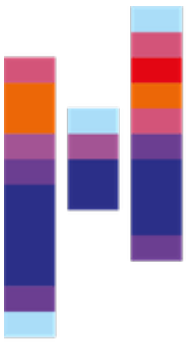

Hochschule
Flensburg
University of
Applied Sciences

Neuroevolution mit MPI

Analyse und Optimierung von NEAT für ein verteiltes System

Masterthesis
zur Erlangung des akademischen Grades
Master of Science (M.Sc.)
im Studiengang Angewandte Informatik
an der Hochschule Flensburg

Simon Hauck
Matrikelnummer: 660158

Erstprüfer: Prof. Dr. rer. nat. Tim Aschmoneit
Zweitprüfer: Prof. Dr. rer. nat. Torben Wallbaum

8. Juli 2020

Neuroevolutionäre Algorithmen sind ein mögliches Optimierungsverfahren für neuronale Netze. Abhängig von dem verwendeten Algorithmus können die Gewichte der Verbindungen im Netz und die Struktur entwickelt und optimiert werden.
Der Optimierungsprozess ist, unabhängig vom Verfahren, sehr aufwändig und dementsprechend zeit- und rechenintensiv. Für eine schnellere Durchführung des Trainingsprozesses bieten sich Algorithmen an, die gut parallelisierbar sind. Die benötigte Ausführungszeit dieser kann durch Hinzufügen weiterer Rechenknoten mit geringem Aufwand maßgeblich reduziert werden.
Neuroevolutionäre Algorithmen bieten sich aufgrund der Verfahrensweise und der vielen unabhängigen neuronalen Netzen für eine parallele Ausführung an.
In dieser Arbeit wird, stellvertretend für neuroevolutionäre Algorithmen, der NeuroEvolution of Augmenting Topologies (NEAT) Algorithmus betrachtet. Dieser wurde im Jahr 2002 veröffentlicht und ist im Vergleich zu den damals bekannten Algorithmen besonders effizient. Zudem dient der Algorithmus als Grundlage für viele Erweiterungen. Die erhaltenen Ergebnisse dieser Arbeit lassen sich somit gut auf ebendiese Erweiterungen übertragen.
Im ersten Schritt dieser Arbeit wird die Laufzeit des NEAT Algorithmus mit verschiedenen Optimierungsaufgaben analysiert. Mit den erhaltenen Ergebnissen wird eine parallelisierte Implementierung erstellt. Diese führt mit unterschiedlich vielen Rechenknoten dieselben Optimierungsaufgaben durch. Am Ende dieser Arbeit werden die Ergebnisse von beiden Implementierungen verglichen.

Inhaltsverzeichnis

Abbildungsverzeichnis

Akronymverzeichnis

NEAT NeuroEvolution of Augmenting Topologies

KNN Künstliche neuronale Netze

PNS Periphere Nervensystem

ZNS Zentrale Nervensystem

tanh Tangens Hyperbolicus

1 Motivation

1.1 Problemstellung

1.2 Ziel der Arbeit

1.3 Struktur der Arbeit

2 Grundlagen

2.1 Neuronale Netze

Klassische Algorithmen in der Informatik beschreiben mit welchen Schritten ein spezielles Problem gelöst werden kann. In vielen Anwendungsfällen, wie zum Beispiel beim Sortieren einer Liste, verwenden Computersysteme diese und lösen das gegebene Problem schneller und effizienter als es Menschen möglich ist.

Dennoch gibt es Aufgaben, die von Menschen ohne Aufwand gelöst werden, aber Computersysteme vor große Herausforderungen stellen. Hierzu zählt unter anderem die Klassifizierung von Bildern. Ein Mensch kann Bilder von Hunden und Katzen unabhängig von Blickwinkel und Bildqualität unterscheiden beziehungsweise richtig zuordnen. Trotzdem lassen sich für solche Probleme keine klassischen Algorithmen finden, da die Lösung von vielen subtilen Faktoren abhängt [1].

