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 :	18. Dezember 2024

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

EBM Energiebasiertes Modell

Symbolverzeichnis

Glossar

Backpropagation Lernalgorithmus für MLP basierend auf dem Gradienten-Verfahren. VII, VIII, 4, 12, 13

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

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. 8, 11, 12

Equilibrium Propagation Ein Lernalgorithmus für neuronale Netzwerke, der auf Energie-Minimierung basiert und Gradienten durch Gleichgewichtszustände berechnet. 11, 12, 13

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. VII, VIII, 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. 9, 10, 12, 13, 15

Kelvin Feedback Technik Die Kelvin Feedback Technik nutzt negative Rückkopplungsschleifen in analogen Rechnern, um Differenzialgleichungen zu lösen. 7, 8

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

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. 7

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 Differenzialgleichungen, 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 Operationenverstärker 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 Operationenverstärker 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 4.4 S. 76 nachbilden

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 Operationenverstärker 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 Operationenverstärker 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, so wird die Summe der Eingangssignale über Zeit integriert. Man erhält also einen Integrierer. Durch ein Steuerelement *IC* wird ein Initialwert festgelegt, der solange ausgegeben wird, bis ein weiteres Steuerelement *OP* betätigt wird. Im aktivierten Zustand wird solange das Integral der Eingangssignale ausgegeben, bis der *OP* Schalter erneut betätigt wird und der letzte gespeicherte Wert ausgegeben wird. ebd., vgl. S. 89 ff.

Die Anwendung eines Koeffizienten am Operationenverstärker 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. *Ulmann*, *B.*, 2022, vgl. S. 92 ff.

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. ebd., vgl. S. 97 ff.

Die im ersten Augenblick einfache Multiplizierung ist auf analogen Rechnern komplex zu implementieren ebd., vgl. S. 105. Moderne analoge Rechner greifen typischerweise auf die Gilbert Zelle zurück (TODO: Quellenangabe). 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 Operationenverstärkers zu platzieren. Nach einem ähnlichen Prinzip kann auch ein Begrenzer erzeugt werden, dessen Eingangssignal zwischen einer Ober- und Untergrenze limitiert wird. ebd., S. 116

2.2.3 Aufbau von Schaltkreisen zur Lösung von Differenzialgleichungen

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

Um einen analogen Rechner mithilfe der Kelvin Feedback Technik zu konstruieren, muss die vorliegende Differenzialgleichungen 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. ebd., vgl. S. 153 ff.

Abbildung aus Ulmann2022 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, wie von ebd., S. 168 ff. beschrieben, hier 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:

Ulmann2022, Fig. 8.11. S. 170

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

Ulmann2022, Fig. 8.12. S. 170

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

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 En-

ergie 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. ebd.

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:

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

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. *Scellier*, *B.*, *Bengio*, *Y.*, 2017

2.3.3.2 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. ebd.

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:

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

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.

2.3.3.3 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:

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

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

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

Géron, *Aurélien* (2019): Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2. Aufl., o. O.: O'Reilly Media, 2019

Goldberg, *E. A.*, *Jules*, *L.* (1954): U.S. Patent No. 2,684,999, (o. O.), 1954

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

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

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*, *Graf-*

stein, *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

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)

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, 18.12.2024

(Ort, Datum)

(Eigenhändige Unterschrift)