Bachelor Thesis

at the Ludwig-Maximilians-Universität München

Numerical optimization using flow equations with applications to spin glasses and artificial neural networks

submitted by

Johannes Flommersfeld

Munich, 29 May 2017

Supervisor:
Prof. Dr. Matthias Punk
Chair of Theoretical Solid State Physics
Ludwig-Maximilians-Universität München

Bachelorarbeit

an der Ludwig-Maximilians-Universität München

Numerische Optimierung unter Verwendung von Flussgleichungen mit Anwendungen auf Spingläser und künstliche neuronale Netze

vorgelegt von

Johannes Flommersfeld

München, den 29. Mai 2017

Gutachter:
Prof. Dr. Matthias Punk
Lehrstuhl für theoretische Festkörperphysik
Ludwig-Maximilians-Universität München

Zusammenfassung

In der vorliegenden Arbeit betrachten wir ein numerisches Optimierungsverfahren unter Verwendung von Flussgleichungen und diskutieren mögliche Optimierungen des Verfahrens. Dabei erhalten wir eine Methode, die große Ähnlichkeit zum Verfahren des steilsten Abstiegs aufweist. Wir vergleichen optimierte und ursprüngliche Version der Flussgleichungsmethode bezüglich Qualität und Geschwindigkeit anhand der Suche nach Grundzuständen in Spingläsern. Außerdem wenden wir die optimierte Variante auf den Trainingsprozess künstlicher neuronaler Netze an und vergleichen sie mit dem üblicherweise verwendeten Gradientenverfahren.

Inhaltsverzeichnis

1 Einleitung

Um die Probleme der modernen Naturwissenschaften zu lösen, ist in den letzten Jahrzehnten die interdisziplinäre Zusammenarbeit zwischen den verschiedenen Fachrichtungen immer wichtiger geworden. Dies ermöglicht es, ein Problem nicht nur mit den Konzepten und Methoden einer Naturwissenschaft zu behandeln, sondern auch auf die der anderen Bereiche zurückzugreifen. So kann zum Beispiel die Studie eines physikalischen Problems zu Konzepten und Methoden führen, die auch auf Probleme der Biologie, Neurowissenschaften, Soziologie oder Ökonomie anwendbar sind und zu deren Lösung beitragen können. Gerade die Beschreibung und Nutzung komplexer Systeme stellt eine besondere Herausforderung dar, deren Lösung unterschiedlicher Ansätze und Ideen aus den verschiedenen Fachbereichen bedarf. Gleichzeitig sind komplexe Systeme aktueller Forschungsgegenstand in vielen Bereichen der Wissenschaft, die weit über die Physik hinausreichen. Besonderes Interesse gilt hierbei der Selbstorganisation und dem Ausbilden von Mustern in solchen Systemen. Aufgrund der Nichtlinearität und der großen Anzahl an Variablen in komplexen Systemen, spielen numerische Verfahren in ihrer Behandlung eine entscheidende Rolle. Aufgrund der Vielfältigkeit komplexer Systeme besitzen solche, anhand einzelner Probleme entwickelte Methoden, eine Vielzahl von Anwendungsbereichen und Gültigkeit für die verschiedensten Fragestellungen der modernen Wissenschaft. Die Erforschung und Optimierung dieser Verfahren hat also eine enorme Relevanz für die wissenschaftliche und technische Entwicklung [1, 2].

Beispiele für komplexe Systeme findet man in den unterschiedlichsten wissenschaftlichen Bereichen. So stellen beispielsweise in der Physik Spingläser ideale Modellsysteme für komplexe Systeme dar. Die Methoden und Erkenntnisse, die für die Untersuchung dieser Systeme entwickelt wurden, haben einen breiten Anwendungsbereich in den unterschiedlichsten Wissenschaften gefunden. Gerade die Suche nach Grundzuständen in einem solchen System stellt eine besondere Herausforderung dar. Approximative Verfahren spielen daher in seiner Behandlung eine wesentliche Rolle. Diese können Anwendung in verwandten Problemen, wie dem Traveling Salesman Problem finden [1].

Ein anderes, sehr interdisziplinäres Beispiel für komplexe Systeme sind künstliche neuronale Netze. Dieser biologisch motivierte Programmieransatz, die kognitiven Fähigkeiten des Gehirns nachzubilden, hat einen breiten Bereich an Anwendungen gefunden und reicht von

der Modellierung neurobiologischer Phänomene, über Mustererkennung und Approximation von Funktionen in der Mathematik sowie vielfältiger technischer Verwendung, bis hin zu Anwendungen im Bereich der Festkörperphysik, um beispielsweise Phasenzustände von Festkörpersystemen zu klassifizieren [3]. Eine der Hauptschwierigkeiten dieser neuronalen Netze besteht in ihrem Trainingsprozess. Dieser ist auf ein Optimierungsproblem zurückzuführen, wie es auch in vielen anderen Bereichen der modernen Wissenschaft zu finden ist [4, 5].

Darüber hinaus gibt es noch eine Vielzahl anderer Beispiele für Komplexität in den Wissenschaften. So stellen in der Meterologie nichtlineares Verhalten und Selbstorganisation, wie zum Beispiel das Phänomen der Rayleigh-Bénard-Konvektion, einen Schlüssel zum Verständnis der Dynamik des Wetters dar. In der Biologie spielen, bei der Frage nach der Entstehung von Leben und dem Verständnis von Evolutionsprozessen, aber auch der Proteinfaltung, Musterbildung und Selbstorganisation eine zentrale Rolle. In der Medizin stellt der Körper ein komplexes und sensibles System dar, aber auch einzelne Organe, wie zum Beispiel Herz oder Gehirn, können als komplexe Systeme aufgefasst werden. Darüber hinaus gibt es auch ökonomische, ökologische und soziologische Entwicklungen, die Beispiele für komplexe Prozesse sind [2].

In der vorliegenden Arbeit betrachten wir das von M. Punk in [6] vorgestellte numerische Optimierungsverfahren unter Verwendung von Flussgleichungen. Das Verfahren basiert auf Homotopie-Verfahren für nichtlineare Gleichungssysteme in Kombination mit dem Prinzip maximaler Entropie. Wir werden versuchen, dieses numerisch zu optimieren und zur Bestimmung von Grundzuständen in Spingläsern sowie für den Trainingsprozess neuronaler Netze zur handschriftlichen Ziffernerkennung anzuwenden. Abschließend werden wir dieses Verfahren mit den herkömmlichen Trainingsmethoden neuronaler Netze vergleichen.

2 Mathematische Grundlagen und die Flussgleichungsmethode

Bevor wir mögliche Optimierungen oder Anwendungen des Verfahrens diskutieren können, müssen wir zunächst auf die mathematischen Grundlagen der Flussgleichungsmethode, diese selbst und ihre numerische Lösung mithilfe des Heun-Verfahrens eingehen.

2.1 Homotopie-Verfahren zur Lösung nichtlinearer Gleichungssysteme

Numerische Verfahren zum Lösen nichtlinearer Gleichungssysteme, wie z.B. das Newton-Verfahren besitzen typischerweise eine große Abhängigkeit von den gewählten Startwerten. Besitzt man nur wenig oder gar keine Information über die Lösung des Problems, stellt die Wahl der richtigen Startwerte jedoch eine große Herausforderung dar. Homotopie-Verfahren bieten einen Ansatz, diese Abhängigkeit zu reduzieren, indem man die Lösung eines bekannten Systems nutzt, um auf die Lösung des unbekannten Systems rückzuschließen.

Ist man an der Lösung eines Systems von N nichtlinearen Gleichungen

$$F(x) = 0, \tag{2.1}$$

mit $F : \mathbb{R}^N \to \mathbb{R}^N$ interessiert und kennt die Lösung des Systems

$$G(x) = 0, \tag{2.2}$$

wobei $G : \mathbb{R}^N \to \mathbb{R}^N$, so sucht man eine Homotopie $H : \mathbb{R}^N \times [0, 1] \to \mathbb{R}^N$ mit den folgenden Eigenschaften:

$$H(x, 0) = G(x), \qquad H(x, 1) = F(x) \tag{2.3}$$

Wobei H stetig und differenzierbar ist. Dies gelingt z.B. durch die folgende Form von H:

$$H(x, \lambda) = \lambda F(x) + (1 - \lambda)G(x) \tag{2.4}$$

Man folgt nun der so definierten Kurve, startend bei einer Lösung von $G(x)$ bei $\lambda = 0$ hin zu einer Lösung von $F(x)$ bei $\lambda = 1$. Wir werden in Abschnitt 2.4 sehen, dass mithilfe des Satzes von Bayes und dem Prinzip der maximalen Entropie, eine Homotopie von der Form in Gleichung (2.4) hergeleitet werden kann [7].

2.2 Bayessche Statistik

Wir suchen ein Verfahren, mit dem wir die Parameter eines Systems so anpassen können, dass es einen von uns gewünschten Zustand annimmt. Für die Suche nach Grundzuständen in Festkörpersystemen wird dafür die Energie, im Fall des Trainings neuronaler Netze die Fehlerfunktion, minimiert. Je nachdem an welchem System und an welchen Eigenschaften dieses Systems man interessiert ist, kann aber auch jedes andere Funktional verwendet werden.

Für diese Anpassung der Parameter eignet sich die Bayessche Statistik. Hierbei verwendet man sein bestehendes Wissen über die Parameter f, die sogenannten *a-priori* Verteilung $p(f)$ und das Wissen über das System, in unserem Fall ausgedrückt durch das Funktional $\mathcal{E}$, die sogenannte Likelihood $p(\mathcal{E}|f)$. Mithilfe des Satzes von Bayes lässt sich daraus wie folgt auf die neue Verteilung der Parameter, die *a-posteriori* Verteilung $p(f|\mathcal{E})$ schließen: [8]

$$p(f|\mathcal{E}) \propto p(\mathcal{E}|f)p(f) \tag{2.5}$$

Die Schwierigkeit besteht nun darin, eine sinnvolle *a-priori* Verteilung zu finden. Einen Ansatz für dieses Problem bietet das Prinzip der maximalen Entropie.

2.3 Prinzip der maximalen Entropie

Die a-priori Verteilung sollte das gesamte vorhandene Wissen über die Parameter, darüber hinaus jedoch keine zusätzlichen Annahmen enthalten. Um dies zu gewährleisten benötigt man ein Informationsmaß. C. Shannon zeigte in [9], dass ein solches Maß die gleiche mathematische Form besitzt, wie die Entropie. Wie E. Jaynes in [10] zeigte, liefert dies einen neuen Zugang zur statistischen Physik und ermöglicht es die genannten Bedingungen an die a-priori Verteilung zu erfüllen, indem man diejenige Verteilung wählt, welche die Entropie maximiert. J. Skilling zeigte in [11], dass einige wenige einfache Beispiele die Form der a-priori Verteilung bis auf eine Konstante κ eindeutig zu

$$p(f) \propto \exp(\kappa S[f(x)]) \tag{2.6}$$

festlegen. Wobei $f(x)$ eine positive, additive Funktion der reellen Variable x sein muss. Dies stellt jedoch für die weitere Diskussion keine Einschränkung dar. Die Entropie S hat dabei die Form [12]

$$S[f] = \int dx \left(f(x) - f_0(x) - f(x) \ln \frac{f(x)}{f_0(x)} \right). \tag{2.7}$$

Hierbei stellt $f_0(x)$ die ursprüngliche Wahl von f dar.