In vielen dieser Aufgabenfelder werden Künstliche neuronale Netze (KNN) eingesetzt, welche von den biologischen neuronalen Netzen inspiriert sind und zum Forschungsgebiet des maschinellen Lernens gehören. Die Grundlage für die KNN bildet die Arbeit von McCulloch und Pitts, in der sie 1943 ein einfaches neuronales Netz mit Schwellwerten entwickelt haben. Dies ermöglicht die Berechnung von logischen und arithmetischen Funktionen [2]. In den folgenden Jahrzehnten wird die Funktionsweise der neuronalen Netze weiterentwickelt und der Einsatz in verschiedensten Aufgabenfeldern ermöglicht. Hierzu zählen neben der Klassifizierung von Bildern [3] unter anderem das Erkennen und die Interpretation von Sprache [4], [5] sowie das selbständiges Lösen von Computer- und Gesellschaftsspielen [6], [7].

In diesem Kapitel wird zuerst ...

2.1.1 Biologische neuronale Netze

Wie bereits beschrieben orientiert sich das Fachgebiet der KNN an den erfolgreichen biologischen neuronalen Netzen, wie zum Beispiel dem menschlichen Gehirn [1]. In diesem Abschnitt werden die Eigenschaften betrachtet, die das Vorbild erfolgreich machen und für die KNN übernommen werden sollen. Im Zuge dessen wird ein grober Überblick über die Struktur und Funktionsweise des menschlichen Gehirns gegeben.

Jede Sekunde erfassen die Rezeptoren des menschlichen Körpers unzählige Reize, wie zum Beispiel Licht, Druck, Temperatur und Töne. Die Reize werden anschließend elektrisch oder chemisch kodiert und über Nervenbahnen an das Gehirn geleitet, welches die Aufgabe hat diese zu filtern, zu verarbeiten und entsprechend zu reagieren. Als Reaktion können

zum Beispiel Signale an entsprechende Muskeln oder Drüsen gesendet werden [8].

Bevor im nächsten Kapitel die Funktionsweise des Gehirns näher betrachtet wird, sollen drei Eigenschaften genannt werden, die klassische Algorithmen nicht besitzen beziehungsweise nur schwer umsetzen können, aber für biologische neuronale Netze keine Herausforderung sind. Ziel ist es, diese für die KNN zu übernehmen [1].

1. **Fähigkeit zu Lernen**
   Das menschliche Gehirn ist nicht wie ein klassischer Algorithmus für seine Aufgaben programmiert. Stattdessen besitzt es die Fähigkeit anhand von gegebenen Beispielen und oder einfachem Ausprobieren zu lernen [1]. Hierbei wird das gewünschte Ergebnis mit dem tatsächlich erhaltenen Ergebnis verglichen und das Verhalten entsprechend angepasst. Dies ermöglicht es Menschen verschiedenste Aufgabengebiete erfolgreich zu lösen und sich ändernden Anforderungen anzupassen.

2. **Fähigkeit zur Generalisierung**
   Allerdings kann nicht jedes mögliche Szenario für ein Aufgabenfeld durch Ausprobieren oder Beobachtung gelernt werden. Trotzdem trifft das Gehirn in den meisten Situationen plausible Lösungen, da es die Fähigkeit zur Generalisierung besitzt [1]. Das bedeutet, dass viele Situationen bereits bekannten Problemen zugeordnet werden können, mithilfe derer eine passende Verhaltensstrategie ausgewählt wird.

3. **Toleranz gegenüber Fehlern**
   Die Fähigkeit zu Generalisieren erlaubt auch eine hohe Fehlertoleranz gegenüber verrauschten Daten. Bei dem oben genannten Beispiel der Klassifizierung von Bildern kann ein Teil des Bildes fehlen oder unscharf sein und trotzdem kann das abgebildete Motiv richtig zugeordnet werden.

