, indem sie eine geeignete Substitution einführt, um die Gleichung auf eine einfachere Form zu transformieren, die direkt integriert oder gelöst werden kann. 10

TLU Threshold Logic Unit; Berechnet die gewichtete Summe seiner Eingabewerte und gibt abhängig von der Überschreitung eines Schwellenwerts entweder 0 oder 1 aus. 2, 3

1 Einleitung

2 Grundlagen

2.1 Neuronale Netze

2.1.1 Aufbau und Arten neuronaler Netze

> „Birds inspired us to fly, burdock plants inspired Velcro, and nature has in- spired 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)

Neuronale Netze wurden erstmals 1943 von den Wissenschaftlern Warren S. McCulloch und Walter Pitts in ihrer gemeinsamen Arbeit „A logical calculus of the ideas immanent in nervous activity“ *McCulloch*, *W. S.*, *Pitts*, *W.*, 1943 eingeführt. Sie stellten ein vereinfachtes Modell eines künstlichen Neurons vor, das lediglich aus binären Eingaben und einer binären Ausgabe besteht und seine Ausgabe aktiviert, sobald sich eine bestimmte Anzahl an Eingabewerten aktiviert. McCulloch und Pitts zeigten, dass dieser einfache Baustein ausreicht, um jeden möglichen logischen Ausdruck als neuronales Netz darzustellen.

Neuronale Netze zeichnen sich mittlerweile durch ihre Vielseitigkeit, Leistungsfähigkeit und Skalierbarkeit aus und haben damit maßgeblich zur Gründung eines neuen Forschungsfeldes, dem „Deep Learning“, beigetragen. *Géron*, *A.*, 2019 Die neu gewonnen Aufmerksamkeit im Zusammenhang mit Deep Learning brachte Innovationen wie das von OpenAI entwickelte Sprachmodell „GPT-4“ oder der Bildgenerierungs-KI „Midjourney“ hervor, wodurch bewiesen wurde, dass neuronale Netze zur Lösung komplexer Aufgaben geeignet sind und sogar teilweise ähnlich brauchbare Ergebnisse wie ein Mensch liefern können. *OpenAI* et al., 2024

Eines der einfachsten neuronalen Netze ist das von *Rosenblatt*, *F.*, 1958 vorgestellte Perceptron, dieses basiert auf dem Konzept der TLU. Die Eingabe- und Ausgabewerte einer TLU sind numerisch und den eingehenden Verbindungen ist jeweils eine Gewichtung zugewiesen. Die TLU berechnet nun die gewichtete Summe der Eingabewerte $z = w_1x_1 + w_2x_2... + w_nx_n = xtw$. 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 eine Bias-Gewichtung zuzuweisen. Die Ausgabe eines Perceptron berechnet sich durch $h_W, b(X) = \rho(XW + b)$ *Géron, A.*, 2019, vgl. S. 284 ff.

Das Perceptron kann nun in mehreren Ebenen genutzt werden, um ein 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 FNN bezeichnet werden, eines mit vielen versteckten Ebenen wird 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 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. ebd., vgl. S. 447 f.

Das 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. *Géron*, *A.*, 2019, 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 Gewichtungen 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 Gewichtungen verstärkt, die zu einer Verringerung des Fehlers führen. *Géron*, *A.*, 2019, vgl. S. 289 ff. Die Lernregel lautet damit:

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

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 Gewichtungen 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 Gewichtungen 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. *Ulmann*, *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. ebd., vgl. S. 69

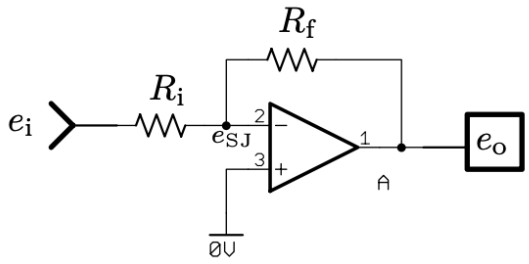
George A. Philbrick spielte eine bedeutende Rolle in der Weiterentwicklung und Verbreitung elektronischer Analogrechner. Er führte kommerzielle Operationsverstärker ein und entwickelte in den 1950er Jahren modulare elektronische Analogrechner, was einen großen Einfluss auf die Standardisierung und Verbreitung 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. *Ulmann*, *B.*, 2022, vgl. S. 73 f.

Abbildung 1: Operationsverstärker mit Rückkopplung. Quelle: ebd., S. 76



Die Formel zu Berechnung der Ausgangsspannung lautet damit wie folgt:

$$e_o = \frac{R_f}{R_i} e_i$$

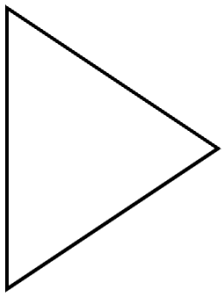
Durch Anpassungen des Eingangssignals bzw. des Rückkopplungs-Signals kann die Funktion des Operationsverstärkers geändert werden. So sind hiermit Operationen wie Summieren, Subtrahieren, Invertieren, das Multiplizieren von Konstanten, Integration und (bedingt) die Differenzierung möglich.

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 ausgeglichen. *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

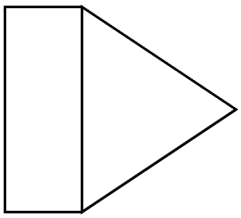
Wird der Feedback-Widerstand R_f durch einen Kondensator C ersetzt, wird die Summe der Eingangssignale über Zeit integriert. Man erhält also einen Integrierer. Durch ein Steuerelement *IC* wird ein Initialwert festgelegt, der so lange ausgegeben wird, bis ein weiteres

Abbildung 2: Symbol des Summierers. Quelle: *Eigene Darstellung*



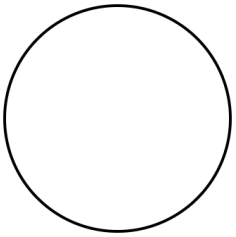
Steuerelement *OP* betätigt wird. Im aktivierten Zustand wird so lange das Integral der Eingangssignale ausgegeben, bis der *OP* Schalter erneut betätigt und der letzte gespeicherte Wert ausgegeben wird. *Ulmann*, *B.*, 2022, vgl. S. 89 ff.

Abbildung 3: Symbol des Integrierers. Quelle: *Eigene Darstellung*



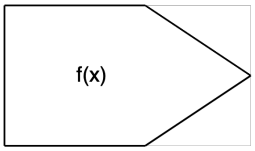
Die Anwendung eines Koeffizienten am Operationsverstärker kann durch ein Potentiometer erfolgen, welches am Eingangssignal angeschlossen ist. Das somit erhaltene Koeffizient-Potentiometer lässt sich über einen speziellen Zustand des Analogrechners, dem *Potentiometer Set* (kurz: *Pot Set*) setzen. In diesem Zustand werden alle Recheneinheiten 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.

Abbildung 4: Symbol des Potentiometers. Quelle: *Eigene Darstellung*



Ein Funktionsgenerator kann genutzt werden, um eine beliebige Funktion $f(x, ...)$ 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 *Ulmann*, *B.*, 2022, vgl. S. 97 ff.

Abbildung 5: Symbol des Funktionsgenerators. Quelle: *Eigene Darstellung*



Die im ersten Augenblick einfache Multiplizierung ist auf analogen Rechnern komplex zu implementieren ebd., vgl. S. 105. Moderne analoge Rechner greifen typischerweise auf die Gilbert Zelle zurück. Die Berechnung einer Division bzw. Quadratwurzel kann durch modifizierte Multiplikatoren erfolgen. ebd., S. 114 f.