2.4 Flussgleichungsmethode

Wir werden nun die Ergebnisse der vorherigen Abschnitte nutzen, um eine Flussgleichung für die Parameter f herzuleiten und mithilfe dieser das Funktional $E[f]$, welches beispielsweise die Energie des Systems darstellen kann, bezüglich f zu minimieren. Um Gleichung (2.5) verwenden zu können, benötigen wir zunächst noch die Likelihood, also die Wahrscheinlichkeit eine bestimmte Energie $\mathcal{E}$ für gegebene f zu erhalten. Diese ist gegeben durch $p(\mathcal{E}|f) = \delta(\mathcal{E} - E[f])$, was sich mithilfe der Exponentialfunktion der Breite σ wie folgt ausdrücken lässt:

$$p(\mathcal{E}|f) = \lim_{\sigma \to 0} \frac{\exp(-|E[f(x)] - \mathcal{E}|/\sigma)}{2\sigma} \tag{2.8}$$

Um die Parameter f des Grundzustandes zu bestimmen, wollen wir nun die Wahrscheinlichkeit $p(\mathcal{E}_{\min}|f)$ maximieren, wobei $\mathcal{E}_{\min} = \min_f E[f]$ die Energie des Grundzustandes darstellt. Aus den Gleichungen (2.5), (2.6) und (2.8) folgt, dass man dies erreicht, indem man das Funktional

$$Q[f(x); t] = E[f(x)]t - S[f(x)](1 - t) \tag{2.9}$$

minimiert. Dabei wurde $\kappa = (1 - t)/(t\sigma)$ gewählt. Der Parameter $t \in [0, 1]$ gewichtet die relative Bedeutung der *a-priori* Verteilung und der Likelihood zueinander. Man erkennt deutlich, dass es sich bei $Q[f(x); t]$ um eine Homotopie wie in Gleichung (2.4) handelt. Lassen wir also t von 0 bis 1 laufen, erwarten wir bei $t = 1$ ein Minimum der Energie zu finden.

Wie in [6] gezeigt wird, erhält man durch Minimierung von $Q[\{f_n\}; t]$ die folgende diskrete Form einer Flussgleichung:

$$\frac{\partial f_n}{\partial t} = -\sum_m \left(\frac{1 - t}{f_n} \delta_{m,n} + t \frac{\partial^2 E}{\partial f_m \partial f_n} \right)^{-1} \frac{\partial E}{\partial f_m} \tag{2.10}$$

Dabei wurde die Abhängigkeit von der ursprünglichen Wahl der Parameter $f_0(x)$ eliminiert, indem während des Flusses kontinuierlich $f_0(x) = f(x)$ gesetzt wird. Dies hat jedoch zur Folge, dass nicht mehr garantiert ist, dass bei $t = 1$ auch ein Minimum von E erreicht wird.

Man muss die Parameter also unter Umständen mehrfach fließen lassen. Die Flussgleichung besitzt Fixpunkte bei $\partial E/\partial f_n = 0$, wodurch garantiert ist, dass die Parameter f_n solange fließen, bis ein Extremum von E erreicht ist.

Der numerisch aufwendigste Schritt beim Lösen dieser Flussgleichung ist die Invertierung der Hesse-Matrix. Im Falle neuronaler Netze ist zusätzlich auch schon die Berechnung der zweiten Ableitungen sehr aufwendig. Wir werden deshalb in Abschnitt 4 einige Möglichkeiten diskutieren, diese Schritte zu optimieren.

2.5 Heun-Verfahren

Gleichung (2.10) stellt ein Anfangswertproblem (AWP) dar, welche im Allgemeinen von der Form

$$\dot{y} = g(t, y), \quad y(t_0) = y_0 \tag{2.11}$$

sind. AWPs lassen sich unter anderem mit den sogenannten Heun-Verfahren lösen. Dabei handelt es sich um ein Predictor-Corrector-Verfahren. Für den Predictor Schritt schätzt man dabei die Ableitung der Funktion y durch

$$\frac{y(t+h) - y(t)}{h} \approx \dot{y}(t) = g(t, y) \tag{2.12}$$

ab. Durch Umformen nach $y(t+h)$ erhält man daraus

$$y(t+h) \approx y(t) + hg(t, y). \tag{2.13}$$

Dabei handelt es sich um das explizite Euler-Verfahren. Dieses liefert jedoch in vielen Fällen keine zufriedenstellenden Ergebnisse. Deshalb ergänzt man es um einen zusätzlichen Corrector Schritt. Für diesen nutzen wir

$$y(t+h) - y(t) = \int_t^{t+h} dt' g(t', y) \tag{2.14}$$

und schätzen das in Gleichung(2.14) auftretende Integral mithilfe der Trapezregel

$$\int_t^{t+h} dt' g(t', y) \approx h\frac{g(t, y) + g(y(t+h), t+h)}{2} \tag{2.15}$$

ab. Aus den Gleichungen (2.14) und (2.15) folgt

$$y(t+h) = y(t) + \frac{h}{2}(g(t, y) + g(y(t+h), t+h)). \tag{2.16}$$

Hierbei ist jedoch auch die rechte Seite noch von $y(t+h)$ abhängig. Dazu verwenden wir den Predictor Schritt und erhalten damit die folgende Iterationsvorschrift:

$$\tilde{y}_{i+1} = y_i + hg(y_i, t_i) \tag{2.17}$$

$$y_{i+1} = y_i + \frac{h}{2}(g(y_i, t_i) + g(\tilde{y}_{i+1}, t_{i+1})). \tag{2.18}$$

Mit $t_{i+1} = t_i + h$ und $y_i = y(t_i)$ [13].

3 Grundlagen künstlicher neuronaler Netze

Das in Abschnitt 2.4 dargestellte Verfahren werden wir im Laufe der Arbeit verwenden, um damit künstliche neuronale Netze (KNN) zu trainieren und diese zur Klassifizierung handgeschriebener Ziffern zu verwenden. Deshalb beschäftigen wir uns zunächst mit der grundlegenden Funktionsweise und dem Trainingsprozess neuronaler Netze.

3.1 Aufbau künstlicher neuronaler Netze

Um den Lernprozess eines KNNs nachvollziehen zu können, müssen wir zunächst den prinzipiellen Aufbau und die Funktionsweise eines solchen Netzes verstehen.

3.1.1 Künstliche Neuronen

Dazu betrachten wir zunächst seine Bestandteile, die Neuronen. Es gibt verschiedene Modelle für künstliche Neuronen, das grundlegende Prinzip ist aber in allen Fällen identisch und wird in Abbildung 3.1 schematisch dargestellt. Ein Neuron erhält eine Reihe von Eingabedaten und erzeugt daraus einen einzelnen Ausgabewert. Dieser Wert wird durch die genaue Form der sogenannten Aktivierugsfunktion, den Gewichten und dem Schwellenwert des Neurons bestimmt.

Die Gewichte w_j des Neurons sind reelle Zahlen und legen fest, welche Rolle eine bestimmte Eingabe für den Ausgabewert spielt. Dies geschieht mithilfe der gewichteten Summe $\sum_j w_j X_j$ der Eingabewerte X_j. Der Begriff Schwellenwert s macht hauptsächlich für Neuronen mit binärer Ausgabe, den sogenannten Perzeptronen, Sinn. Wir werden ihn deshalb anhand dieser definieren und dann mit einer etwas veränderten Größe, dem Bias-Parameter, arbeiten.

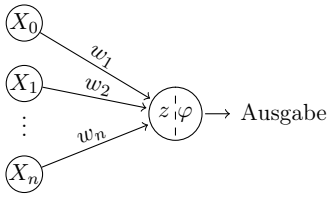

Abbildung 3.1: Schematische Darstellung eines künstlichen Neurons. Mit den Eingabewerten X_n, den Gewichten w_n, der Übertragungsfunktion z und der Aktivierungsfunktion φ. Die Ausgabe ergibt sich aus $\varphi(z(w, X))$.

Ein Perzeptron hat nur zwei Ausgabewerte, 0 und 1. Ist der Ausgabewert 1, so feuert das Neuron, ist er 0 so feuert es nicht. Überschreitet die gewichtete Summe den Schwellenwert des Neurons, so feuert es. Ist dies nicht der Fall, so wird 0 zurückgegeben. Mathematisch lässt sich dies wie folgt darstellen:

$$Ausgabe = \begin{cases} 0 & \text{falls } \sum_j w_j X_j \leq s \\ 1 & \text{falls } \sum_j w_j X_j > s \end{cases} \tag{3.1}$$

Gleichung (3.1) lässt sich auch mithilfe der Heaviside-Funktion Θ ausdrücken:

$$Ausgabe = \Theta\left(\sum_j w_j X_j - s\right) \tag{3.2}$$

Anhand von Gleichung (3.2) lässt sich nun leicht eine mögliche Verallgemeinerung erkennen. Möchte man für den Ausgabewert kontinuierliche Werte zwischen 0 und 1 erhalten, so lässt sich dies umsetzen, indem man statt der Heaviside-Funktion einfach eine beliebige andere beschränkte Funktion verwendet. Diese bezeichnet man als Aktivierungsfunktion. Dies ist sinnvoll, da die Ableitung der Heaviside-Funktion, wie in Abbildung 3.2 zu erkenne ist, singulär ist, was eine Optimierung mithilfe der Flussgleichungsmethode unmöglich macht.

In diesem Zusammenhang macht nun aber der Begriff Schwellenwert keinen wirklichen Sinn mehr, da ein solches Neuron nicht mehr einfach nur feuert oder nicht feuert, sondern beliebige Werte zwischen 0 und 1 ausgeben kann. Deshalb definiert man den vorher erwähnten Bias-Parameter b als $b = -s$ und erhält somit die folgende mathematische Charakterisierung eines künstlichen Neurons:

$$Ausgabe = \varphi\left(\sum_j w_j X_j + b\right) := \varphi(z) \tag{3.3}$$

Wobei die Summe $z = \sum_j w_j X_j + b$ als Übertragungsfunktion z bezeichnet wird. Für eine einfachere Darstellung und eine einheitlichere Form, vor allem der Ableitungen des Netzes,

bietet es sich an, den Bias-Parameter ebenfalls als Gewicht darzustellen. Dies erreicht man, indem man die Eingabewerte X_j um einen zusätzlichen, fixierten Wert $X_k = 1$ erweitert und für das entsprechende Gewicht $w_k = b$ wählt. Damit erhält man für die Übertragungsfunktion:

$$z = \sum_j w_j X_j \tag{3.4}$$

In dieser Arbeit werden wir zwei verschiede Aktivierungsfunktionen verwenden. Zunächst betrachten wir die sogenannte Sigmoidfunktion oder logistische Funktion

$$\sigma(z) = \frac{1}{1 + e^{-z}}. \tag{3.5}$$

Diese stellt, wie in Abbildung 3.2 deutlich zu erkennen, eine geglättete Heaviside-Funktion dar. Ein solches Neuron (Sigmoid) zeigt also für die Grenzwerte $z \to \pm\infty$ ein identisches Verhalten zu dem des Perzeptrons.

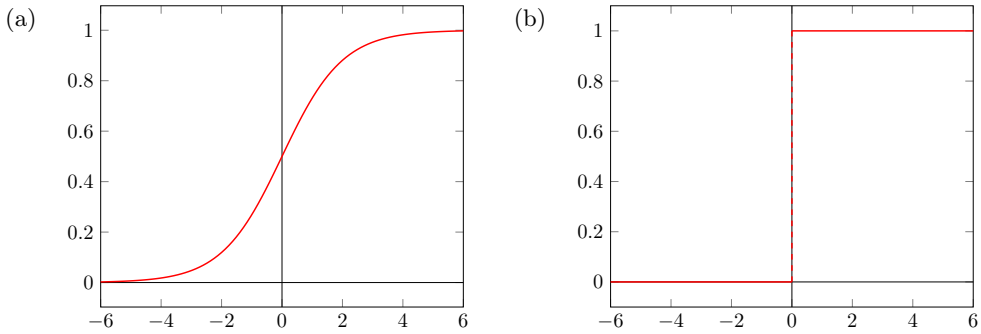

Abbildung 3.2: Unterschiedliche Aktivierungsfunktionen: (a) die Heaviside-Funktion eines Perzeptrons, (b) die logistische Funktion eines Sigmoids

Wie McCulloch und Pitts in [14] zeigten, lässt sich mit der Kombination mehrerer Perzeptronen jede einfache logische Funktion darstellen. Wie erwähnt ist es jedoch nicht möglich, ein Netz aus Perzeptronen mithilfe der Flussgleichungsmethode zu trainieren. Aufgrund der identischen Grenzwerte der beiden in Abbildung 3.2 dargestellten Aktivierungsfunktionen, ist es möglich das grundsätzliche Verhalten der Perzeptronen auch mithilfe von sigmoidalen Neuronen zu erzeugen.