Struktur des menschlichen Gehirns

Die Forschungsgebiet der Neurowissenschaften befasst sich unter anderem mit dem menschlichen Gehirn, dessen Funktionsweise noch nicht vollständig nachvollzogen werden kann. Trotzdem ist schon seit 1861 durch die Arbeit von Paul Broca bekannt, dass es im menschlichen Gehirn verschiedene Regionen mit unterschiedlichen Aufgaben gibt [9]. Zum Beispiel wird das sogenannte Kleinhirn (Cerebellum) für einen Großteil der motorischen Koordination verwendet während an das Großhirn (Telencephalon) unter anderem visuelle Reize geleitet werden [1]. Trotz der unterschiedlichen Aufgaben haben alle Bereiche des Gehirns einen gemeinsamen Grundbaustein, die sogenannten Neuronen [9]. Im folgenden wird der Aufbau und die Funktionsweise von diesen oberflächlich im Bezug zu den später vorgestellten künstlichen Neuronen betrachtet. Für einen vollständigen Überblick und eine genaue Beschreibung der Vorgänge wird auf entsprechende Fachliteratur verwiesen.
Das menschliche Gehirn besitzt ungefähr 10^{11} einzelne Neuronen, deren schematischer Aufbau in Abbildung 2.1 dargestellt ist. Jedes Neuron besitzt einen Zellkern, der sich im Zellkörper (Soma) befindet. Von dem Zellkörper gehen mehrere Fasern aus, die Dendriten genannt werden [9]. An diesen befinden sich Synapsen, welche als Übertragungsstelle fungieren und elektrische oder chemische Signale von Rezeptoren oder anderen Neuronen

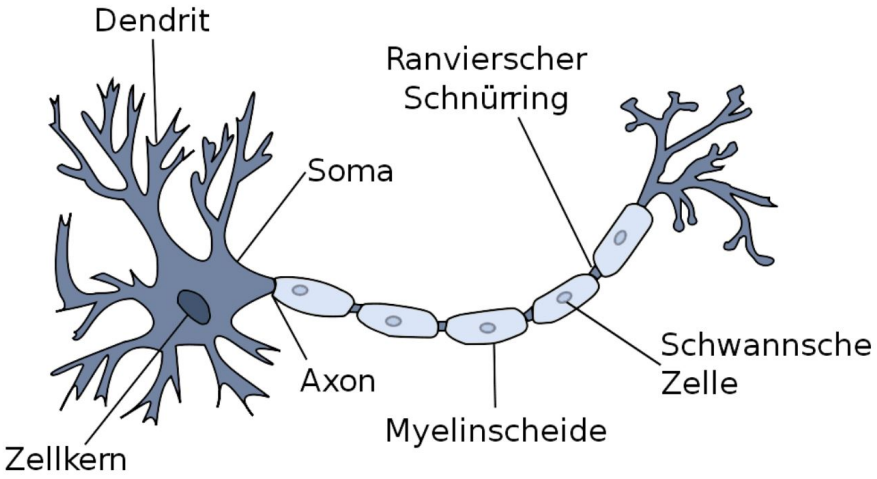

Abbildung 2.1: Schematische Abbildung einer Nervenzelle, Quelle [1].

empfangen [1]. Typischerweise empfängt ein Neuron Signale von 2000 und 10.000 anderen Nervenzellen [10].
Synapsen, die elektrische Signale empfangen, haben eine starke, direkte, nicht regulierbare Verbindung vom Sender zum Empfänger. Diese sind für hart kodierte Verhaltensmechanismen nützlich wie zum Beispiel den Fluchtreflex. Die chemische Synapse hingegen ist nicht direkt mit dem Sender verbunden, sondern durch den synaptischen Spalt getrennt [1]. Zur Übertragung eines elektrischen Signals wird dieses auf der präsynaptischen Seite in ein chemisches Signal kodiert, indem Neutransmitter freigesetzt werden. Diese können über den synaptischen Spalt übertragen und anschließend auf der postsynaptischen Seite wieder in ein elektrisches Signal kodiert werden. Ein großer Vorteil dieser Übertragungsart ist die Regulierbarkeit [1]. Verschiedene Neurotransmitter können unterschiedliche Effekte auf das Neuron haben, beispielsweise anregend (exzitatorisch) oder hemmend (inhibitorisch) [11]. Zusätzlich kann die Menge der freigesetzten Neurotransmitter die Stärke des Signals beeinflussen [1]. Auf lange Zeit gesehen können auch neue Verbindungen entstehen oder alte aufgelöst werden. Es wird angenommen, dass dies die Grundlage des Lernens im menschlichen Gehirn ist [9].
Sowohl die erregenden als auch hemmenden Signale werden über die Dendriten an den Axonhügel weitergeleitet, welcher sich zwischen dem Soma und dem Axon befindet. Dort werden die Signale akkumuliert. Wird bei diesem Vorgang ein gewisser Schwellwert überschritten, wird ein elektrischer Impuls erzeugt der über das Axon weitergeleitet wird [11]. Das Axon ist typischerweise 1cm in Ausnahmen sogar bis zu einem 1m lang und von der Myelinscheide umgeben, die unter anderem Schutz vor mechanischer Überanspruchung bietet [9]. Zusammen mit den Ranvierschen Schnürringen ermöglicht diese zudem eine schnellere Weiterleitung des Aktionspotenzials [11]. Das Axon endet mit dem sogenannten Endknopf oder auch Axonterminal genannt. Dieses ist mit den Synapsen von anderen Neuronen verbunden und kann beim Eintreffen eines Signals die Neurotransmitter freisetzten und somit das Signal übertragen [11]. Typischerweise gibt ein einzelnes Neuron