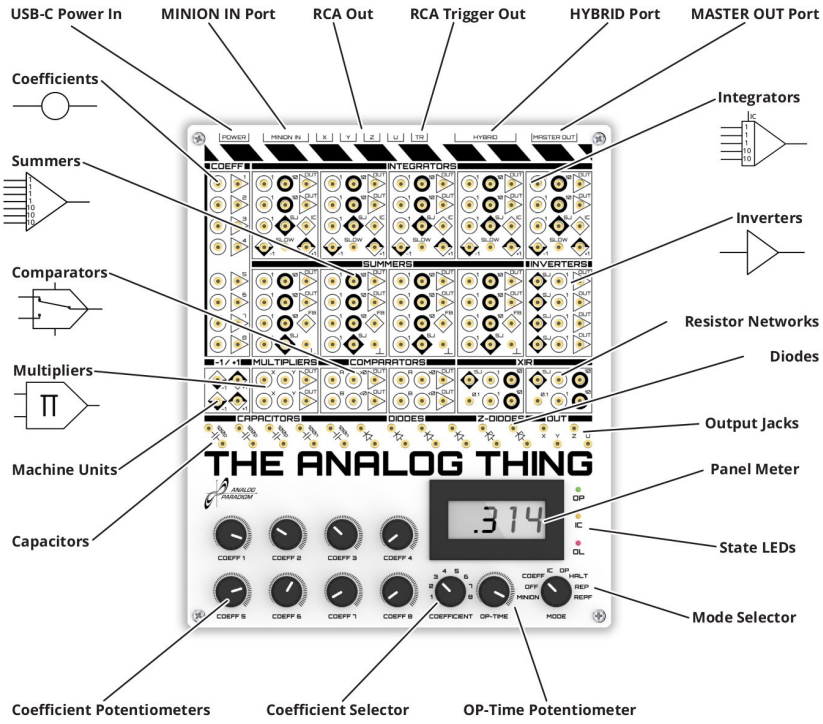
Ein Komparator kann z. B. zur Implementierung einer Schrittfunktion oder zum Tauschen von Variablen genutzt werden. Um das zu erreichen, reicht es schon aus, Dioden in der Feedback-Schleife eines Operationsverstärkers zu platzieren. Nach einem ähnlichen Prinzip kann auch ein Begrenzer erzeugt werden, 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 6). Dabei arbeitet das Gerät mit 10 Volt und in einem Wertebereich von ± 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

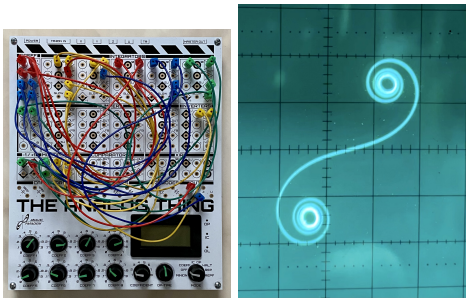
Abbildung 6: The Analog Thing des Herstellers anabrid GmbH. Quelle: ebd.



einige Variablen, welche als Ein- oder Ausgaben zur Auswertung oder Verschaltung mit weiteren analogen Rechnern genutzt werden können *GmbH*, *A.*, 2024b.

Wie in Abbildung 6 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 7)

Abbildung 7: Simulation der Euler-Spirale auf einem THAT. Quelle: ebd.



Das neueste Modell des Herstellers Anabrid, der „lucidac“, lasst sich digital über Software progammieren, 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 eine Gewichtung 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 Verbindung zwischen Komponenten, so gilt:

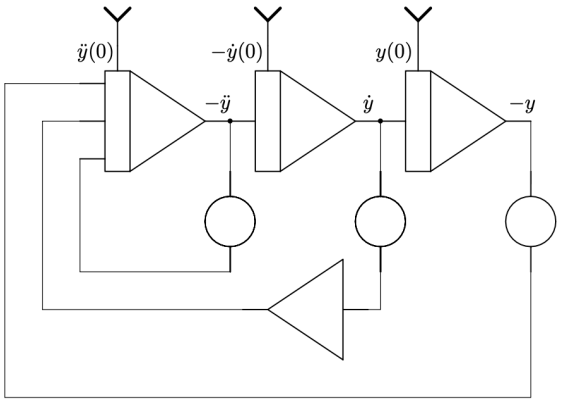
$$C_i^{in} = \sum_{j=0}^{16} M_{ij} C_j^{out}, i, j \in [0, 16]$$

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.

Abbildung 8: Anwendung der Kelvin Feedback Technik. Quelle: ebd., S. 154



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. *Ulmann*, *B.*, 2022, vgl. S. 155 ff.

Der Wertebereich eines analogen Computers liegt i.d.R. bei ± 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:

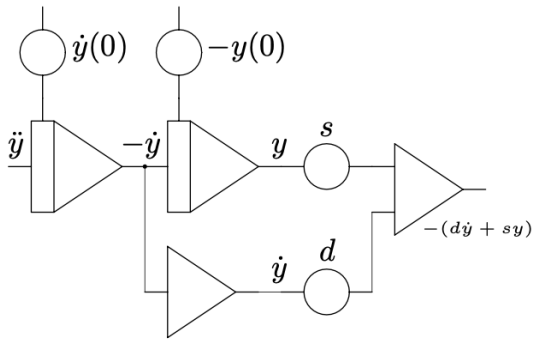
$$my'' + dy' + sy = 0$$

Unter Anwendung der Kelvin Feedback Technik ergibt sich folgende Gleichung:

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

Der Schaltkreis zur Lösung von $-(dy' + sy)$ kann wie folgt gebildet werden:

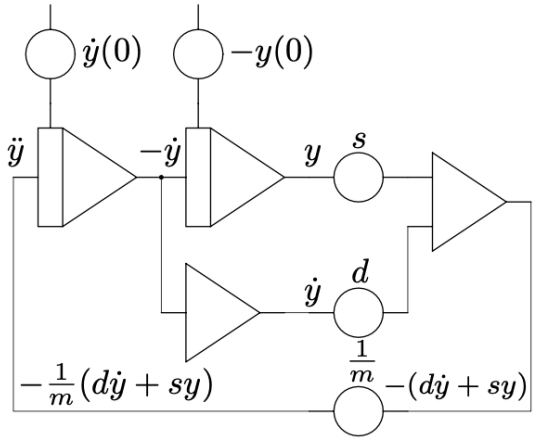
Abbildung 9: Masse-Feder-Dämpfer System ohne Rückkopplung. Quelle: ebd., S. 170



Abschließend muss nur noch die Rückkopplung zu y'' hergestellt werden:

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 10: Vollständiges Masse-Feder-Dämpfer System. Quelle: *Ulmann*, *B.*, 2022, S. 170