Die zweite Aktivierungsfunktion, die wir verwenden werden, ist die sogenannte Softmax Funktion:

$$a_j = \frac{e^{z_j}}{\sum_k e^{z_k}} \tag{3.6}$$

Wobei a_j die Aktivierung des j-ten Neurons in einer Schicht aus k Neuronen ist und z_k die zugehörigen Übertragungsfunktionen sind. Die Anordnung der Neuronen in Schichten und die Motivation für die Verwendung der Softmax Funktion wird im Abschnitt 3.1.2 genauer erläutert. Wie anhand von Gleichung (3.6) leicht zu erkennen ist, gibt die Softmax Funktion immer einen Wert im Intervall $[0, 1]$ zurück und die Summe über alle a_j ergibt immer 1. Außerdem führen große z_j zu großen, kleine z_j zu geringeren Aktivierungen. In diesem Sinne lässt sich die Softmax Funktion als geglättete Maximumsfunktion interpretieren, wie der Name bereits andeutet [5, 15].

3.1.2 Grundlegende Architektur neuronaler Netze

Die im vorherigen Abschnitt vorgestellten künstlichen Neuronen lassen sich zu einem neuronalen Netz kombinieren. Dabei werden die Neuronen, wie in Abbildung 3.3 dargestellt, in verschiedenen Schichten angeordnet. Eine besondere Rollen spielen die erste bzw. letzte

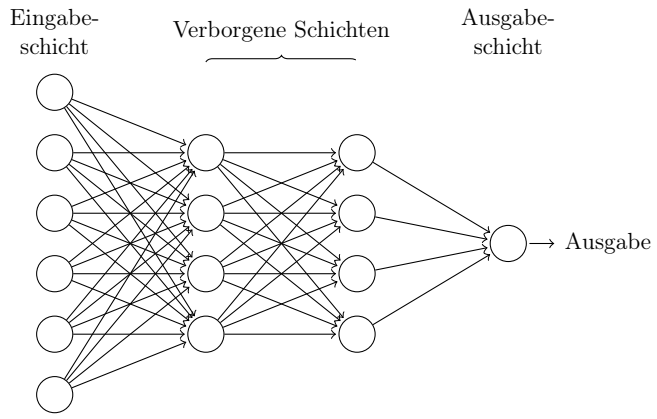

Abbildung 3.3: Schematische Darstellung eines KNNs. Dieses lässt sich in Eingabe-, Ausgabe- und verdeckte Schicht aufteilen.

Schicht, die sogenannte Ein- bzw. Ausgabeschicht. Die Neuronen in der ersten Schicht sind im Sinne der Definition in Abschnitt 3.1.1 keine richtigen Neuronen, da sie keine Eingabe besitzen. Ein Neuron ohne Eingabewerte gibt, wie anhand von Gleichung (3.3) zu erkennen ist, einfach einen festen Wert zurück. Bei der Eingabeschicht handelt es sich also einfach um die Eingabedaten selber. Die Ausgabewerte der Neuronen in der letzten Schicht dienen keiner neuen Schicht als Eingabewerte. Sie stellen die Ausgabe des neuronalen Netzes dar. Wie diese Ausgabe im Detail aussieht, ist an die Problemstellung anzupassen, auf die das neuronale Netz angewandt wird. Für die Klassifizierung handschriftlicher Ziffern bietet sich eine Ausgabeschicht aus 10 Neuronen an, wobei jedes Neuron eine Ziffer darstellt. Das Neuron mit der höchsten Aktivierung entspricht dann dem Ergebnis des Netzes.

Für viele Anwendungen ist es nützlich, die Ausgabe der letzen Schicht als Wahrscheinlichkeitsverteilung aufzufassen. In unserem Beispiel also die Wahrscheinlichkeit, dass auf einem Bild eine bestimmte Ziffer abgebildet ist. Dies ist mit einer Sigmoid-Ausgabeschicht jedoch nicht möglich, da die einzelnen Aktivierungen zwar immer zwischen 0 und 1 liegen, die Summe über die verschiedenen Aktivierungen aber nicht notwendigerweise 1 ergeben muss. Um dieses Problem zu lösen, verwendet man häufig eine Ausgabeschicht mit der in Gleichung (3.6) eingeführten Softmax Funktion als Aktivierung. Diese bringt darüber hinaus auch noch einige andere Vorteile. In den meisten Fällen wird die Softmax Aktivierungsfunktion ausschließlich in der Ausgabeschicht verwendet. Das liegt zum einen daran, dass die Interpretation als Wahrscheinlichkeitsverteilung in den verborgenen Schichten nicht notwendig ist. Zum anderen hat eine Änderung an einem Neuron in einer Softmaxschicht zwangsläufig Auswirkungen auf alle anderen Neuronen in dieser Schicht, da die Summe über alle Aktivierungen weiterhin 1 ergeben muss. Diese zusätzlichen Abhängigkeiten versucht man zu vermeiden. Für verborgene Schichten werden deshalb üblicherweise sigmoidale Neuronen verwendet.

Die Schichten zwischen der Ein- und der Ausgabeschicht werden als verborgene Schicht bezeichnet. Dienen in diesen Schichten die Ausgabewerte der Neuronen in der Schicht n-1 ausschließlich als Eingabewerte der Neuronen in der n-ten Schicht, so spricht man von einem feedforward-Netz. Im Gegensatz dazu gibt es auch rekurrente Netze, in denen eine Rückkopplung stattfinden kann. Sie kommen damit der Funktionsweise eines menschlichen Gehirns wesentlich näher, sind jedoch deutlich schwieriger zu trainieren. Im Rahmen dieser Arbeit werden wir uns auf feedforward-Netze konzentrieren, rekurrente Netze seien somit nur der Vollständigkeit halber erwähnt.

Mehrere verborgene Schichten ermöglichen das Lösen von komplexeren Aufgaben, jedoch wird dadurch der Lernprozess deutlich erschwert. Ähnliches gilt für die Anzahl der Neuronen pro Schicht. Mehr Neuronen in einer Schicht verbessern unter Umständen die Ergebnisse des Netzes, jedoch wird dadurch der Lernprozess aufwendiger. Für die genaue Architektur eines neuronalen Netzes gibt es keine exakten, allgemein gültigen Regeln [5]. Wir werden in Abschnitt 6 verschiedene Architekturen kennen lernen und diese miteinander vergleichen.

3.2 Lernprozess neuronaler Netze

In Abschnitt 3.1 haben wir gesehen, wie ein KNN aufgebaut ist und funktioniert. Jedoch bleibt noch die Frage offen, wie die optimale Wahl der Parameter erfolgt. Das geschieht im Lernprozess des neuronalen Netzes. Im Wesentlichen gibt es zwei Arten von Lernprozessen, überwachtes und unüberwachtes Lernen. Für die Anwendung in dieser Arbeit spielt un-

überwachtes Lernen keine Rolle. Wir werden deshalb im Folgenden nur auf das überwachte Lernen eingehen.

3.2.1 Überwachtes Lernen mithilfe der Fehlerfunktion

Beim überwachten Lernen werden ein Satz von Trainingsdaten x_i mit den dazugehörigen korrekten Ergebnissen $y(x_i)$ verwendet. Ziel des Trainingsprozesses ist es, die Parameter des Netzes so anzupassen, dass die Ausgabe des Netzes möglichst genau mit den $y(x_i)$ übereinstimmt. Dies ist am besten umzusetzen, wenn eine kleine Änderung der Parameter auch nur kleine Änderungen in der Ausgabe des Netzes erzeugen. Hier erkennen wir nun den Vorteil stetiger Aktivierungsfunktionen. Eine Unstetigkeit wie in Gleichung (3.2) führt dazu, dass kleine Änderungen der Gewichte eine völlig andere Ausgabe verursachen können. Dieses chaotische Verhalten führt zu großen Schwierigkeiten, einen konvergierenden Lernalgorithmus zu finden. Somit verwendet man, trotz der Analogie zwischen Perzeptronen und biologischen Nervenzellen, in den meisten Fällen Neuronen mit stetigen Aktivierungsfunktionen.

Um nun herauszufinden, welche Modifikationen der Parameter das KNN verbessern, führen wir die Fehlerfunktion $C(w)$ als quantitatives Maß für die Qualität des neuronalen Netzes ein. Man kann verschiedene Formen der Fehlerfunktion wählen. Da sich anhand des mittleren quadratischen Fehlers die charakteristischen Eigenschaften einer Fehlerfunktion besonders gut erkennen lassen, verwenden wir zunächst diesen als Fehlerfunktion:

$$C(w) = \frac{1}{2n} \sum_x ||y(x) - a(w, b, x)||^2 \tag{3.7}$$

Hierbei stellt n die Gesamtzahl der Trainingsdaten und $a(w, x)$ die Ausgabe des Netzes dar. Man erkennt an Gleichung 3.7 deutlich, dass die Fehlerfunktion verschwindet, wenn die Ausgaben des KNNs mit den korrekten Ergebnissen übereinstimmen. Gleichzeitig gilt $C(w) \geq 0$ für alle w.

Verwendet man die Softmax Funktion in der Ausgabeschicht, kann man die Aktivierungen, wie zuvor erläutert, als Wahrscheinlichkeitsverteilung interpretieren. Dies ermöglicht uns nun einen allgemeineren Ansatz zum Anpassen der Parameter des Netzes. Nach der Maximum-Likelihood-Methode wählt man diejenigen Parameter, die die Wahrscheinlichkeit für die beobachteten Daten maximiert. Im Fall der Ziffernerkennung wird also die Wahrscheinlichkeit der korrekten Ziffer maximiert. Zu diesem Zweck führen wir die Log-Likelihood Fehlerfunktion ein:

$$C(w) = -\frac{1}{n} \sum_x \ln\left(a_y^L\right) = -\frac{1}{n} \sum_x \ln\left(\frac{e^{z_y^L}}{\sum_k e^{z_k^L}}\right) \tag{3.8}$$

Hierbei sind $y \in [0, 9]$ die in entsprechenden Trainingsdaten x dargestellte Ziffern. a_y^L ist also die Aktivierung des zu dieser Ziffer gehörenden Neurons in der Ausgabeschicht L. z_y^L ist die zugehörige Übertragungsfunktion, n stellt wie üblich die Anzahl der Trainingsdaten dar.

Ist das Netz in der Lage, den Zahlenwert y des entsprechenden Bildes korrekt zu klassifizieren, so nimmt a_y^L ein Wert nahe 1 an und die Fehlerfunktion wird gering. Ist das Ergebnis des Netzes sehr schlecht, so liegt a_y^L nahe der 0, die Fehlerfunktion wird sehr groß. Aufgrund des Vorzeichens ist gewährleistet, dass $C(w) \geq 0$ für alle w gilt. Die negative Log-Likelihood Funktion hat also alle Eigenschaften, die wir von einer Fehlerfunktion erwarten [5, 15].

Aus der vorangegangenen Diskussion ist klar geworden, dass das Ziel des Trainingsprozesses sein wird, die Fehlerfunktion zu minimieren. Diese ist eine hochdimensionale Funktion, die aufgrund der nichttrivialen Abhängigkeit der Ausgabe a von w eine Vielzahl von lokalen Minima besitzen kann. Wir haben also den Lernprozess des neuronalen Netzes auf ein nichttriviales Optimierungsproblem zurückgeführt. Üblicherweise versucht man dieses mit einem einfachen Gradientenverfahren zu behandeln. Wir werden dieses im späteren Verlauf der Arbeit durch die Flussgleichungsmethode ersetzen.