sein Signal an 1000 bis 10.000 anderen Neuronen weiter, in Ausnahmefällen sogar an bis zu 150.000 andere Neuronen [10], die alle parallel arbeiten. So entsteht ein sehr großes und leistungsfähiges neuronales Netz.

2.1.2 Künstliche neuronale Netze

KNN sind ein mathematisches Modell, dass im Vergleich zum biologischen Vorbild stark vereinfacht und idealisiert. Trotzdem können unterschiedlichste mathematische Funktionen abgebildet werden. In diesem Kapitel wird die grundsätzliche Funktionsweise sowie die einzelnen Komponenten der KNN vorgestellt.

Betrachtet man ein KNN als Blackbox (TODO REFERENZ BILD), gibt es eine Menge von Eingabewerten, die in einem Eingabevektor kodiert sind und eine Menge an Ausgaben, die in einem Ausgabevektor kodiert sind [12]. Die Eingaben werden im Falle der KNN nicht durch Rezeptoren erfasst sondern durch ein Optimierungsproblem gegeben. Der Ausgabevektor soll das gewünschte Ergebnis enthalten. Die Interpretation von diesem variiert je nach Optimierungsproblem und Netzarchitektur.

Betrachtet man die Struktur der KNN sind einige Ähnlichkeiten zum biologischen Vorbild erkennbar. Diese werden im folgenden genauer betrachtet [10]:

1. **Neuronen**
   Ähnlich zu den biologischen neuronalen Netzen, besteht auch das KNN aus vielen Neuronen [10]. Dies sind einfache Recheneinheiten, die primitive Funktionen bestimmen können [12] und deren genaue Funktionsweise in Kapitel 2.1.3 erläutert wird. Vorweggenommen sei, dass ein Neuron mehrere Eingabewerte besitzt, welche gewichtet sind und akkumuliert werden. Hierbei entsteht ein skalarer Ausgabewert, der den Aktivierungsgrad des Neurons repräsentiert und von anderen Neuronen als Eingabe verwendet werden kann [1].

2. **Gerichtete gewichtete Verbindungen**
   Wie im vorherigen Punkt angedeutet, sind Neuronen über gerichtete Verbindungen miteinander vernetzt. Der Aktivierungszustand eines Neurons wird entsprechend der Verbindungen an die Zielneuronen weitergegeben, welche diesen Wert als Eingabe verarbeiten. Wie bei den biologischen neuronalen Netzen auch, können Eingaben unterschiedlich stark anregend und hemmend wirken. Dies wird bei den KNN über Gewichte in den Verbindungen realisiert [10].

3. **Struktur und Gewichte**
   Der Ausgabevektor eines KNN ist abhängig von der Struktur des Netzwerkes und der Gewichte in den einzelnen Verbindungen. Für das erfolgreiche Lösen eines Optimierungsproblems muss ein KNN die richtige Kombination von Neuronen, Netzwerkstruktur und gewichteten Verbindungen besitzen. Diese müssen durch Lernverfahren bestimmt werden, auf die in Kapitel 2.1.5 näher eingegangen wird.