2.3 Energiebasierte Modelle

2.3.1 Definition: energiebasierte Modelle und energiebasiertes Lernen

Energiebasierte Modelle finden im maschinellen Lernen Anwendung und arbeiten mithilfe einer Energiefunktion, welche jeder beliebigen Konfiguration an Variablen eine skalare Energie zuweist. Am Beispiel eines neuronalen Netzes könnten diese Variablen die Eingabe-Variablen, die Parameter- und Versteckten-Variablen sowie die Ausgabe-Variablen sein. Zur Inferenz des Modells werden zunächst die Eingabe-Variablen festgelegt und anschließend ein Minimum der Energiefunktion bestimmt. Wurde ein Minimum gefunden, kann die Ausgabe des Modells anhand der Ausgabe-Variablen ausgelesen werden. Ein EBM wird durch Anpassung seiner Energiefunktion trainiert, indem sie kleinere Energien für korrekte Werte und größere Energien für falsche Werte generiert. *LeCun*, *Y.* et al., 2006 Die ersten vorgestellten EBM waren das Hopfield-Netzwerk *Hopfield*, *J. J.*, 1984 und die auf dem Hopfield-Netzwerk 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 Hopfield-Netzwerk 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 verrauschte Eingaben zu vervollständigen, wodurch es robust gegenüber Störungen und Ausfällen einzelner Neuronen ist. Die Fähigkeit, Informationen anhand von Teilinformationen abzurufen, macht das Hopfield-Netzwerk 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 Gewichtungen 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 Hopfield-Netzwerk 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 Hopfield-Netzwerk 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 Energiebasierte Modelle stellt das Equilibrium Propagation dar. In diesem Verfahren wird der Gradient einer Energiefunktion bestimmt und damit die Parameter des Modells angepasst. Die Vorhersagen des Modells werden aus dem Datenpunkt und den Parametern impliziert anstelle diese explizit zu definieren, weshalb sich dieses Verfahren besonders für die Anwendung auf analogen Computern eignet. *Scellier*, *B.*, *Bengio*, *Y.*, 2017

Der Zustand des Modells wird durch den Vektor s dargestellt, v gibt die Eingabe-Variablen an und θ steht für die Parameter des Modells. Die Zustandsvariable s verändert sich über Zeit, sodass die Energiefunktion $E(\theta, v, s)$ minimiert wird. Neben der Energiefunktion wird auch eine Kosten-Funktion $C(\theta, v, s)$ definiert, welche die Diskrepanz zwischen der Ausgabe des Modells und den Zielwerten angibt. Liefert die Energiefunktion für eine Konfiguration an Variablen eine kleinere Energie, so sollte auch der Wert der Kosten-Funktion geringer sein. Zusätzlich wird der Einfluss-Parameter β definiert, der als Skalierungsfaktor für die Kostenfunktion dient. Das Equilibrium Propagation definiert nun eine Gesamtenergiefunktion F als:

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

Die Fixpunkte des Modells werden in der Form $s^{\beta}_{(\theta,v)}$ dargestellt, und stehen für jeweils ein lokales Minimum der Gesamtenergiefunktion F. Mit $\beta = 0$ ergibt sich $s^{0}_{(\theta,v)}$, ein Minimum der Energiefunktion E und damit die Vorhersage des Modells. *Scellier*, *B.*, *Bengio*, *Y.*, 2017

Die Zielfunktion J, die im Equilibrium Propagation optimiert werden soll, lautet:

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

Die Kostenfunktion gibt die Qualität des Modells zu einem beliebigen β an, J hingegen nur für die Vorhersage mit $\beta = 0$. Der Gradient der Zielfunktion J nach θ ist nun durch folgende Formel gegeben:

$$\frac{\partial J}{\partial \theta}(\theta, v) = \lim_{\beta \to 0} \frac{1}{\beta} \left(\frac{\partial F}{\partial \theta}(\theta, v, \beta, s^{\beta}_{(\theta,v)}) - \frac{\partial F}{\partial \theta}(\theta, v, \beta, s^{0}_{(\theta,v)}) \right)$$

Anhand dieser Formel lässt sich die Änderungsrate von θ ableiten:

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

Hieraus ergibt sich in der Praxis unter Anwendung am Hopfield-Netzwerk ein zweiphasiger Lernprozess. In der ersten Phase wird Inferenz durchgeführt, also ein Minimum der Energiefunktion gesucht und die Ausgabe des Netzes ausgelesen. In dieser Phase ist $\beta = 0$. Die zweite Phase setzt $\beta > 0$ und lenkt damit die Ausgabe des Netzes in Richtung des Zielwertes. Die versteckten Variablen des Netzes befinden sich zu Beginn dieser Phase im Gleichgewicht, die Störung an den Ausgabe-Variablen propagiert aber über Zeit zu den versteckten Variablen. Wird ein Netzwerk mit mehreren Ebenen betrachtet, so propagiert die Störung rückwärts durch das Netz, was auch als Backpropagation bezeichnet werden kann. ebd.

2.3.3.2 Theoretische Anwendung am Beispiel eines Hopfield-Netzwerks

Wie von ebd. bereits beschrieben, ist Equilibrium Propagation auf das Hopfield-Netzwerk anwendbar. Für diese Anwendung muss zuerst die Energiefunktion des Modells bestimmt

werden. Seien u die Neuronen des Netzwerks, $W_{i,j}$ die Gewichtung der Verbindung zweier Neuronen und b_i der Bias eines Neuron, 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, versteckte 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:

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

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

$$C := \frac{1}{2}\|y - d\|^2$$

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:

$$F := E + \beta C$$

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:

$$\frac{ds}{dt} = -\frac{\partial E}{\partial s} - \beta\frac{\partial C}{\partial s}$$

Hierbei ist die durch die Hopfield-Energiefunktion ausgeübte Kraft:

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

Durch die Kostenfunktion wirken die Kräfte $-\beta\frac{\partial C}{\partial h_i} = 0$ und $-\beta\frac{\partial C}{\partial y_i} = \beta(d_i - y_i)$.

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^β hervor. Daraus lässt sich die Lernregel für das Hopfield-Netzwerk ableiten:

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

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

2.4 Methodik

2.4.1 Forschungsansatz: Konstruktion

2.4.2 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.

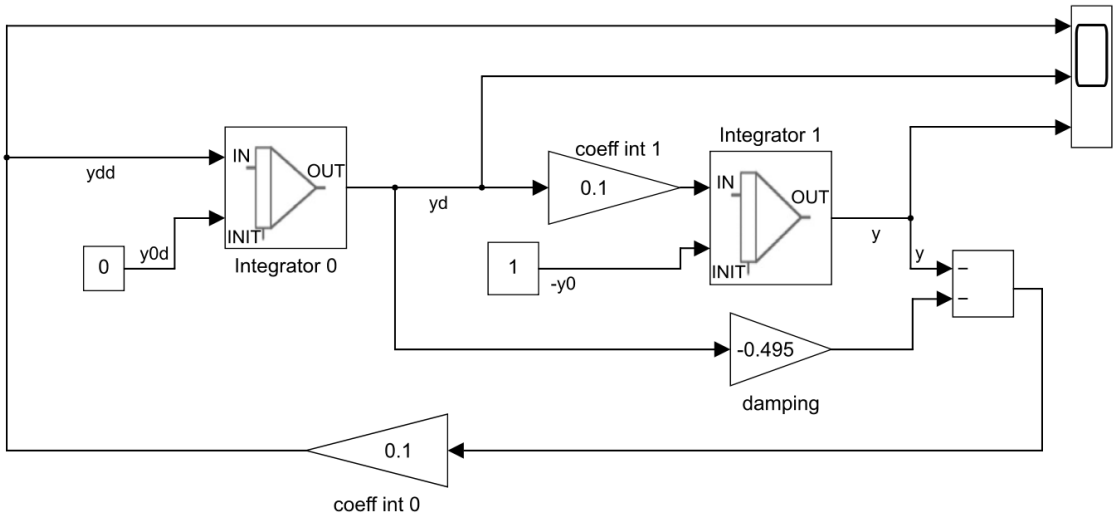
Als Erweiterung zu Matlab kommt Simulink für diese Arbeit infrage, eine grafische Entwicklungsumgebung für modellbasierte Systementwicklung. Simulink zeichnet sich durch die Modellierung von Programmen durch Blockdiagramme aus und stellt Funktionen zur Simulation und Analyse bereit. Des Weiteren lässt 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 10 modelliert werden kann, wurde bereits von *Ulmann* 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 hinzugefügt, wodurch analoge Rechenelemente modelliert werden können. Eine Implementierung des Masse-Feder-Dämpfer Systems anhand dieser Komponenten ist in Abbildung 12 dargestellt. Physikalische Signale können mit den herkömmlichen Simulink-