3.2.2 Berechnung des Gradienten der Fehlerfunktion

Sowohl für einfache Gradientenverfahren als auch für die Flussgleichungsmethode werden wir den Gradienten der Fehlerfunktion brauchen. Durch Anwendung der Kettenregel erhält man

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \frac{\partial C}{\partial z_j^l}. \tag{3.9}$$

Dabei stellt w_{jk}^l das Gewicht des k-ten Neurons in der Schicht $l-1$ zum j-ten Neuron in der l-ten Schicht und a_k^{l-1} die Aktivierung des k-ten Neurons in der Schicht $l-1$ dar. Für $\partial C/\partial z_j^l$ erhält man durch erneute Anwendung der Kettenregel und Gleichung (3.4) die folgende Form:

$$\frac{\partial C}{\partial z_j^l} = \sum_k \frac{\partial C}{\partial z_j^{l+1}} w_{kj}^{l+1} \sigma'(z_j^l) \tag{3.10}$$

Dabei stellt $\sigma'(z_j^l) = \partial\sigma(z_j^l)/\partial z_j^l$ die Ableitung der Aktivierungsfunktion dar, welche problemlos berechnet werden kann. Anhand von Gleichung (3.10) erkennt man, dass die partiellen Ableitungen in der Schicht l mithilfe der Ableitungen aus der Schicht $l+1$ berechnet werden können. Sind also die Ableitungen der Ausgabeschicht bekannt, so lässt sich der gesamte Gradient iterativ berechnen. Mithilfe der Kettenregel erhält man für die Ableitungen

der Ausgabeschicht L die folgende Formel:

$$\frac{\partial C}{\partial z_j^L} = \frac{\partial C}{\partial a_j^L}\sigma'(z_j^L) \tag{3.11}$$

Alle Terme in Gleichung (3.11) lassen sich elementar berechnen, dabei müssen noch die verschiedenen Aktivierungsfunktionen im neuronalen Netz sowie die verwendete Fehlerfunktion berücksichtigt werden. Für die in Abschnitt 3.2.1 erläuterte Kombination aus Softmax Funktion in der Ausgabeschicht und der Log-Likelihood Fehlerfunktion erhält man für Gleichung (3.11):

$$\frac{\partial C}{\partial z_j^L} = \frac{1}{n}\sum_x (a_j^L - y_j) \tag{3.12}$$

In den verborgenen Schichten wird die Sigmoidfunktion verwendet und man erhält somit für die Ableitung der Aktivierungsfunktion:

$$\sigma'(z_j^l) = a_j^l(1 - a_j^l) \tag{3.13}$$

Damit lässt sich nun der gesamte Gradient berechnen, indem man zunächst die Eingabedaten durch das Netz propagieren lässt, um die Werte der a_j^l zu erhalten und dann das Netz rückwärts durchläuft, wobei man iterativ die einzelnen Ableitungen berechnet. Dieses Verfahren zur Berechnung des Gradienten in einem KNN wird auch als Backpropagation bezeichnet, da es als Rückführung des Fehlers des Netzes interpretiert werden kann [5].

3.2.3 Effiziente Berechnung mithilfe stochastischer Approximation

Sowohl der mittlere quadratische Fehler als auch die Log-Likelihood Fehlerfunktion lassen sich in der Form

$$C(w) = \frac{1}{n}\sum_x C_x \tag{3.14}$$

darstellen. Wobei über alle n verschiedenen C_x, den Fehlerfunktionen der einzelner Trainingsbeispiele, gemittelt wird. Daraus folgt, dass auch der Gradient wie folgt schreiben lässt:

$$\frac{\partial C}{\partial w} = \frac{1}{n}\sum_x \frac{\partial C_x}{\partial w} \tag{3.15}$$

Da man versucht die Zahl der Trainingsdaten x möglichst groß zu wählen, um das KNN zu verbessern, wie in Abschnitt 3.3.1 genauer erläutert wird, stellt die Berechnung aller $\frac{\partial C_x}{\partial w}$ einen erheblichen Aufwand dar. Eine Möglichkeit, dieses Problem zu umgehen und die Berechnung des Gradienten damit zu beschleunigen, ist die stochastische Approximation.

Dabei schätzt man den Gradienten der gesamten Fehlerfunktion ab, indem man nur über eine Teilmenge der Trainingsdaten, einen sogenannten Mini-Batch, mittelt. Um jedoch weiterhin nicht nur einen Teil der Daten für den Lernprozess zu verwenden, wird nicht das gesamte Training mit einer Teilmenge durchgeführt. Stattdessen verwendet man einen zufällig ausgewählten Trainingssatz aus, führt mit diesem einen Teil des Trainingsprozesses durch und wählt dann einen anderen Satz aus, um mit diesem weiter zu trainieren. So verfährt man solange, bis alle zur Verfügung stehenden Daten verwendet wurden. Dies nennt man einen Trainingsepoch. Nach Beendigung eines Epochs werden die Trainingsdaten zufällig neu zu Mini-Batches angeordnet und das Training von Vorne begonnen [5].

3.2.4 Berechnung der Hesse-Matrix der Fehlerfunktion

Für die Flussgleichungsmethode in Form von Gleichung (2.10) sowie verschiedene andere fortgeschrittene Optimierungsverfahren ist es nötig, auch die zweiten Ableitungen der Fehlerfunktion zu berechnen. Dies ist jedoch aufgrund der hohen Zahl an Parametern im neuronalen Netz mit einem erheblichen numerischen Aufwand verbunden [5]. Wie wir im Laufe der Arbeit sehen werden, erreichen wir auch unter Vernachlässigung der Hesse-Matrix Ergebnisse von über 99% für die Klassifizierung handgeschriebener Ziffern. Aus diesem Grund werden wir in dieser Arbeit nicht näher auf die Berechnung höherer Ableitungen in KNNs eingehen. Es besteht aber natürlich die Möglichkeit, dass es Anwendungen neuronaler Netze gibt, für die die Berechnung der Hesse-Matrix Sinn macht. Dies ist z.B. nach dem in [16] vorgestellten Verfahren exakt möglich. Aufgrund des hohen numerischen Aufwands sind vor allem Näherungsverfahren von besonderem Interesse, wie sie z.B. in [17] dargestellt werden.

3.3 Probleme beim Training neuronaler Netze

Der Trainingprozess neuronaler Netze weist verschiedene Schwierigkeiten auf und stellt dadurch die größte Herausforderung bei der Verwendung eines KNNs dar. Eine wichtiges Ziel der vorliegenden Arbeit ist es, diesen mithilfe der Flussgleichungsmethode zu verbessern. Deshalb fassen wir in diesem Abschnitt einige der bedeutendsten Probleme, sowie mögliche Lösungsansätze zusammen. Außerdem betrachten wir, inwieweit wir von der Verwendung der Flussgleichungsmethode Verbesserungen erwarten und welche negativen Effekte vermieden werden sollten.

3.3.1 Überanpassung

Überanpassung betrifft besonders Modelle, die eine große Anzahl freier Parameter besitzen. Die große Menge an Variablen führt dazu, dass solche Modelle zwar das Problem, anhand dessen sie entwickelt wurden, sehr gut beschreiben können, jedoch scheitern, wenn sie auf neue oder verallgemeinerte Probleme angewandt werden. KNNs besitzen eine extrem große Anzahl solcher freien Parameter, gleichzeitig will man aber eine Überanpassung unbedingt vermeiden. Schließlich trainiert man neuronale Netze mit Trainingsdaten, um sie dann auf neue Probleme anzuwenden. Die korrekte Klassifikation der Trainingsdaten ist hierbei nicht das Ziel, sondern dient nur als Hilfsmittel. Es macht somit oftmals gar keinen Sinn, das globale Minimum der Fehlerfunktion zu finden, da dies zwar einer perfekten Klassifikation der Trainingsdaten entspräche, das Netz jedoch nicht mehr so gut in der Lage wäre neue Daten korrekt einzuordnen. Es hätte somit im Prinzip nur die Trainingsdaten auswendig gelernt und würde damit keinen Mehrwert liefern. Es gibt verschiedene Strategien, um diese Situation zu verhindern.

Zahl der Trainingsdaten

Das wohl einfachste und effektivste Mittel gegen Überanpassung ist die Anzahl der Trainingsdaten. Je größer die Zahl der verwendeten Trainingsdaten, desto besser verallgemeinert das Netz üblicherweise. Verwendet man nur wenige Trainingsdaten, so wird das Netz in den meisten Fällen kein Problem damit haben, diese sehr gut zu erlernen und die Fehlerfunktion damit zu minimieren, jedoch hat es dann typischerweise größere Probleme beim Lösen eines neuen Problems. Stehen viele Daten zum Training zur Verfügung, so wird das Netz größere Schwierigkeiten haben die Fehlerfunktion zu minimieren, dafür wird der Unterschied zwischen neuen und bereits bekannten Daten nicht so groß ausfallen. In vielen aktuellen Anwendungen stellt die Beschaffung großer Datenmengen eine erhebliche Schwierigkeit dar. Überanpassung spielt daher in diesen Fällen eine große Rolle. Für die handschriftliche Ziffernerkennung stehen in Form des MNIST Datensatzes bis zu 60.000 Trainingsdaten zur Verfügung. Die Überanpassung fällt daher eher schwach aus, kann jedoch trotzdem beobachtet werden. Deshalb betrachten wir im Folgenden einige fortgeschrittenere Methoden zur Abschwächung dieses Problems.

Vorzeitiger Abbruch

Eine einfache Methode zur Vermeidung einer Überanpassung ist der vorzeitige Abbruch des Optimierungsprozesses. Allerdings stellt die Wahl des richtigen Zeitpunkts für das Been-

den des Trainingsprozesses eine gewisse Herausforderung dar. Man kann zwar problemlos während des Trainings die Qualität des neuronalen Netzes prüfen, jedoch sind Plateaus oder sogar Verschlechterungen ganz normal und bedeuten nicht notwendigerweise, dass eine Überanpassung stattfindet und dass sich das KNN nicht zu einem späteren Zeitpunkt wieder verbessert.

Weight-Decay

Eine geschicktere Methode, um Überanpassung zu vermeiden, ist die Regularisierung. Es gibt verschiedene Methoden zur Regularisierung, die am häufigsten verwendete ist jedoch die Weight-Decay-Methode. Dabei wird die bisherige Fehlerfunktion C_0 durch einen zusätzlichen Regularisierungsterm erweitert:

$$C = C_0 + \frac{\lambda}{2n} \sum_w w^2 \tag{3.16}$$

Hierbei ist n die Gesamtzahl der Trainingssätze, w die Gewichte des Netzes und λ der sogenannte Regularisierungsparameter. Man erkennt anhand von Gleichung (3.16) deutlich, dass große Gewichte nur dann auftreten können, wenn sie zu einer deutlichen Verringerung von C_0 führen. Es werden also überwiegend kleine Gewichte ermittelt werden. Dies stellt natürlich eine Einschränkung dar und verschlechtert die Minimierung der ursprünglichen Feherfunktion C_0. Allerdings führt die Verwendung kleiner Gewichte dazu, dass kleine Änderungen der Eingabe nur einen geringen Unterschied für die Ausgabe des Netzes verursachen. Ein solches Netz ist somit robuster gegenüber verrauschten Daten oder, wie in unserem Beispiel, Unterschieden in den verschiedenen Handschriften.

Dropout

Ein anderer Ansatz zur Regularisierung ist der Dropout. Hierbei trainiert man in jedem Schritt nur mit einem Teil der Neuronen. Man passt also auch immer nur einen Teil der Parameter an. Nach jedem Mini-Batch werden dann neue Neuronen ausgewählt und mit diesen trainiert. Dabei vernachlässigt man sinnvollerweise jedoch ausschließlich Neuronen aus den verdeckten Schichten. Verwendet man nun nach dem Trainingsprozess das gesamte Netz, so besteht dieses aus deutlich mehr Neuronen, als das im Training der Fall war. Es ist also nicht zu erwarten, dass die so erlernten Parameter besonders gute Ergebnisse liefern. Um diesen Effekt auszugleichen, werden die Parameter entsprechend verringert. Verwendet man während des Lernprozesses beispielsweise nur die Hälfte der verdeckten Neuronen, so werden die erlernten Gewichte am Ende halbiert.