Trotz der vorgestellten Ähnlichkeiten, gibt es sehr viele Unterschiede zwischen den biologischen neuronalen Netzen und den KNN. Beispiel hierfür ist der Größenunterschied. Das menschlichge Gehirn mit seinen 10^{11} Neuronen besitzt pro Neuron ungefähr 10^4 Verbindungen, während die meisten KNN nur 10^2 bis 10^4 Neuronen mit insgesamt 10^5 Verbindungen besitzen. Auch werden keine chemischen Effekte die auf benachbarte Neuronen wirken sowie zeitliche und räumliche Lokalitätsprinzipien beachtet [10]. Aus diesen Gründen sind die KNN keine Nachbildung der biologischen neuronalen Netzen sondern verwenden diese nur als Inspiration.

2.1.3 Das Neuron

In diesem Kapitel wird die genaue Funktionsweise der einzelnen Neuronen betrachtet. Hierfür werden drei Phasen vorgestellt, in denen die Ausgabe eines einzelnen Neurons berechnet wird. Betrachtet man ein KNN führen typischerweise mehrere Verbindungen zu einem Neuron j, welche von den Neuronen $i_1, i_2, ..., i_n$ ausgehen [1]. Dieses ist schematisch in Abbildung (TODO ABBILDUNG EINFÜGEN) dargestellt.

Propagierungsfunktion

Die Ausgabewerte $o_{i_1}, o_{i_2}, ..., o_{i_n}$ der Neuronen $i_1, i_2, ..., i_n$ werden als Eingabewerte für das Neuron j verwendet. Für jeden Eingabewert existiert ein entsprechendes Gewicht $w_1, w_2, ..., w_n$ [1]. Somit repräsentiert w_{ij} das Gewicht für die Verbindung von Neuron i zu Neuron j [10]. Die Propagierungsfunktion f_{prop} berechnet die Netzeingabe net_j, welche in der nächsten Phase weiterverwendet wird [1].

$$net_j = f_{prop}(o_1, o_2, ..., o_n, i_1, i_2, ..., i_n)$$

Die meist verwendete Propagierungsfunktion, welche auch in den späteren Beispielen genutzt wird, ist die gewichtete Summe. Hierbei werden, entsprechend der Formel, die Werte o_i mit dem entsprechenden Gewicht w_i multipliziert und aufsummiert [1]:

$$net_j = \sum_i (o_i \cdot w_{i,j})$$

Aktivierungsfunktion

Der Aktivierungszustand $a_j(t)$ gibt den Grad der Aktivierung von Neuron j zum Zeitpunkt t an [10]. Ein neuer Aktivierungszustand zum Zeitpunkt $t + 1$ wird mit der Aktivierungsfunktion f_{act} berechnet. Diese berücksichtigt nicht nur die Netzeingabe $net_j(t)$ sondern auch den vorherigen Aktivierungszustand $a_j(t)$ und den Schwellwert Θ der Aktivierungsfunktion [10]. Ein Schwellwert Θ_j, auch Bias genannt, ist dem Neuron j zugeordnet und gibt die Stelle an, an welcher die Aktivierungsfunktion die größte Steigung hat [1]. Somit kann die Berechnung der Aktivierung $a_j(t + 1)$ durch folgende Formel ausgedrückt werden [10]:

$$a_j(t + 1) = f_{act}(a_j(t), net_j, \Theta_j)$$

Bei der Berechnung ist der Schwellwert Θ besonders wichtig. Oftmals verwenden einige oder alle Neuronen eines KNN dieselbe Aktivierungsfunktion, die Schwellwerte hingegen unterscheiden sich je nach Neuron. Des weiteren sei angemerkt, dass die vorherige Aktivierung $a_j(t)$ je nach Netzstruktur oft nicht bei der Berechnung berücksichtigt wird [1]. Zudem wird in der Praxis bei Verwendung der gewichteten Summe als Propagierungsfunktion der Schwellwert eines Neurons oft schon in der ersten Phase miteinbezogen. Hierdurch ändert sich die Berechnung der Netzeingabe zu $net_j = \sum_i (o_i \cdot w_{i,j}) - \Theta_j$. Bei der Berechnung der Aktivierungsfunktion gilt dann $\Theta_j = 0$.