Abbildung 11: Simulation eines Masse-Feder-Dämpfer Systems in Simulink. Quelle: *Ulmann, B.*, 2022, S. 241

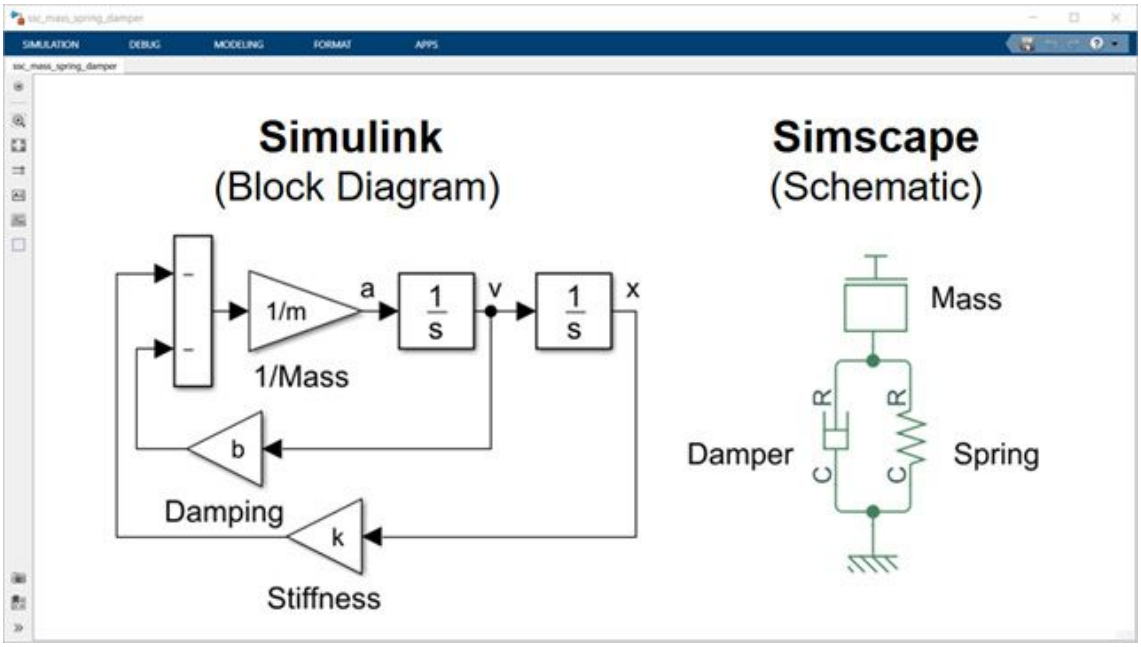


Blöcken über Signal-Konverter verbunden werden, wodurch Simulink- und Simscape-Modelle miteinander kompatibel werden *MathWorks*, 2024. Für diese Arbeit wird jedoch nur Simulink betrachtet, die theoretischen Konzepte werden also nicht physikalisch modelliert.

2.4.3 Auswahl der Referenzarbeit für das Netzwerkdesign

Es haben sich bereits einige Arbeiten mit der Umsetzung eines Hopfield-Netzwerk auf Basis elektrotechnischer Schaltungen beschäftigt. So hat z. B. *Guo* et al. einen Analog-Digital-Wandler mithilfe von Memristoren zur Implementierung der Gewichtungen 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 Hopfield-Netzwerk auf einem Field Programmable Analog Array vor, womit der maximale Schnitt eines Graphen gefunden werden konnte. Diese Umsetzung eignet sich aber auch aufgrund des Einsatzes eines Field Programmable Analog Array auch nicht für diese Arbeit. Ein weiterer Einsatz von Memristoren als Gewichtungen 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 Hopfield-Netzwerk betrachtet, welches 2020 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

Abbildung 12: Masse-Feder-Dämpfer System in Simulink und Simscape. Quelle: *MathWorks*, 2024



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 Equilibrium Propagation 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 Equilibrium Propagation validiert.

2.4.4 Erwartete Ergebnisse

2.5 Konstruktion

2.5.1 Simulationsumgebung

2.5.1.1 Übernahme des Hopfield-Netzwerks

Im Folgenden soll die von *Scellier*, *Bengio* vorgestellte Dynamik $\frac{ds}{dt}$ eines Hopfield-Netzwerk (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 Hopfield-Netzwerk definiert, welche dann über die Dynamik zu einem Fixpunkt gelangen:

$$\frac{ds}{dt} = -\frac{\partial E}{\partial s} - \beta \frac{\partial C}{\partial s}$$

Um diesen Term abzubilden werden die Gewichtungen 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 **??**) 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, Gewichtungen (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 **??** 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 **??** dargestellt, wobei der Bias zur Übersichtlichkeit weggelassen wurde. Aus dieser Schaltung wird ersichtlich, dass die Implementierung eines Hopfield-Netzwerk 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 **??**. 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.

2.5.1.2 Notwendige Modifikationen am Netzwerk für Equilibrium Propagation

Das Equilibrium Propagation definiert mit $C := \frac{1}{2}\|y-d\|^2$ eine Kostenfunktion des Hopfield-Netzwerk, 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 **??** 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 Gewichtungen wurden bisher als konstanten implementiert, müssen aber für die Anwendung eines Lernalgorithmus dynamisch anpassbar sein. Die von *Scellier*, *Bengio* vorgestellte Lernregel ΔW_{ij} kann auch als Integration der Lernregel $\frac{dW_{ij}}{dt}$ interpretiert werden ebd., vgl. S. 5, weshalb zur Implementierung der Gewichtungen Integratoren zum Einsatz kommen können (gleiches gilt für die Bias-Werte). Typischerweise werden Gewichtungen in neuronalen Netzen mit Zufallswerten initialisiert, was in Simulink durch den Block „Random Number“ möglich ist. Da das hier implementierte Hopfield-Netzwerk mit einem Vektor als Repräsentation der Zustände arbeitet, müssen die Gewichtungen durch einen Matlab-Funktionsblock zu einer Matrix zusammengesetzt werden (siehe Anhang **??**). Das angepasste Netzwerk ist im Anhang **??** abgebildet.

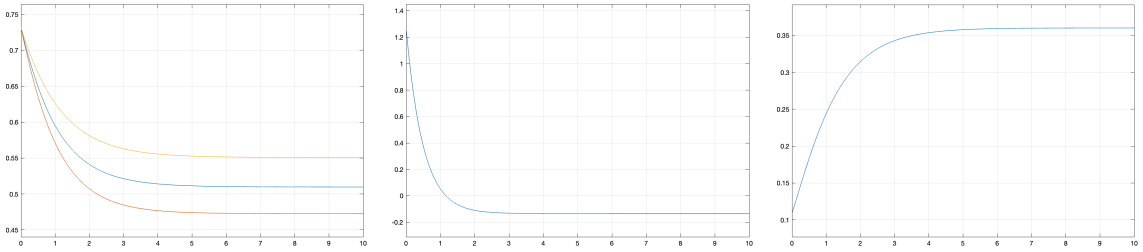
2.5.1.3 Simulation des angepassten Netzwerks

Im Folgenden werden einige Simulationen des in Simulink implementierten Hopfield-Netzwerk beschrieben, ein Beweis für die korrekte Funktionsweise des Netzwerkes ist im Anhang **??** zu finden. Die Gewichtungen 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 Gewichtungen werden mit $(-1.21, 1.319, 0.3204)$ zufällig erzeugt. Wie in Abbildung **??** 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.