Dropout hat eine recht ähnliche Funktionsweise wie die Weight-Decay-Methode. Indem man im Trainingsprozess immer wieder das neuronale Netz verändert, ist das Netz am Ende nicht so anfällig für kleine Änderungen in der Eingabe. Die Eingaben der einzelnen Neuronen haben sich ja schließlich während dem Lernprozess ständig verändert. Somit ist es dem Netz nicht mehr möglich, sich zu sehr auf die Trainingsdaten zu spezialisieren und man erreicht eine bessere Verallgemeinerung [5, 18].

3.3.2 Wahl der Startwerte

Die Wahl günstiger Startwerte hat in vielen Fällen erhebliche Auswirkungen auf das Ergebnis eines Optimierungsprozesses. Besonders für Probleme mit einer Vielzahl von Minima, was auf die Fehlerfunktion, wie in Abschnitt 3.2.1 erwähnt, zutrifft. Die richtige Wahl der Startwerte kann also erheblichen Einfluss auf den Erfolg des Lernprozesses eines neuronalen Netzes haben. Wie zuvor erläutert führt eine Regularisierung zu kleineren Gewichten. Es erscheint also sinnvoll, für die Startwerte direkt Werte nahe 0 zu wählen. Darüber hinaus verläuft die Sigmoidfunktion an dieser Stelle annähernd linear, was das System und damit auch den Trainingsprozess deutlich vereinfacht. Während des Optimierungsprozess werden dann, falls nötig, die Gewichte erhöht und damit die Nichtlinearität verstärkt.

Die Wahl der Anfangswerte hat aber nicht nur Auswirkungen auf den Erfolg des Lernprozesses, sondern auch auf die Geschwindigkeit, mit der das KNN lernt. Werden die Startwerte z.B. durch eine Gaußverteilung mit Erwartungswert 0 und Varianz 1 erzeugt, so erfüllt man zwar die vorherige Forderung, Startwerte nahe 0 zu wählen, jedoch führt das dazu, dass die die Ausgaben der Neuronen häufig in der Nähe von 0 oder 1 liegen. In diesem Bereich ist die Sigmoidfunktion allerdings sehr flach, wodurch kleine Änderungen der Gewichte nur sehr geringe Auswirkungen auf die Ausgabe des Netzes haben. Es lernt also sehr langsam. Eine bessere Wahl ist stattdessen eine zentrierte Gaußfunktion mit Varianz $\frac{1}{\sqrt{n_{in}}}$, wobei n_{in} die Zahl der Eingabewerte des entsprechenden Neurons ist [5, 18].

Man erkennt also, dass eine Verringerung der Abhängigkeit von den Startwerten eine erhebliche Erleichterung im Umgang mit neuronalen Netzen bedeuten kann. Dies motiviert die Verwendung fortgeschrittenerer Optimierungsverfahren, wie die Flussgleichungsmethode. Jedoch ist dabei besonders auf eine Überanpassung zu achten.

4 Numerische Optimierung der Flussgleichungsmethode

Wie in den Abschnitten 2.4 und 3.2.4 diskutiert wurde, besteht der größte numerische Aufwand in der Invertierung der Hesse-Matrix bzw., für die Anwendung auf neuronale Netze, bereits in der Berechnung dieser. Wir wollen deshalb in diesem Kapitel einige Möglichkeiten betrachten, die Flussgleichungsmethode numerisch zu optimieren.

4.1 Vernachlässigung der Hesse-Matrix

Vernachlässigt man die Hesse-Matrix komplett in Gleichung (2.10), erhält man eine Flussgleichung der Form

$$\frac{\partial f_n}{\partial t} \approx -\frac{\partial E}{\partial f_m}. \tag{4.1}$$

Löst man dieses Anfangswertproblem mit dem Euler Verfahren, ist dies äquivalent zum sogenannten Gradientenverfahren oder auch Verfahren des steilsten Abstiegs (siehe z.B. [19]). Mithilfe des Corrector-Schritts lässt sich die Konvergenz des Verfahrens, wie in Abschnitt 2.5 diskutiert, noch verbessern. Man erhält dann die folgende Iterationsvorschrift:

$$\tilde{f}_n(t_{i+1}) = f_n(t_i) - \delta t \frac{\partial E}{\partial f_n}\bigg|_{t=t_i} \tag{4.2}$$

$$f_n(t_{i+1}) = f_n(t_i) - \frac{\delta t}{2}\left(\frac{\partial E}{\partial f_n}\bigg|_{t=t_i} + \frac{\partial E}{\partial \tilde{f}_n}\bigg|_{t=t_{i+1}}\right) \tag{4.3}$$

Der Nachteil eines solchen Gradientenverfahrens ist, dass die Änderung von f_n selber nur noch von f_n abhängt. Dies führt dazu, dass solche Verfahren meist nur sehr langsam konvergieren, da man sich nicht auf direktem Weg dem Minimum nähert. Dadurch ergeben sich mit dieser Methode schlechtere Resultate für den Fall, dass man am exakten Wert des Minimums interessiert ist, wie wir bei der Suche nach Grundzuständen in Spinglas-Systemen in Abschnitt 5.2 sehen werden. Im Falle neuronaler Netze ist man jedoch, wie in Abschnitt

3.3.1 diskutiert, oftmals gar nicht daran interessiert, die exakte Position des Minimums zu finden, sondern möchte die Parameter nur bis zu einem gewissen Punkt verbessern. Für solche Anwendungen eignet sich dieses Verfahren somit sehr gut.

4.2 Optimierung mittels Neumannscher Reihe

Wir wollen nun eine weitere Möglichkeit betrachten, die Flussgleichungsmethode zu optimieren. Dabei soll nicht die Hesse-Matrix an sich vernachlässigt, sondern ihr Inverses mithilfe einer Neumannschen Reihe approximiert werden. Damit lässt sich eine inverse Matrix der Form $(I+T)^{-1}$ ausdrücken durch [20]

$$(I+T)^{-1} = \sum_{k=0}^{\infty}(-T)^k \approx I - T. \tag{4.4}$$

Wobei $||T|| < 1$ gelten muss, was im Folgenden durch die entsprechende Wahl der Schrittweite zu gewährleisten ist. I stellt die Einheitsmatrix dar. Zur übersichtlicheren Notation bezeichnen wir die Hesse Matrix aus Gleichung (2.10) zum Zeitschritt $l-1$ des Heun-Verfahrens als

$$A_{l-1} = \frac{1-t_{l-1}}{f_n(t_{l-1})}\delta_{m,n} + t_{l-1}\frac{\partial^2 E}{\partial f_m \partial f_n}\bigg|_{t_{l-1}}. \tag{4.5}$$

Die Idee ist es nun, die Hesse-Matrix zum Zeitschritt l durch eine kleine Variation der Matrix zum Zeitschritt $l-1$ auszudrücken. Sodass wir die Form

$$A_l = A_{l-1} + \delta t B_{l-1} \tag{4.6}$$

erhalten, wobei auch B_{l-1} aus dem vorherigen Schritt bekannt sein soll und δt die Schrittweite darstellt. Dies ermöglicht es uns, das Inverse der Hesse-Matrix wie folgt zu schreiben:

$$A_l^{-1} = (A_{l-1} + \delta t B_{l-1})^{-1} = (I + \delta t A_{l-1}^{-1} B_{l-1})^{-1} A_{l-1}^{-1} \tag{4.7}$$

Mithilfe von Gleichung (4.4) erhält man daraus

$$A_l^{-1} \approx (I - \delta t A_{l-1}^{-1} B_{l-1}) A_{l-1}^{-1}. \tag{4.8}$$

Damit haben wir die Matrixinvertierung als Produkt von Matrizen approximiert, die aus dem vorherigen Schritt bekannt sind. Man muss damit nur einmal für den ersten Schritt eine explizite Inversion durchführen. Diese ist jedoch trivial, da A_0 für $t_0 = 0$ diagonal ist, wie man in Gleichung (4.5) erkennen kann. Da das Multiplizieren von Matrizen erheblich schneller ist als das Invertieren, erwarten wir hiervon eine deutliche Beschleunigung des Verfahrens.

Betrachten wir nun Gleichung (4.5) um A_l auf die Form von Gleichung (4.6) zu bringen. Dazu stellen wir A_l durch in Abhängigkeit von t_{l-1} wie folgt dar.

$$A_l = \frac{1 - t_{l-1} - \delta t}{f_n(t_{l-1} + \delta t)} \delta_{m,n} + (t_{l-1} + \delta t) \frac{\partial^2 E}{\partial f_m \partial f_n}\bigg|_{t_{l-1}+\delta t} \tag{4.9}$$

Dies lässt sich umformen zu

$$A_l = A_{l-1} + \delta t \left(-\frac{\delta_{m,n}}{f_n(t_{l-1})} - \frac{(1 - t_{l-1}) \dot{f}_n(t_{l-1})}{f_n(t_{l-1})^2} + \left(1 + t_{l-1} \frac{\partial}{\partial t}\right) \frac{\partial^2 E}{\partial f_n \partial f_m}\bigg|_{t_{l-1}} \right). \tag{4.10}$$

Aus dem Vergleich mit Gleichung (4.6) ergibt sich damit

$$B_{l-1} = -\frac{\delta_{m,n}}{f_n(t_{l-1})} - \frac{(1 - t_{l-1}) \dot{f}_n(t_{l-1})}{f_n(t_{l-1})^2} + \left(1 + t_{l-1} \frac{\partial}{\partial t}\right) \frac{\partial^2 E}{\partial f_n \partial f_m}\bigg|_{t_{l-1}}. \tag{4.11}$$

Alle in Gleichung (4.11) vorkommenden Terme sind prinzipiell aus dem vorherigen Schritt bekannt. Je nach Form von E kann jedoch die Zeitableitung von $\partial^2 E/(\partial f_n \partial f_m)$ problematisch sein. Gerade für die Anwendung auf neuronale Netze ist diese Form deshalb besonders ungünstig, da die Berechnung der dritten Ableitungen der Fehlerfunktion, analog zur Diskussion in Abschnitt 3.2.4, besonders aufwendig ist. Diese Form der Entwicklung ist deshalb in diesem Fall nicht praktikabel. Ist die Schrittweite gering genug und die zeitliche Änderung von $f_n(t)$ und $\partial^2 E/(\partial f_n \partial f_m)$ nur sehr langsam, so könnte man die entsprechenden Terme in Gleichung (4.11) unter Umständen vernachlässigen und erhält damit

$$B_{l-1} \approx -\frac{\delta_{m,n}}{f_n(t_{l-1})} + \frac{\partial^2 E}{\partial f_n \partial f_m}\bigg|_{t_{l-1}}. \tag{4.12}$$

Wie wir im späteren Verlauf der Arbeit sehen werden, ist es jedoch bei der Anwendung auf KNNs sinnvoll, die Schrittweite sehr groß zu wählen, womit nicht zu erwarten ist, dass diese Näherung gerechtfertigt ist. Darüber hinaus tritt auch in Gleichung (4.12) die zweite Ableitung der Fehlerfunktion auf, was, wie in Abschnitt 3.2.4 diskutiert, ungünstig ist. Auch diese Form der Entwicklung ist somit ungeeignet für die Anwendung auf neuronale Netze. Für Problemstellungen mit einer günstigeren Form des Funktionals E könnte diese Entwicklung jedoch prinzipiell eine erhebliche Beschleunigung der Flussgleichungsmethode bedeuten.

5 Spingläser

Gläser bzw. amorphe Festkörper unterscheiden sich von kristallinen Festkörpern durch ihre ungeordnete atomare Struktur in der festen Phase. Analog dazu unterscheiden sich Spingläser von Ferromagneten durch ihre ungeordnete Spinstruktur bei tiefen Temperaturen. Diese Unordnung wird durch zufällige, miteinander konkurrierende Wechselwirkungen mit verschiedenen Vorzeichen in diesen Systemen verursacht und es ergeben sich daraus ungeordnete und frustrierte magnetische Systeme.