Je nach Anwendungsgebiet können verschiedene Aktivierungsfunktionen mit unterschiedlichen Eingenschaften eingesetzt werden, von denen vier in Abbildung (TODO ABBILDUNG) dargestellt sind. Im folgenden wird angenommen, dass gilt $\Theta_j = 0$.
Das einfachste Beispiel für eine Aktivierungsfunktion ist die sogenannte binäre Schwellwertfunktion, welche abhängig vom Schwellwert Θ nur die Werte 0 und 1 zurückgeben kann [1]. Die Formel hierfür ist:

$$f_{act}(net_j) = \begin{cases} 1 & \text{wenn } net_j \geq 0 \\ 0 & \text{wenn } net_j < 0 \end{cases}$$

Allerdings ist diese Funktion an ihrem Schwellwert nicht differenzierbar und ansonsten ist der Wert der Ableitung immer 0 [1]. Diese Eigenschaften machen sie ungeeignet für bestimmte Lernverfahren, wie zum Beispiel den Backpropagation Algorithmus auf welchen kurz in Kapitel 2.1.5 eingegangen wird [1]. Dieses Problem kann durch die Verwendung einer Sigmoidfunktion gelöst werden. Zwei bekannte Beispiele für Sigmoidfunktionen sind die logistische Funktion und der Tangens Hyperbolicus (tanh) [13]. Die logistische Funktion kann Werte von 0 bis 1 annehmen und durch einen entsprechenden Parameter T bezüglich der x-Achse gestreckt und gestaucht werden [1]. Berechnet wird sie mit:

$$f_{act}(net_j) = \frac{1}{1 + e^{-T \cdot net_j}}$$

Allerdings können neuronale Netze je nach Verfahren schneller optimiert werden, wenn das durchschnittliche Gewicht aller Verbindungen nahe 0 ist. In diesem Fall ist die tanh Funktion besser geeignet, da sie Werte zwischen -1 und 1 annehmen kann [13]. Das letzte hier vorgestellte Beispiel ist die sogenannte *Rectifier* Funktion. Diese wird oft in Zusammenhand mit dem Backpropagation Algorithmus erfolgreich eingesetzt und erzielt mit diesem schneller bessere Optimierungsergebnisse [14]. Berechnet wird sie mit:

$$f_{act}(net_j) = max(0, net_j)$$

Ausgabefunktion

Die Ausgabefunktion f_{out} berechnet die Ausgabe o_j von Neuron j. Als Eingabewert wird die Aktivierung a_j verwendet [10]. Somit ist die Funktion definiert mit:

$$o_j = f_{out}(a_j)$$

In der Praxis ist die Ausgabefunktion, ähnlich wie die Aktivierungsfunktion, meistens global für alle Neuronen definiert. Zudem wird oft die Identitätsfunktion verwendet. In diesem Fall gilt $o_j = a_j$ [1]. Dies gilt auch für die später vorgestellten Beispiele. Ist die Ausgabe o_j berechnet, kann sie als Eingabewert für andere verbundene Neuronen dienen.

2.1.4 Netzstrukturen

Aus dem vorherigen Kapitel ist ersichtlich, dass die Gewichte einen großen Einfluss auf das Ergebnis eines einzelnen Neuron haben. Das Ergebnis von einem KNN wird neben den Gewichten auch von der Anzahl an Neuronen sowie deren Verbindungsstruktur beeinflusst. Je nach Optimierungsproblem können unterschiedliche Varianten eingesetzt werden, welche in diesem Kapitel genauer vorgestellt werden.

Unabhängig von der Struktur, besitzt jedes KNN Eingabe- und Ausgabeneuronen. Optional kann ein KNN beliebig viele verdeckte Neuronen enthalten. Diese werden auch als *Input*-, *Output*- und *Hidden*-Neuronen bezeichnet [10]. Die Anzahl der Eingabe- bzw. Ausgabeneuronen ist abhängig von der Größe des Eingabe- bzw. Ausgabevektors. Für jedes Element in den Vektoren gibt es ein entsprechendes Neuron (TODO ABBILDUNG). Bei vielen Netzstrukturen werden die Neuronen des KNN verschiedenen Schichten zugeordnet. In der ersten Schicht befinden sich die Eingabeneuronen und in der letzten die Ausgabeneuronen. Dazwischen befinden sich n Schichten mit verdecken Neuronen [10].