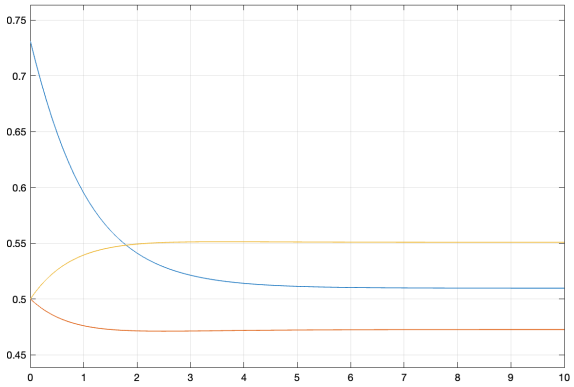
Für die zweite Simulation wird die Eingabe zu $\vec{d} = (1,0,0)$ geändert. Da das Hopfield-Netzwerk die Eigenschaften eines assoziativen Speichers besitzt (vgl. Kapitel 2.3.2), ist zu erwarten, dass das Netzwerk die gleiche Ausgabe wie aus der vorherigen Simulation liefert. Diese These lässt sich anhand der Abbildung 14 bestätigen.

Abbildung 13: Erste Simulation des Hopfield-Netzwerk. Quelle: *Eigene Darstellung*



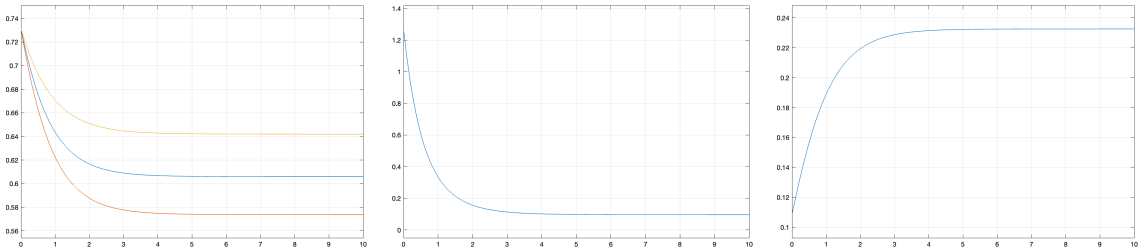
(a) Ausgabe **(b) Energie** **(c) Kosten**

Abbildung 14: Zweite Simulation des Hopfield-Netzwerk. 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 aus Abbildung 15 bestätigen diese Annahmen.

Abbildung 15: Dritte Simulation des Hopfield-Netzwerk. Quelle: *Eigene Darstellung*



(a) Ausgabe **(b) Energie** **(c) Kosten**

2.5.2 Konstruktion des Equilibrium Propagation Algorithmus

2.5.2.1 Umsetzung der theoretischen Modelle in der Simulationssoftware

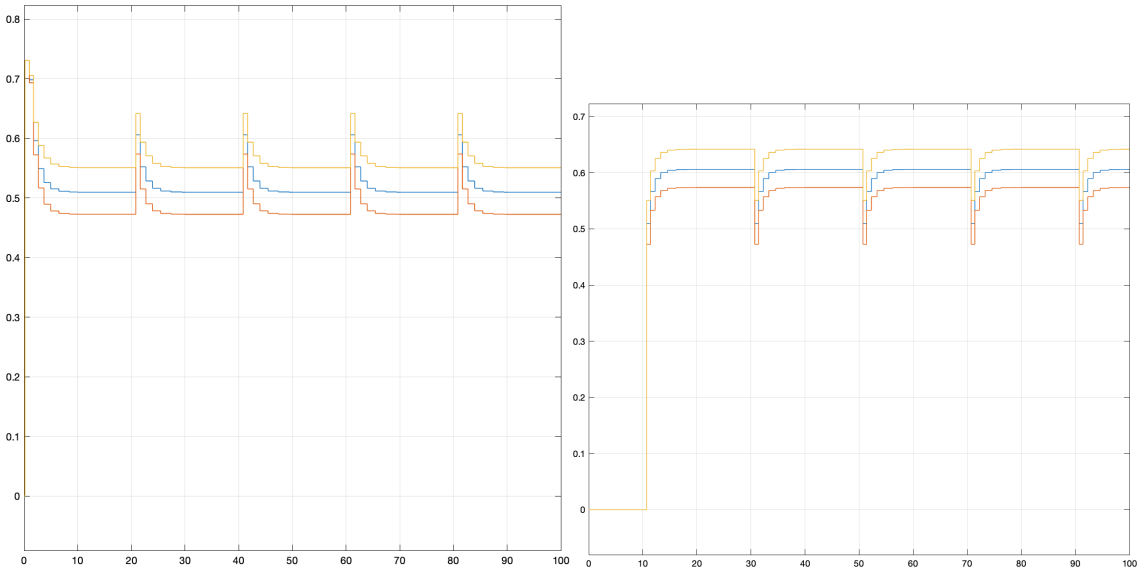
Ein erster, naiver Ansatz zur Implementierung des Equilibrium Propagation beinhaltet die Aufteilung des zweiphasigen Lernprozesses in zeitlich getrennte Abläufe. Damit wird ein einziges Hopfield-Netzwerk benötigt, um Inferenz mit $\beta = 0$ und $\beta > 0$ durchzuführen.

Die Integratoren zur Repräsentation der Gewichtungen 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 β, 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 **??**), 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 16 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.

In einer Arbeit von *Kendall* et al. wurde bereits gezeigt, dass Equilibrium Propagation in einer analogen Simulation in Zusammenarbeit mit herkömmlicher Software implementiert

Abbildung 16: Ausgabe des zweiphasigen Lernprozesses des Equilibrium Propagation im ersten vorgestellten Ansatz. Quelle: *Eigene Darstellung*



(a) Freie Phase **(b) Feste Phase**

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 Equilibrium Propagation auf analoger Hardware realisierbar wird, muss eine kontinuierliche Lernregel aufgestellt werden. Einen Ansatz hierfür bietet das 2020 von *Ernoult* et al. vorgestellte Continual Equilibrium Propagation. Diese Abwandlung des Equilibrium Propagation bietet einerseits seine räumliche Lokalität, welche durch die Eigenschaft der Lernregel ΔW_{ij}, nur mit den beiden Zuständen u_i und u_j auszukommen, gegeben ist (siehe Kapitel 2.3.3.2). Zusätzlich arbeitet Continual Equilibrium Propagation 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 Continual Equilibrium Propagation werden die Parameter θ des Modells nicht durch eine Lernregel wie ΔW aktualisiert, sondern ähnlich wie die Zustände des Modells, durch eine Dynamik beschrieben.

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

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

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

Auf Grundlage des Continual Equilibrium Propagation stellte *Martin* et al. das Spike-driven Equilibrium Propagation, eine weitere Abwandlung des Equilibrium Propagation 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 Hopfield-Netzwerk, vor (*Martin*, *E.* et al., 2020, vgl. S. 3), mit derer hier weitergearbeitet wird.

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

Zur Umsetzung dieses angepassten Lernprozesses kommt, wie im vorherigen Ansatz auch, ein zeitgesteuerter Pulsgenerator zum Einsatz. Der Einflussparameter β wird weiterhin über einen Komparator verbunden mit zwei Konstanten implementiert, zusätzlich wird nun die Lernrate η benötigt, welche einen konstanten Wert über den gesamten Lernprozess annimmt. Die Implementierung des Hopfield-Netzwerk berechnet bereits die Aktivierungen der Neuronen bzw. die Änderungsraten dieser, siehe Kapitel 2.5.1.1, weshalb die entsprechenden Signale aus dem Netzwerk für die Berechnung von $\frac{dW_{ij}}{dt}$ genutzt werden können.