Solche ungeordneten Systeme weisen oft eine große Komplexität auf, wodurch ihre theoretische Behandlung eine erhebliche Herausforderung darstellt. Gleichzeitig führt ihre Komplexität zu einem großen Anwendungsbereich der, anhand ihrer, entwickelten Methoden. Spingläser stellen ein ausgezeichnetes Modellsystem für komplexe Systeme dar, da die Komplexität hierbei bereits aus einem sehr simplen Modell hervorgeht. Dies hat dazu geführt, dass sie, trotz eingeschränkten technischen Nutzens, seit ihrer experimentellen Beobachtung zu Beginn der 1960er Jahre intensiv untersucht werden [1].

Es gibt verschiedene theoretische Modelle zur Beschreibung von Spingläsern. Wir werden in dieser Arbeit mithilfe der Flussgleichungsmethode Grundzustände im Edwards-Anderson Modell suchen und deshalb hier nur auf dieses näher eingehen.

5.1 Edwards-Anderson Modell

Im Edwards-Anderson Modell berücksichtigt man ausschließlich die Wechselwirkung zwischen benachbarten Gitterpunkten auf einem d-dimensionalen Gitter. Der Hamiltonoperator des Systems sieht dann wie folgt aus [21].

$$\mathcal{H} = \sum_{\langle i,j \rangle} J_{ij} \sigma_i^z \sigma_j^z \tag{5.1}$$

Wobei σ_i^z die Paulimatrix am Gitterplatz i und J_{ij} die Kopplungskonstante zwischen den benachbarten Gitterplätzen i und j darstellt. Die J_{ij} ergeben sich zufällig aus einer zentrierten Gaußschen Verteilung mit Varianz 1. Man erhält somit sowohl negative als auch

positive Werte für J_{ij}, was zu konkurrierenden ferromagnetischen und antiferromagnetischen Wechselwirkungen führt.

Wir werden im Folgenden ein zweidimensionales System in einem kombinierten longitudinalen und transversalen Magnetfeld betrachten. Der Ausdruck aus Gleichung (5.1) erweitert sich damit zu

$$\mathcal{H} = \sum_{\langle i,j \rangle} J_{ij} \sigma_i^z \sigma_j^z - h_x \sum_i \sigma_i^x - h_z \sum_i \sigma_i^z. \tag{5.2}$$

h_x bzw. h_z stellen dabei die x- bzw. z-Komponente des externen Magnetfelds dar.

5.2 Suche nach Grundzuständen mithilfe der Flussgleichungsmethode

Die Suche nach Grundzuständen von Spinglassystemen gehört zur Klasse der kombinatorischen Optimierungsprobleme, die in verschiedenen Bereichen, wie z.B. der Informatik oder den Ingenieurswissenschaften eine wichtige Rolle spielen. Wie viele Beispiele aus diesem Bereich ist auch dieses Problem NP-schwer und somit nicht effizient exakt lösbar. Deshalb benötigt man zu seiner Behandlung numerische Näherungsverfahren wie die Flussgleichungsmethode [22].

Um diese anwenden zu können, müssen wir zunächst Gleichung (5.2) in ein Optimierungsproblem mit positiven Parametern kleiner 1 überführen. Dazu verwenden wir den allgemeinsten Produktansatz

$$|\psi_0\rangle = \prod_i (\alpha_i |\uparrow_i\rangle + \beta_i |\downarrow_i\rangle) \tag{5.3}$$

für eine Lösung des Systems. Dabei sind $|\uparrow\rangle$ und $|\downarrow\rangle$ die Eigenzustände von σ^z. Im Grundzustand des Systems können die Parameter α_i und β_i reell gewählt werden, außerdem gilt die Normierungsbedinungung $\alpha^2 + \beta^2 = 1$. Somit lässt sich Gleichungen (5.2) auf ein Minimierungsproblem der Energie

$$E[f_n] = \sum_{\langle i,j \rangle} J_{ij}(2f_i - 1)(2f_j - 1) - 2h_x \sum_i \sqrt{f_i(1 - f_i)} - h_z \sum_i (2f_i - 1), \tag{5.4}$$

bezüglich der Wahrscheinlichkeiten $f_i = \beta_i^2 \in [0, 1]$, sich in einem Spin-Up Zustand zu befinden, zurückführen [6].

Wir werden auf dieses Minimierungsproblem sowohl die in Abschnitt 2.4 dargestellte Flussgleichungsmethode, als auch die im Abschnitt 4 hergeleiteten numerisch optimierten Ver-

sionen anwenden und die Qualität der Ergebnisse sowie die benötigten Laufzeiten miteinander vergleichen. Bei dem untersuchten System handelt es sich um ein Gitter mit 5×5 Gitterpunkten in einem externen Magnetfeld mit $h_x = 0.05$ und $h_z = 0.1$. Dabei wurden 100 zufällige Instanzen des Spinglassystems mit einer Schrittweite $dt = 0.001$ minimiert, wobei jeweils die Lösung mit der geringsten Energie von 5 verschiedenen Startzuständen gewählt wurde. Die Startzustände der f_i wurden gleichverteilt aus dem Intervall $[0.45, 0.55]$ gewählt.

Für die ursprüngliche Form der Flussgleichungsmethode erhielten wir eine mittlere Energie pro Gitterpunkt von $E/N = -1.114$ mit einer Standardabweichung von $\sqrt{E}/N = 0.139$ bezüglich der verschiedenen Realisierungen der J_{ij}. Im Falle der kompletten Vernachlässigung der Hesse-Matrix, wie im Abschnitt 4.1 erläutert, erhielten wir für $E/N = -1.105$ mit $\sqrt{E}/N = 0.130$. Es ist anhand dieser Werte deutlich zu erkennen, dass die beiden Methoden in dieser Anwendung vergleichbare Ergebnisse liefern, wenn auch die ursprüngliche Methode etwas bessere Ergebnisse liefert. Die in Abschnitt 4.2 durchgeführte Entwicklung der Flussgleichungsmethode weist in dieser Anwendung numerische Instabilitäten auf und führt somit zu keinem brauchbaren Ergebnis. Dies könnte daran liegen, dass die Bedingung $\|T\| < 1$ nicht erfüllt wird, da die Schrittgröße zu groß gewählt wurde. Eventuell ließe sich das durch eine erhebliche Verringerung dieser verhindern, aber auch eine Verringerung der Schrittweite um den Faktor 100 hat zu keiner Verbesserung geführt. Außerdem widerspricht dies dem ursprünglichen Ziel, das Verfahren zu beschleunigen. Die auftretenden Instabilitäten für diese spezielle Anwendung bedeutet jedoch nicht, dass das Verfahren nicht auf andere Problemstellungen erfolgreich angewandt werden kann.

Die Vernachlässigung der Hesse-Matrix führt zu einer Beschleunigung des Verfahrens um einen Faktor von 1.5. Im Falle eines Spinglassytems stellt jedoch die Berechnung dieser keinen großen Aufwand dar. Hier führt also hauptsächlich die umgangene Invertierung zu dem Geschwindigkeitsunterschied. Für Anwendungen, bei denen die Berechnung höherer Ableitungen sehr aufwendig ist, ist deshalb ein erheblich größerer Geschwindigkeitsvorteil zu erwarten. Wie in Abschnitt 3.2.4 diskutiert, stellen neuronale Netze einen solchen Fall dar. Wir haben anhand der Ergebnisse für Spinglassysteme gesehen, dass wir auch ohne Berücksichtigung der Hesse-Matrix durchaus vergleichbare Ergebnisse erhalten. Aus diesen Gründen werden wir im folgenden Abschnitt ausschließlich auf diese Methode zurückgreifen.

6 Handschriftliche Ziffernerkennung mithilfe neuronaler Netze

Nach der Untersuchung der Energielandschaften von Spingläsern im vorherigen Abschnitt, wollen wir nun die Flussgleichungsmethode auf den Trainingsprozess neuronaler Netze anwenden. Ein häufig betrachtetes Beispiel in der Entwicklung neuronaler Netze ist die Klassifizierung handgeschriebener Ziffern. Dieses Problem ist anspruchsvoll genug, um an ihm ernsthaft neue Methoden zu testen und gleichzeitig einfach genug, dass sich die Komplexität der Lösungen, sowie die benötigte Rechenleistung im Rahmen halten. Außerdem liegen in Form des MNIST Datensatzes ausreichend Trainings- und Testdaten vor.

6.1 Verwendete Trainings- und Testdaten

Der MNIST Datensatz enthält insgesamt 70.000 Graustufen Bilder mit einer Größe von 28×28 Pixel. Davon dienen 60.000 Bilder als Trainingsdaten und die übrigen 10.000 als Testdaten, diese beiden Datensätze enthalten Bilder von unterschiedlichen Personen, sodass die Qualität des neuronalen Netzes mithilfe von Bildern überprüft wird, die während des Trainingsprozesses nicht verwendet wurden. Dies ermöglicht es auch, das in Abschnitt 3.3.1 diskutierte Übertraining zu detektieren. Die 60.000 Trainingsdaten werden wir in zwei weitere Datensätze aufteilen. 50.000 der Bilder verwenden wir für den Trainingsprozess des neuronalen Netzes, die übrigen 10.000 Bilder dienen zur Validierung des KNNs, um damit die Werte für die Hyperparameter des Netzes, wie Schrittgröße oder Neuronenzahl anzupassen.

6.2 Verschiedene Architekuren neuronaler Netze

Neben dem Trainingsprozess spielt auch die Architektur eines neuronalen Netzes eine entscheidende Rolle für die Qualität des Netzes. Die Architektur hat wiederum Auswirkungen

auf den Erfolg sowie den Aufwand des Trainingsprozesses. Wir werden deshalb verschiedene KNNs mit unterschiedlich komplizierten Architekturen mit der Flussgleichungsmethode trainieren und die Ergebnisse vergleichen.

Die Ein- und Ausgabeschichten sind dabei bei allen Netzen identisch. Da die Eingabeschicht nur die Eingabedaten repräsentiert, ist ihre Form durch das Format der Daten festgelegt. In unserem Fall handelt es sich um Bilder mit einer Größe von $28 \times 28 = 784$ Pixeln. Die Eingabeschicht besteht also aus 784 Neuronen, deren Ausgabewerte den entsprechenden Graustufenwerten des jeweiligen Pixels entsprechen. Die Ausgabeschicht besteht aus 10 Neuronen, wobei jedes Neuron einer Ziffer entspricht. Die Ausgabe des Netzes stellt das Neuron mit der höchsten Aktivierung dar.

6.2.1 Vollständig verbundene Netze

Bei vollständig verbundenen Netzen ist jedes Neuron aus einer Schicht mit allen Neuronen in der nächsten Schicht verbunden. Dies hat den Effekt, dass jede zusätzliche Schicht, die Zahl der Parameter des Netzes deutlich erhöht und damit den Trainingsprozess aufwendiger gestaltet. Es gilt also abzuwägen, ob die Verbesserung, die durch eine zusätzliche Schicht erreicht werden kann, groß genug ist, um das aufwändigere Training zu rechtfertigen.

Die einfachste vollständig verbundene Struktur verzichtet auf eine verdeckte Schicht. Ein- und Ausgabeschicht werden also einfach direkt miteinander verbunden. Von einem solchen Netz ist jedoch nicht zu erwarten, dass es komplexe Aufgaben besonders gut lösen kann und wir werden später sehen, dass es auch für das MNIST Problem deutlich schlechtere Ergebnisse liefert, als Netze mit tieferen Schichten. Das liegt daran, dass tiefere Architekturen die Möglichkeit bieten, eine Aufgabe in kleinere, einfachere Teilaufgaben aufzuteilen. Die Ergebnisse dieser Teilprobleme werden dann zur Lösung der eigentlichen Aufgabe kombiniert. Am Beispiel der Ziffernerkennung wären solche Teilaufgaben beispielsweise das Erkennen von einfachen Formen, wie geraden Linien, Kreisen oder Bögen, aus denen die unterschiedlichen Ziffern dann zusammengesetzt werden.