Bei der Berechnung eines KNN werden die Werte des Eingabevektors in die entsprechenden Eingabeneuronen gesetzt. Nachdem alle Neuronen berechnet wurden, bilden die Werte der Ausgabeneuronen den Ausgabevektor. Die verstecken Neuronen befinden sich zwischen den Eingabe- und Ausgabeneuronen und werden so genannt, da ihr Ausgabewert vor dem Anwender verborgen ist. Je nach Art des Verbindungsmuster können KNN einer von zwei Gruppen zugeordnet werden. Die erste Gruppe enthält KNN ohne Rückkopplung, welche auch *feedforward*-Netze genannt werden. Die zweite Gruppe umfasst Netze mit Rückkopplung, welche auch als *recurrent*-Netzwerke bezeichnet werden [10].

Netze ohne Rückkopplung

Die Definition der *feedforward*-Netze ist einfach: Es darf keine Verbindung von einem Neuron j ausgehen, welche wieder zu sich selbst führt. Dabei ist es irrelevant ob eine direkte oder indirekte Verbindungen über Zwischenneuronen besteht. Somit entsteht ein azyklischer Graph [10] und das KNN kann infolgedessen keinen keinen internen Zustand besitzen. Für die gleiche Eingabe immer dasselbe Ergebnis berechnet. Innerhalb dieser

Kategorie gibt es zwei Untergruppen, die ebenenweise verbundenen KNN und die KNN welche über sogenannte *shortcut* Verbindungen verfügen.
Bei den rein ebenenweise verbundenen KNN sind die Eingabewerte eines Neurons immer aus der vorherigen Schicht. Der berechnete Ausgabewert eines Neurons wird nur an die Neuronen der nächsten Schicht weitergeleitet [10]. Ein Beispiel hierfür ist in Abbildung (TODO ABBILDUNG) dargestellt.
Im Gegensatz dazu stehen die KNN mit *shortcut* Verbindungen können. Eine *shortcut* Verbindung kann eine oder mehrere Schichten überspringen. Für gewisse Optimierungsprobleme, unter anderem für das Beispiel in Kapitel (TODO CHAPTER), können so kleiner KNN erzeugt werden [10].

Netze mit Rückkopplung

Netze mit Rückkopplung werden oft auch in Schichten dargestellt, allerdings kann ein KNN sich je nach Art selbst beeinflussen indem Zyklen in der Berechnung entstehen, wodurch das Zwischenspeichern von Werten ermöglicht wird [10]. Somit wird das Ergebnis sowohl durch die Eingabewerte des KNN als auchd durch die vorherigen Berechnungen beeinflusst [15]. Auch die Netze mit Rückkopplung können vier verschiedenen Untergruppen zugeordnet werden [10].

1. Bei KNN mit direkter Rückkopplung können Neuronen Verbindungen zu sich selbst haben (TODO ABBILDUNG). Dadurch können sie ihre Aktivierung verstärken oder abschwächen [10].

2. Netze mit einer indirekten Rückkopplung erlauben im Gegensatz zu den *feedforward*-Netzen auch Verbindungen in die vorherige Schicht (TODO ABBILDUNG) [10]. Wie bei der direkten Rückkopplung kann sich ein Neuron j selbst beeinflussen, wenn es seinen Ausgabewert an ein Neuron i der nächsten Schicht weiterleitet, welches eine Rückkopplung zu j hat [1].

3. KNN mit lateralen Rückkopplungen erlauben Verbindungen von Neuronen innerhalb einer Schicht (TODO ABBILDUNG), welche hemmend oder aktivierend wirken können. Oft entsteht dabei ein *Winner-Takes-All*-Schema, da das beste Neuron alle anderen hemmt und sich selbst aktiviert [1].

4. Bei den vollständig verbundenen Netze darf ein Neuron zu jedem anderen verbunden sein, außer sich selbst (direkte Rückkopplung). Ein KNN, in welchem jedes Neuron zu jedem anderen eine Verbindung hat, wird auch Hopfield-Netz genannt. Ein Beispiel hierfür ist in Abbildung (TODO ABBILDUNG) dargestellt [1].