Damit die Integratoren zur Darstellung der Gewichtungen 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 **??** beschrieben. Die Konstruktion des Continual Equilibrium Propagation in Simulink zusammen mit der schematischen Darstellung eines analogen Computers wird im Anhang **??** gezeigt.

2.5.2.2 Anwendung auf das Hopfield-Netzwerk

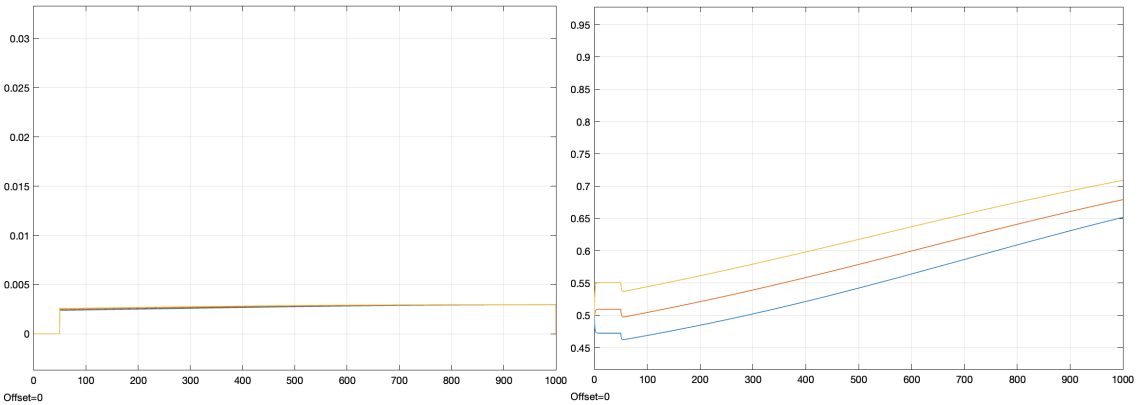
2.5.2.3 Herausforderungen bei der Übernahme und Anpassung

Eine Herausforderung bei der Umsetzung des Continual Equilibrium Propagation ist das Finden einer passenden Aktivierungsfunktion für die Zusammenarbeit mit der Lernregel

$\frac{dW_{ij}}{dt}$ und einer passenden Lernrate η, welche eine effiziente Konvergenz der Gewichtungen gewährleistet. Für die bisher gewählte Aktivierungsfunktion, die logistische Funktion wie dargestellt in Abbildung 17, gilt $\rho(x) > 0$, weshalb auch die Lernregel immer positiv ist. Die Auswirkungen dessen sind in Abbildung 18 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 Gewichtungen den negativen der Kostenfunktion übersteigt.

Abbildung 17: Graph der Aktivierungsfunktion $\rho(x) = \frac{1}{1+e^{-x}}$ (rot) und ihrer Ableitung (grün). Quelle: *Eigene Darstellung*

Abbildung 18: Auswirkungen einer nicht-negativen Aktivierungsfunktion auf das Continual Equilibrium Propagation. Quelle: *Eigene Darstellung*

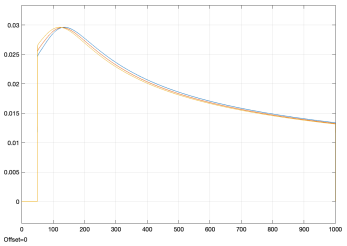


(a) Änderungsrate der Gewichtungen **(b) Ausgabe des Modells**

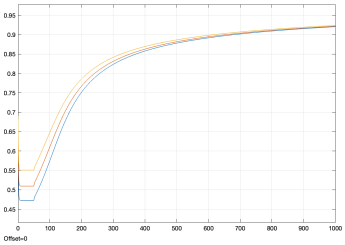
Die Dauer der beiden Phasen des Continual Equilibrium Propagation wurde durch die Konvergenz dieser definiert. So soll die erste Phase durchgeführt werden bis das Netzwerk einen Fixpunkt findet, für die zweite Phase müssen Fixpunkte des Netzwerkes und der Parameter gefunden werden *Ernoult*, *M.* et al., 2020, vgl. S. 3. Die Bestimmung sinnvol-

ler Werte für die Dauer dieser Phasen stellt die Umsetzung des Lernprozesses vor eine weitere Herausforderung. Wie bereits in 2.5.1.3 gezeigt, propagiert das implementierte Hopfield-Netzwerk mit beliebigem β zuverlässig innerhalb von weniger als zehn Zeiteinheiten zu einem Fixpunkt. Gleiches sollte auf die feste Phase zutreffen, dies ist aber, wie in Abbildung 19 gezeigt, nicht der Fall. Eine mögliche Ursache hierfür stellt erneut die Auswahl der Aktivierungsfunktion dar, da, wie aus Abbildung 17 abzulesen, $\rho(x) > 0$ bzw. $\dot{\rho}(x) > 0$ und somit $\frac{dW_{ij}}{dt} > 0$ gilt. Eine Simulation mit $\vec{d} = (0.8, 0.8, 0.8)$ und $\eta = 0.1$ zeigt die valide Anpassung der Gewichtungen bis zu einem Zeitpunkt $t \approx 250$ aber die anschließende fehlende Konvergenz der Parameter.

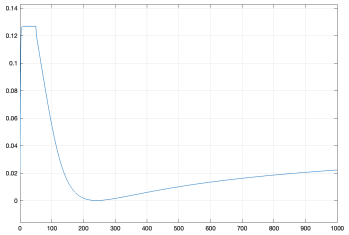
Abbildung 19: Die Parameter des Modells gelangen zu keinem Fixpunkt und verfehlen damit das Minimum der Kostenfunktion. Quelle: *Eigene Darstellung*



(a) Änderungsrate der Gewichtungen



(b) Ausgabe des Modells



(c) Kosten des Modells

Um Fehler bei der Auswahl der Aktivierungsfunktion oder der Übernahme der Lernregel auszuschließen, wird im Folgenden die von *Ernoult* et al. ursprünglich für Continual Equilibrium Propagation vorgestellte Lernregel verwendet:

$$\Delta^{C-EP}_{Wnn+1}(\beta,\eta,t) = \frac{1}{\beta}\left(\sigma\left(s^{n,\beta,\eta}_{t+1}\right)\sigma\left(s^{n+1,\beta,\eta}_{t+1}\right)^{\mathsf{T}} - \sigma\left(s^{n,\beta,\eta}_{t}\right)\sigma\left(s^{n+1,\beta,\eta}_{t}\right)^{\mathsf{T}}\right)$$

Angewandt auf das hier genutzte Hopfield-Netzwerk und unter Beachtung der Lernrate η ergibt sich:

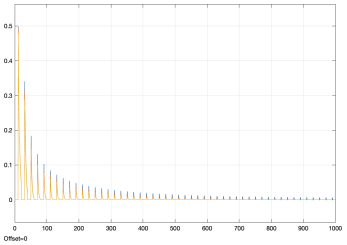
$$\Delta^{C-EP}_{W_{ij}}(\eta,t) \propto \eta\left(\rho(u^{\beta}_{t+1,i})\rho(u^{\beta}_{t+1,j}) - \rho(u^{\beta}_{t,i})\rho(u^{\beta}_{t,j})\right)$$

$$\Delta^{C-EP}_{b_i}(\eta,t) \propto \eta\left(\rho(u^{\beta}_{t+1,i}) - \rho(u^{\beta}_{t,i})\right)$$

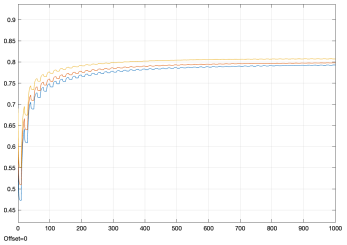
Anhand dieser Lernregel kann die Dynamik $\frac{dW_{ij}}{dt}$ approximiert werden, indem praktisch mit einer Zeitverschiebung gearbeitet wird. Simulink bietet hierfür den Block „Transport

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 mit $\vec{d} = (0.8, 0.8, 0.8)$ (jetzt mit $\eta = 10$) konvergiert mit diesem Ansatz zu sinnvollen Gewichten, siehe Abbildung 20. Im Anhang **??** wird das Simulink-Modell zu diesem Ansatz gezeigt.