Wir führen deshalb verborgene Schichten ein. Die Zahl der Neuronen dieser Schichten spielt ebenfalls eine wichtige Rolle für die Qualität des Netzes. Wir werden verborgene Schichten mit 100 Neuronen pro Schicht verwenden. Wir werden sehen, dass das Einführen einer verborgenen Schicht eine erhebliche Steigerung verursachen, die Verwendung einer zweiten verborgenen Schicht, jedoch keinen großen Unterschied machen wird. Das liegt daran, dass tiefe Netze schwieriger zu trainieren sind als flache Netze. Speziell Netze mit sigmoidalen Neuronen sind davon betroffen. Einer der Hauptgründe dafür ist das Vanishing Gradient

Problem, auf welches wir an dieser Stelle nicht genauer eingehen werden.[1] Wir werden es stattdessen durch die Verwendung der im folgenden Abschnitt vorgestellten Convolutional Nets umgehen [5].

6.2.2 Convolutional Nets

Wie im vorherigen Abschnitt angesprochen, erhofft man sich von tieferen Architekturen die Möglichkeit, Probleme in Subprobleme aufzuteilen und aus deren Lösung auf die des Hauptproblems zu schließen. Diese Idee spielt im Aufbau von Convolutional Nets (ConvNets) eine zentrale Rolle. Im Gegensatz zu den vollständig verbundenen Netzen sind diese in der Lage die räumlichen Strukturen des Bildes zu berücksichtigen. Außerdem ermöglicht ihre Architektur ein schnelles Training trotz einer Vielzahl an Schichten. Anstatt alle Eingabewerte mit allen Neuronen in der nächsten Schicht zu verbinden, tastet man hierbei das Bild Stück für Stück ab. Dieses Vorgehen ist biologisch in Form des rezeptiven Feldes motiviert. Man verbindet dabei nur einen bestimmten Bereich des Bildes, das sogenannte rezeptive Feld, mit dem ersten Neuron der nächsten Schicht. Das zweite Neuron dieser Schicht wird dann mit einem anderen, gleich großen Bereich des Bildes verbunden. Beginnt man beispielsweise, wie in Abbildung 6.1 dargestellt, im linken Eck des Bildes, so könnte man den betrachteten Bereich jeweils um ein Pixel weiter nach rechts verschieben, bis man am Rand des Bildes angelangt ist. Man fährt dann fort, indem man einen Schritt nach unten macht und dann wieder vom linken bis zum rechten Rand läuft. Dies führt man so weiter, bis man schließlich das rechte untere Eck des Bildes erreicht. Im Falle eines rezeptiven Feldes von 5×5 Pixeln hat somit jedes Neuron 5 Gewichte und einen Bias-Parameter. In diesem Fall erhält man damit, mit einer Schrittgröße von einem Pixel, für die 28×28 Pixel Bilder eine erste verdeckte Schicht aus 24×24 Neuronen.

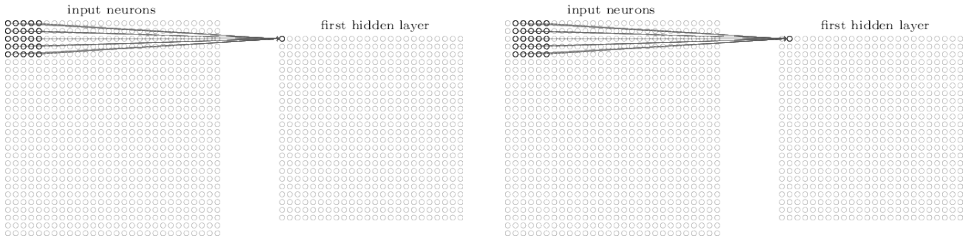

Abbildung 6.1: Abtasten der Eingabeschicht nach dem biologischen Vorbild des rezeptiven Bildes. Das rezeptive Feld hat dabei eine Größe von 5×5 Pixeln. Es wird eine Schrittgröße von einem Pixel verwendet, wie beim Vergleich des rezeptiven Feldes des ersten Neurons (links) mit dem des zweiten Neurons (rechts) zu erkennen ist. Abbildungen aus [5].

[1] Detailliertere Betrachtungen sind z.B. in [5] oder [23] zu finden.

Neben der räumlichen Aufteilung des Bildes in rezeptive Felder gibt es noch einen weiteren konzeptionellen Unterschied zu den vollständig verbundenen Netzen. Man verwendet geteilte Gewichte w, was bedeutet, dass jedes Neuron in der verdeckten Schicht die exakt gleichen Parameter verwendet. Damit ergibt sich für die Aktivierung $a^1_{j,k}$ des j,k-ten Neurons in der ersten verdeckten Schicht

$$a^1_{j,k} = \sigma\left(\sum_{m=0}^{4}\sum_{n=0}^{4} w_{m,n} a^0_{j+m,k+n}\right) = \sigma\left([w * a^0]_{j,k}\right), \tag{6.1}$$

mit einem rezeptiven Feld der Größe 5×5. Wobei $[w * a^0]_{j,k} = \sum_{m=0}^{4}\sum_{n=0}^{4} w_{m,n} a^0_{j+m,k+n}$ die Form einer diskreten Faltung hat, was den Ursprung des Namens Convolutional Net darstellt [15]. Die Verwendung geteilter Gewichte hat natürlich den Effekt, dass die Zahl der benötigten Parameter pro verdeckter Schicht in ConvNets deutlich geringer ist, es ist jedoch nicht sofort ersichtlich, inwiefern das zur erfolgreichen Klassifizierung der Bilder beiträgt. Heuristisch lässt sich die Funktionsweise dieses Vorgehens wie folgt beschreiben. Dadurch dass jedes Neuron in der Schicht die gleichen Parameter besitzt, erlernt diese das Erkennen von gewissen Strukturen oder Merkmalen im Bild und kann diese lokalisieren. Beispiele dafür wären Kanten und Ecken, sowie ihre Position im Bild [5, 24]. Aus diesem Grund werden solche Schichten auch als Feature Maps bezeichnet. Eine Schicht ist dann jedoch nur in der Lage, ein bestimmtes Merkmal zu erkennen. Um die Ziffern richtig zu charakterisieren, sollte das Netz jedoch in der Lage sein, mehrere Merkmale sowie ihre räumliche Lage miteinander zu kombinieren. Dies motiviert die Verwendung mehrerer Feature Maps parallel in der gleichen Schicht, dem sogenannten Convolutional Layer, was in Abbildung 6.2 beispielhaft mit 3 Feature Maps dargestellt ist. Wir werden später ConvNets mit 20 Feature Maps in einer Schicht verwenden.

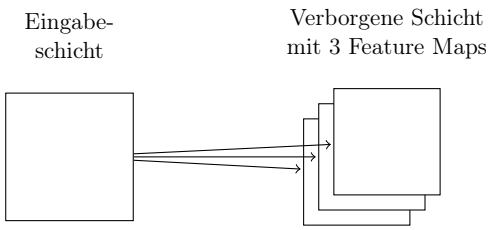

Abbildung 6.2: Schematische Darstellung einer verborgenen Schicht, bestehend aus 3 Feature Maps. Jede Feature Map ist mit der Eingabeschicht verbunden, es bestehen jedoch keine Verbindungen zwischen den Feature Maps der gleichen Schicht.

Der dritte konzeptionelle Unterschied zu vollständig verbundenen Netzen ist das Pooling, was meist direkt nach Feature Maps angewandt wird. Dies geschieht in Form der sogenannten Pooling Layer, welche, wie in Abbildung 6.3 dargestellt, jeweils genau mit einer Feature Map verbunden sind und welche die Information, die sie von diesen erhalten, vereinfachen.

Dies geschieht, indem ein Bereich von mehreren Neuronen der Feature Map auf ein Neuron des Pooling Layers abgebildet wird. Dies kann auf unterschiedliche Weisen erfolgen, wir werden in dieser Arbeit das sogenannte Max-Pooling verwenden und deshalb nur auf dieses näher eingehen. Beim Max-Pooling wird aus einem Bereich von 2×2 Neuronen ausschließlich das Neuron mit der höchsten Aktivierung berücksichtigt. Dieses stellt dann das entsprechende Neuron des Pooling Layers dar. Auf das Pooling Layer kann dann entweder direkt die Ausgabeschicht, eine weitere Kombination aus Feature Maps und Pooling Layer oder einfache vollverbundene Schichten folgen [5].

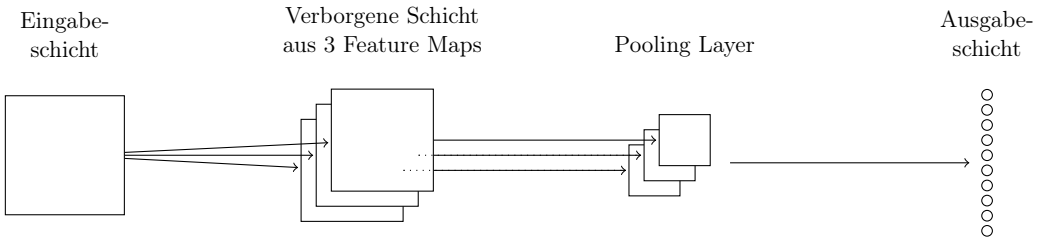

Abbildung 6.3: Schematische Darstellung der Gesamarchitektur eines einfachen ConvNets. Verwendung von Feature Maps in der ersten verborgenen Schicht, gefolgt von den zugehörigen Pooling Layern, welche vollständig mit der Ausgabeschicht verbunden sind.

Wir werden in dieser Arbeit sechs verschiedene Architekturen testen und die Ergebnisse miteinander vergleichen. Die Ein- und Ausgabeschicht sind für alle Netze identisch, die Eingabeschicht wird durch das Format der Trainingsdaten vorgegeben, als Ausgabeschicht wird eine Softmax Schicht (S) verwendet. Das einfachste Netz besteht nur aus Ein- und Ausgabeschicht. Darüber hinaus werden wir Netze mit einer bzw. zwei vollverbundenen Schichten (V) verwenden. Diese enthalten jeweils 100 Neuronen. Außerdem verwenden wir auch Convolutional Nets. In diesen bestehen die Convolutional Layer (C) aus 20 Feature Maps mit einem rezeptiven Feld von 5×5 Pixeln, die Pooling Layer (P) verwenden Max-Pooling und eine Schrittgröße von einem Pixel. Das einfachste der ConvNets besteht nur aus einem Convolutional Layer, einem Pooling Layer und der Ausgabschicht. Dieses Netz werden wir zusätzlich noch durch eine vollständig verbundene Schicht erweitern und so ein weiteres Netz erhalten. Das komplexeste der verwendeten Netze verwendet zweimal hintereinander die Kombination aus Convolutional Layer und Pooling Layer, worauf dann abschließend eine vollständig verbundene Schicht und die Ausgabeschicht folgen.

6.3 Training verschiedener neuronaler Netze mithilfe der Flussgleichungsmethode

Für die Implementierung wurde der in [5] vorgestellten Code als Grundlage genutzt. Dieser verwendet die Theano Bibliothek [25] für die Implementierung des neuronalen Netzes. Das in [5] verwendete Gradientenverfahren wurde durch die Flussgleichungsmethode ersetzt. Die im vorherigen Abschnitt dargestellten KNNs wurden dann sowohl mit der Flussgleichungsmethode als auch dem stochastischen Gradientenabstieg trainiert, die Ergebnisse sind in Tabelle 6.1 dargestellt, wobei der Wert mit dem größten Anteil erkannter Validierungsdaten aus allen Epochs verwendet wurde. Dabei setzen sich die Bezeichnungen der verschiedenen Netze aus den im vorherigen Abschnitt verwendeten Abkürzungen ihrer Schichten zusammen. VS steht also beispielsweise für ein Netz mit einer vollständig verbundenen Schicht und einer Softmax Ausgabeschicht. Beim Training wurden eine Dropout, mit einer Dropoutwahrscheinlichkeit von 50% und Weight-Decay, mit einem Regularisierungsparameter von $\lambda = 0.1$ verwendet. Die Mini-Batches hatten eine Größe von 10 und es wurden 30 Trainingsepochs durchgeführt. Im Falle der Flussgleichungsmethode wurde mit einer Schrittgröße von 0.2 bis zu einem Maximalwert von 0.4 trainiert, für das Gradientenverfahren wurde entsprechend eine Lernrate von 0.2 verwendet. Die wesentlich größere Wahl der Schrittweiten im Vergleich zu Abschnitt 5.2 und die Tatsache, dass das Intervall $[0, 1]$ für den Parameter t nicht ganz ausgenutzt wurde, lässt sich mit der Diskussion in Abschnitt 3.3.1 motivieren. Wie dort erläutert, ist es nicht das Ziel des Trainingsprozesses das Minimum der Fehlerfunktion zu erreichen. Die zuvor genannten Parameter wurden, mithilfe der, im Abschnitt 6.1 erläuterten, Validierungsdaten ermittelt.