2.1.5 Lernverfahren

Nachdem die Funktionsweise der Komponenten in einem KNN in den vorherigen Kapitel erläutert wurde, ist erkennbar, dass das erfolgreiche lösen eines Optimierungsproblems

von vielen Faktoren abhängt. In der Praxis ist es für komplexere Aufgaben nicht möglich, diese manuell zu bestimmen. Aus diesem Grund muss ein Optimierungsverfahren, auch als Lernverfahren bezeichnet, verwendet werden, mit dem Ziel einen Teil oder alle Parameter des KNN durch einen Algorithmus zu bestimmen. Typischerweise ist das Lernverfahren unabhängig von dem eigentlichen Optimierungsproblem und kann daher in verschiedenen Bereichen ohne großen zusätzlichen Aufwand eingesetzt werden.

Bevor im weiteren verschiedene Arten des Lernens näher betrachtet werden, stellt die Liste im Folgenden die theoretischen Möglichkeiten vor, wie ein KNN lernen bzw. optimiert werden kann [10]:

1. Modifizieren der Gewichte von Verbindungen
2. Modifizieren des Schwellwertes von Neuronen
3. Hinzufügen von neuen Verbindungen oder Neuronen
4. Entfernen von Verbindungen oder Neuronen
5. Ändern der Propagierungs, Aktivierungs-, und Ausgabefunktion

asfkajsfö

2.2 Evolutionäre Algorithmen

and [16] with Stanley und Miikkulainen

2.3 NEAT

2.4 MPI

3 Analyse

3.1 Anforderungen

3.2 Softwarearchitektur und Implementierung

3.3 Testsetup

3.4 Evaluation

4 Software Architektur und Implementierung

5 Evaluation

5.1 Testsetup

5.2 Ergebnisse

6 Zusammenfassung und Ausblick

Quellenverzeichnis

[1] David Kriesel. 2008. Ein kleiner überblick über neuronale netze. *Download unter http://www. dkriesel. com/index. php*.

[2] Warren S McCulloch und Walter Pitts. 1943. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5, 4, 115–133.

[3] Alex Krizhevsky, Ilya Sutskever und Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, 1097–1105.

[4] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath u. a. 2012. Deep neural networks for acoustic modeling in speech recognition: the shared views of four research groups. *IEEE Signal processing magazine*, 29, 6, 82–97.

[5] Daniel Andor, Chris Alberti, David Weiss, Aliaksei Severyn, Alessandro Presta, Kuzman Ganchev, Slav Petrov und Michael Collins. 2016. Globally normalized transition-based neural networks. *arXiv preprint arXiv:1603.06042*.

[6] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra und Martin Riedmiller. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

[7] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot u. a. 2016. Mastering the game of go with deep neural networks and tree search. *nature*, 529, 7587, 484.

[8] Werner Kinnebrock. 2018. *Neuronale Netze: Grundlagen, Anwendungen, Beispiele*. Walter de Gruyter GmbH & Co KG.

[9] Stuart Russell und Peter Norvig. 2013. Künstliche intelligenz. ein moderner ansatz, 3. ak. aufl. (2013).

[10] Andreas Zell. 2003. Simulation neuronaler netze. 4., unveränderte auflage. (2003).

[11] Clemens Kirschbaum. 2008. *Biopsychologie von A bis Z*. Springer-Verlag.

[12] Andreas Scherer. 2013. *Neuronale Netze: Grundlagen und Anwendungen*. Springer-Verlag.

[13] Yann A LeCun, Léon Bottou, Genevieve B Orr und Klaus-Robert Müller. 2012. Efficient backprop. In *Neural networks: Tricks of the trade*. Springer, 9–48.

[14] Xavier Glorot, Antoine Bordes und Yoshua Bengio. 2011. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, 315–323.

[15] Tsungnan Lin, Bill G Horne und C Lee Giles. 1998. How embedded memory in recurrent neural network architectures helps learning long-term temporal dependencies. *Neural Networks*, 11, 5, 861–868.

[16] Kenneth O Stanley und Risto Miikkulainen. 2002. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10, 2, 99–127.

Eidesstattliche Erklärung

This is the beginning