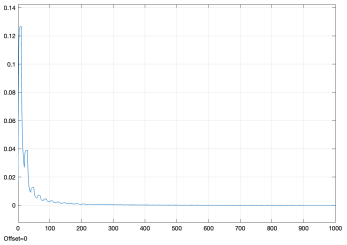
Abbildung 20: Eine Annäherung an C-EP findet passende Parameter für das Netzwerk. Quelle: *Eigene Darstellung*



(a) Änderungsrate der Gewichtungen



(b) Ausgabe des Modells



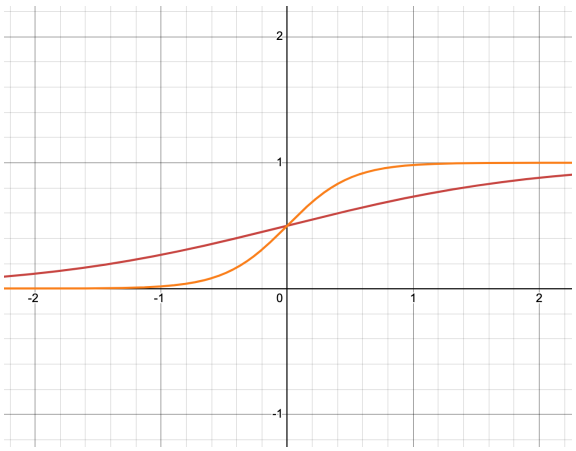
(c) Kosten des Modells

2.5.2.4 Strategien zur Fehlerbehebung und Optimierung

Im vorherigen Kapitel 2.5.2.3 wurde bereits gezeigt, dass die Implementierung des Continual Equilibrium Propagation 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 Gewichtungen ist für diese Zielwerte auch nicht möglich, weshalb das Continual Equilibrium Propagation in diesem Fall die Gewichtungen endlos vergrößert / 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 Hopfield-Netzwerk 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 21 dargestellt.

Abbildung 21: Graph der Aktivierungsfunktion (rot) und der skalierten Variante $\rho(x) = \frac{1}{1+e^{-4x}}$ (orange). Quelle: *Eigene Darstellung*



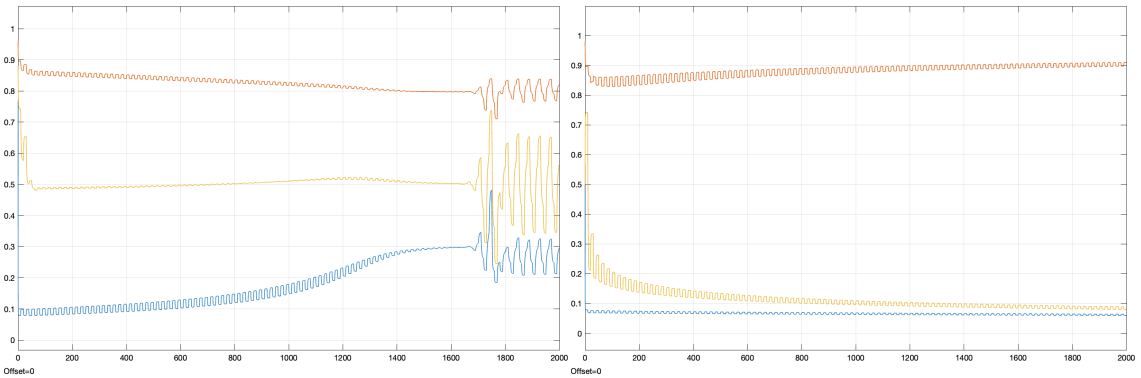
Mit den Parametern $\beta = 1$ und $\eta = 1$ findet das Continual Equilibrium Propagation mit dieser skalierten Aktivierungsfunktion und den Zielwerten $\vec{d} = (0,0,0)$ Gewichte, welche die Ausgabe $\vec{y} \approx (0.14, 0.14, 0.16)$ erzeugen, damit zwar in die Nähe der Zielwerte gelangen aber diese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

2.5.2.5 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 Continual Equilibrium Propagation sind in Abbildung 22 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 22 sind zwei Durchläufe des Lernprozesses mit den genannten Hyperparametern dargestellt. Die Zielwerte $\vec{d} = (0.3, 0.8, 0.5)$ findet das Continual Equilibrium Propagation 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 und 1 zwar annähern, diese aber nicht ohne größeren Fehler erreichen kann.

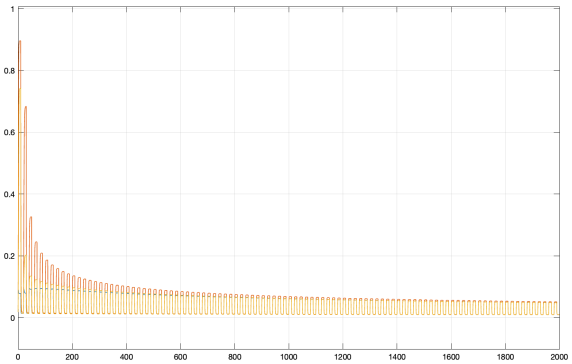
Abbildung 22: Simulationen des Continual Equilibrium Propagation mit $\beta = 1, \eta = 0.5$. Dargestellt ist die Ausgabe des Netzwerks. Quelle: *Eigene Darstellung*



(a) $\vec{d} = (0.3, 0.8, 0.5)$ **(b)** $\vec{d} = (0, 1, 0)$

In Abbildung 23 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 η 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.

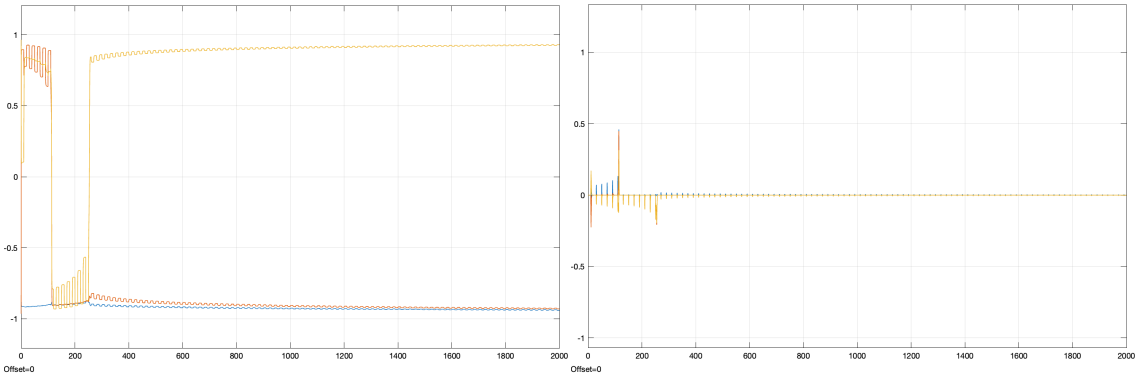
Abbildung 23: Annäherungen des Continual Equilibrium Propagation an den Grenzwert 0. Quelle: *Eigene Darstellung*



Das Continual Equilibrium Propagation 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 24 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 $\rho(x) = tanh(2x)$.