Architektur	S	VS	VVS	CPS	CPVS	CPCPVS
Validierungsdaten FGM	91.19%	96.87%	95.97%	98.72%	98.90%	99.09%
Validierungsdaten SGD	92.21%	96.51%	96.24%	98.72%	98.81%	99.07%
Testdaten FGM	90.13%	96.39%	96.07%	98.53%	98.83%	99.15%
Testdaten SGD	91.44%	96.08%	95.75%	98.58%	98.71%	99.02%

Tabelle 6.1: Ergebnisse der unterschiedlichen Netze und unterschiedlichen Trainingsmethoden im Vergleich. Die Netze wurden mit der Flussgleichungsmethode (FGM), unter Vernachlässigung der Hesse Matrix (siehe Abschnitt 4.1) sowie dem stochastischen Gradientenabstieg (SGD) trainiert. Die Resultate sind vergleichbar, wenn auch die Flussgleichungsmethode, vor allem für tiefe Netze, tendenziell etwas bessere Ergebnisse liefert.

Da sich die beiden Verfahren, wie in Abschnitt 4.1 angemerkt, nur durch den Corrector Schritt unterscheiden, ist es zu erwarten, dass die Ergebnisse vergleichbar sind. Dies bestätigen die in Tabelle 6.1 dargestellten Resultate. Allerdings liefert die Flussgleichungsmethode, besonders für tiefe Netze, tendenziell etwas bessere Werte. Anhand von Tabelle 6.1 lässt sich auch der große Einfluss der Architektur des Netzes auf die Ergebnisse erkennen, der

den Unterschied zwischen den verwendeten Trainingsmethoden deutlich überwiegt. Es lässt sich außerdem erkennen, dass das Training des Netzes mit zwei vollständig verbundenen Schichten eine besondere Schwierigkeit aufweist und damit sogar schlechtere Ergebnisse erzielt wurden als mit einer einzelnen vollständig verbundenen Schicht, was die in Abschnitt 6.2.1 erwähnten Schwierigkeiten beim Training tiefer Netze widerspiegelt. Convolutional Nets weisen diese Probleme hingegen nicht im selben Ausmaß auf.

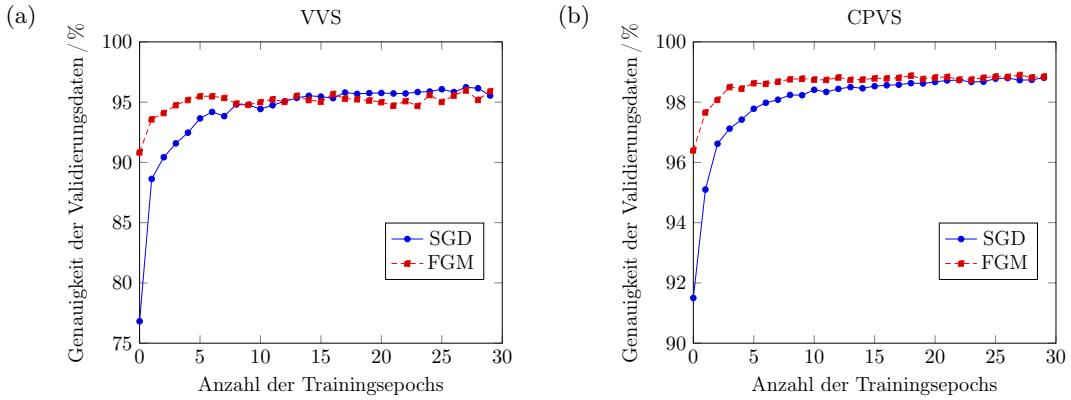

Abbildung 6.4: Prozentualer Anteil der korrekt klassifizierten Validierungsdaten für den Trainingsprozess mit der Flussgleichungsemthode (rot) und dem stochastischen Gradientenabstieg (blau) im Vergleich. Hier beispielhaft für zwei Architekturen dargestellt. (a) zeigt die Ergebnisse für ein Netz mit zwei vollständig verbundenen Schichten. Für (b) wurde ein Convolutional Net mit einer zusätzlichen vollständig verbundenen Schicht verwendet. In beiden Fällen konvergiert die Flussgleichungsmethode deutlich schneller. Beide Methoden konvergieren jedoch gegen vergleichbare Werte.

Durch den Corrector Schritt, ist es im Falle der Flussgleichungsmethode zweimal pro Zeitschritt nötig, den Gradienten zu berechnen, was zu einer merklichen Verlangsamung im Vergleich zum stochastische Gradientenabstieg führt. Allerdings konvergiert der Trainingsprozess für die Flussgleichungsmethode deutlich schneller, wie in Abbildung 6.4 beispielhaft für zwei Architekturen dargestellt ist. Dieses Verhalten war bei allen verwendeten Architekturen zu beobachten, sodass sich die aufwendigere Berechnung wieder teilweise durch eine geringere Zahl an benötigten Trainingsepochs kompensieren lässt.

7 Zusammenfassung und Ausblick

Wir haben in dieser Arbeit das in [6] vorgestellte numerische Optimierungsverfahren erläutert und mögliche numerische Optimierungen diskutiert. Dabei haben wir zum einen die dabei verwendete Hesse-Matrix komplett vernachlässigt, zum anderen eine Reihenentwicklung für ihre Invertierung genutzt. Diese modifizierten Verfahren haben wir anhand der Suche nach Grundzuständen in Spingläsern bezüglich Genauigkeit und Geschwindigkeit verglichen, wobei die entwickelte Variante numerische Instabilitäten aufwies, was sie für diese Anwendung untauglich macht. Abschließend haben wir das Verfahren, unter Vernachlässigung der Hesse-Matrix, auf den Lernprozess künstlicher neuronaler Netze angewandt und die Ergebnisse mit dem üblichen Gradientenverfahren verglichen. Dabei ist vor allem die schnellere Konvergenz der Flussgleichungsmethode aufgefallen. Die Resultate waren vergleichbar, wenn auch die Flussgleichungsmethode, gerade für tiefe Convolutional Nets, etwas bessere Ergebnisse geliefert hat.

Im Falle vieler Anwendungen neuronaler Netze liefern einfache Optimierungsverfahren wie der stochastische Gradientenabstieg zufriedenstellende Ergebnisse. Jedoch gibt es auch Situationen, in denen diese Verfahren an ihre Grenzen kommen. Dies kann an der Architektur des Netzes liegen, wie wir anhand der tiefen, vollständig verbundenen Netze gesehen haben. Aber auch eine geringe Anzahl an vorhandenen Trainingsdaten, sowie die Problemstellung selber, auf welche das neuronale Netz angewandt wird, können eine Herausforderung im Training eines KNNs darstellen. Es wäre nun interessant, die Flussgleichungsmethode auch in solchen Situationen anzuwenden. Wobei die Berücksichtigung der Hesse-Matrix in diesen Fällen wieder in Erwägung zu ziehen ist, da deren schnelle Konvergenz und besseren Ergebnisse von entscheidendem Vorteil sein und damit den erhöhten Rechenaufwand rechtfertigen könnten.

Sowohl das Feld der künstlichen neuronalen Netze als auch der numerischen Optimierungsverfahren allgemein, weisen eine extrem große technische Relevanz und vielseitige Anwendungen auf. Gleichzeitig sieht man in beiden Bereichen noch viel Potential für weitere Entwicklungen. Sei es im Bereich der neuronalen Netze die Verwendung und das effektive Training rekurrenter Netze, oder im Fall der numerischen Optimierung eine weitere Verringerung der Startwertabhängigkeit und die Möglichkeit, sicher globale Minima zu erreichen. Es bietet sich also ein weites Feld an relevanten Problemstellungen für die Zukunft.

Literaturverzeichnis

[1] D. Stein and C. Newman, *Spin Glasses and Complexity*. Primers in complex systems, Princeton University Press, 2013.

[2] K. Mainzer, *Komplexe Systeme und Nichtlineare Dynamik in Natur und Gesellschaft: Komplexitätsforschung in Deutschland auf dem Weg ins nächste Jahrhundert*. Springer Berlin Heidelberg, 2013.

[3] J. Carrasquilla and R. G. Melko, "Machine learning phases of matter," *Nat Phys*, vol. 13, pp. 431–434, 05 2017.

[4] C. W. Eurich, "Neuronale netze," in *Lexikon der Physik*, vol. 4, pp. 81–86, Heidelberg: Spektrum, 2000.

[5] M. A. Nielsen, *Neuronal Networks and Deep Learning*. Determination Press, Jan 2017.

[6] M. Punk, "Numerical optimization using flow equations," *Phys. Rev. E*, vol. 90, p. 063307, Dec 2014.

[7] E. Allgower and K. Georg, *Numerical continuation methods: an introduction*. Springer series in computational mathematics, Springer-Verlag, 1990.

[8] L. Held and D. Bové, *Applied Statistical Inference: Likelihood and Bayes*. Springer Berlin Heidelberg, 2013.

[9] C. E. Shannon, "A mathematical theory of communication," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 5, pp. 3–55, Jan. 2001.

[10] E. T. Jaynes, "Information theory and statistical mechanics," *Phys. Rev.*, vol. 106, pp. 620–630, May 1957.

[11] J. Skilling, *Classic Maximum Entropy*, pp. 45–52. Dordrecht: Springer Netherlands, 1989.

[12] J. Skilling, *The Axioms of Maximum Entropy*, pp. 173–187. Dordrecht: Springer Netherlands, 1988.

[13] M. Bollhöfer and V. Mehrmann, *Numerische Mathematik: Eine projektorientierte Einführung für Ingenieure, Mathematiker und Naturwissenschaftler*. Vieweg+Teubner Verlag, 2013.

[14] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[15] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[16] C. Bishop, "Exact calculation of the hessian matrix for the multilayer perceptron," in *Neural Computation*, vol. 4, pp. 494–501, January 1992.

[17] Y. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, "Effiicient backprop," in *Neural Networks: Tricks of the Trade, This Book is an Outgrowth of a 1996 NIPS Workshop*, (London, UK), pp. 9–50, Springer-Verlag, 1998.

[18] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition*. Springer Series in Statistics, Springer New York, 2009.

[19] A. Meister, *Numerik linearer Gleichungssysteme: Eine Einführung in moderne Verfahren. Mit MATLAB®-Implementierungen von C. Vömel*. Springer Fachmedien Wiesbaden, 2014.

[20] D. Werner, *Funktionalanalysis*. Springer-Lehrbuch, Springer Berlin Heidelberg, 2011.

[21] K. H. Fischer and J. A. Hertz, *Spin Glasses*. Cambridge Studies in Magnetism, Cambridge University Press, 1991.

[22] F. Barahona, "On the computational complexity of ising spin glass models," *Journal of Physics A: Mathematical and General*, vol. 15, no. 10, p. 3241, 1982.

[23] S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber, "Gradient flow in recurrent nets: the difficulty of learning long-term dependencies," 2001.

[24] M. D. Zeiler and R. Fergus, *Visualizing and Understanding Convolutional Networks*, pp. 818–833. Cham: Springer International Publishing, 2014.

[25] Theano Development Team, "Theano: A Python framework for fast computation of mathematical expressions," *arXiv e-prints*, vol. abs/1605.02688, May 2016.

Erklärung

Hiermit erkläre ich, die vorliegende Arbeit selbständig verfasst zu haben und keine anderen als die in der Arbeit angegebenen Quellen und Hilfsmittel benutzt zu haben.

München, den 29. Mai 2017

Johannes Flommersfeld