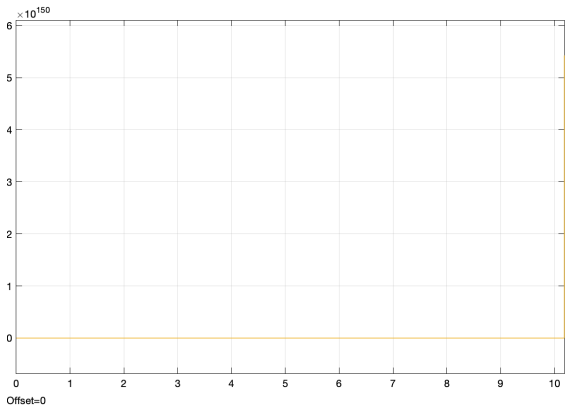
Abbildung 24: Simulationen des Continual Equilibrium Propagation mit der Aktivierungsfunktion: $\rho(x) = tanh(2x)$. Quelle: *Eigene Darstellung*



(a) Ausgabe des Netzwerks **(b) Gewichtsanpassungen**

Wie bereits im Kapitel 2.5.2.4 beschrieben, führt die Aktivierungsfunktion „ReLU“ im Zusammenhang mit dem hier implementierten Hopfield-Netzwerk zu unerwarteten Fehlern. Der schlagartige Anstieg der Zustände des Netzwerks ist in Abbildung 25 dargestellt.

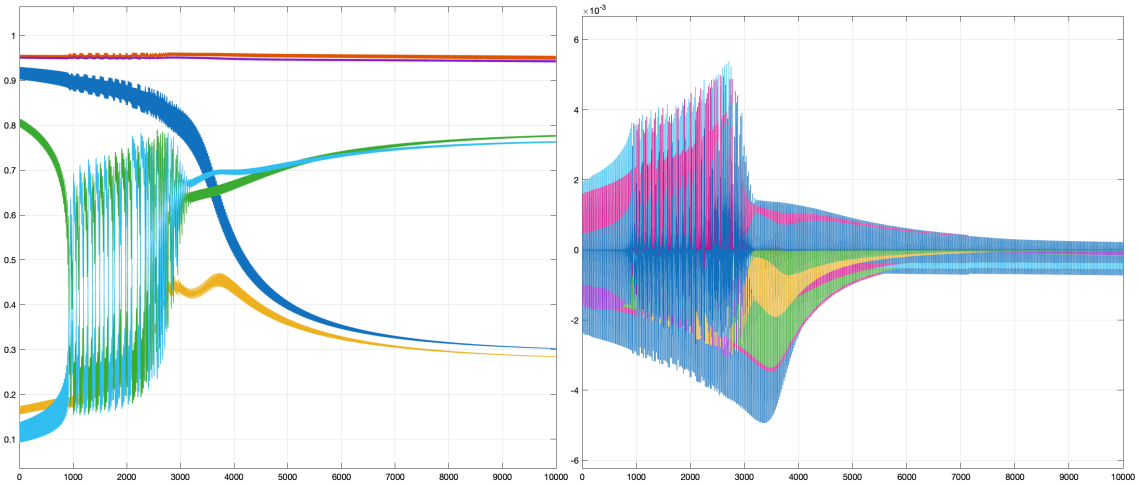
Abbildung 25: Simulation des Continual Equilibrium Propagation mit der Aktivierungsfunktion „ReLU“ ($R(x) = max(x, 0)$). Gezeigt wird die Dynamik der Zustände des Netzwerks $\frac{ds}{dt}$. Quelle: *Eigene Darstellung*



Ein fehlerfreies Speichern von N Mustern im Hopfield-Netzwerk erfordert eine Mindestanzahl von $N/0.15$ 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 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 26. 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 26: Simulationen des Continual Equilibrium Propagation mit 6 Neuronen. Quelle: *Eigene Darstellung*



(a) Ausgabe des Netzwerks **(b) Gewichtsanpassungen**

Anhang

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

Devices, *Analog* (2024): LTspice Simulator - Design Tools and Calculators, Accessed: 2024-02-01, o. O., 2024, URL: https://www.analog.com/en/resources/design-tools-and-calculators/ltspice-simulator.html

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* (2024a): Anabrid - Analog and Hybrid Computing Solutions, Accessed: 2024-02-01, o. O., 2024, URL: https://anabrid.com/

GmbH, *Anabrid* (2024b): Documentation - The Analog Thing, Accessed: 2024-02-01, o. O., 2024, URL: https://the-analog-thing.org/docs/dirhtml/

GmbH, *Anabrid* (2024c): Lucipy - Python Client for LUCIDAC, Accessed: 2024-02-01, o. O., 2024, URL: https://github.com/anabrid/lucipy

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*, *Theogarajan*, *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

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

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

Maffei, *Paolo* (2024): LTSpice Analog Computer, Accessed: 2024-02-01, o. O., 2024, URL: https://github.com/Paolo-Maffei/LTSpice-Analog-Computer

Martin, *Erwann*, *Ernoult*, *Maxence*, *Laydevant*, *Jérémie*, *Li*, *Shuai*, *Querlioz*, *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

MathWorks (2024): Simscape - Physical Modeling Environment, Accessed: 2024-02-01, o. O., 2024, URL: https://de.mathworks.com/products/simscape.html

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

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*, *Deutsch*, *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*, *Kiros*, *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*, *Aalok*, *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*, *Tootoonchian*, *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

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

Scellier, *Benjamin*, *Bengio*, *Yoshua* (2017): Equilibrium Propagation: Bridging the Gap between Energy-Based Models and Backpropagation, in: Frontiers in Computational Neuroscience, 11 (2017)

Systems, *Cadence Design* (2024): PSpice - Analog and Mixed-Signal Simulation, Accessed: 2024-02-01, o. O., 2024, URL: https://www.cadence.com/en_US/home/tools/pcb-design-and-analysis/analog-mixed-signal-simulation/pspice.html

Ulmann, *Bernd* (2022): Analog Computing, o. O.: DeGruyter, 2022

Ehrenwörtliche Erklärung

Hiermit versichere ich, dass die vorliegende Arbeit von mir selbstständig und ohne unerlaubte Hilfe angefertigt worden ist, insbesondere dass ich alle Stellen, die wörtlich oder annähernd wörtlich aus Veröffentlichungen entnommen sind, durch Zitate als solche gekennzeichnet habe. Ich versichere auch, dass die von mir eingereichte schriftliche Version mit der digitalen Version übereinstimmt. Weiterhin erkläre ich, dass die Arbeit in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde/Prüfungsstelle vorgelegen hat. Ich erkläre mich damit einverstanden/nicht einverstanden, dass die Arbeit der Öffentlichkeit zugänglich gemacht wird. Ich erkläre mich damit einverstanden, dass die Digitalversion dieser Arbeit zwecks Plagiatsprüfung auf die Server externer Anbieter hochgeladen werden darf. Die Plagiatsprüfung stellt keine Zurverfügungstellung für die Öffentlichkeit dar.

Hamburg, 17.2.2025

(Ort, Datum)

(Eigenhändige Unterschrift)