Parameteroptimierung bei der evolutionären Entwicklung neuronaler Topologien

Diplomarbeit

am Lehrstuhl für Kognitive Systeme

des Instituts für Informatik und Praktische Mathematik

der Christian-Albrechts-Universität zu Kiel

Prof. Dr. Gerald Sommer

vorgelegt von

L. Jochen Krause

Betreuer: Dr. Nils T. Siebel

Kiel, 8. November 2007

Für meine Familie

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

Abkz.	Begriff oder Beschreibung	
	deutsch	**englisch**
AIC	Gütekriterium	Akaike Information Criterion
ARMA	Modell zur Zeitreihenanalyse	AutoRegressive-Moving Average model
ANOVA	Varianzanalyse	Analysis of Variance
BFGS	Verfahren zur Funktionsminimierung	Broyden-Fletcher-Goldfarb-Shanno
BOT	virtueller Roboter	bot
CDF	Verteilungsfunktion	Cumulated Density Function
CGNE	Verfahren zur Lösung eines linearen Gleichungssystems	Conjugate Gradients on Normal Equations
CMA-ES	Evolutionäre Strategie zur Entwicklung von neuronalen Netzen	Covariance Matrix Adaptation - Evolutionary Strategy
DAG	gerichteter schleifenloser Graph	Directed Acyclic Graph
DCS	dynamische Zellstrukturen	Dynamical Cell Structures
DFP	Verfahren zur Funktionsminimierung	Davidon-Fletcher-Powell
DFS	Tiefensuche	Depth First Search
EA	Evolutionärer Algorithmus	Evolutionary Algorithm
EANT / EANT2	Evolutionäres Verfahren zur Entwicklung von neuronalen Netzen	Evolutionary Acquisition of Neural Topologies
EP	Evolutionäre Programmierung	Evolutionary Programming
ES	Evolutionäre Strategie	Evolutionary Strategy
$\mathbb{EXP}$	Komplexitätsklasse der Algorithmen mit exponentieller Laufzeit	$\mathbb{EXPTIME}$
FPGA	programmierbarer Logikbaustein	Field Programmable Gate Array
FPTAS	approximativer Algorithmus mit güteabhängiger polynomieller Laufzeit	Fully Polynomial Time Approximation Scheme
GCC	Übersetzungsprogramm für die Programmiersprachen C und C++	GNU project C and C++ compiler
GMRES	Verfahren zur Lösung eines linearen Gleichungssystems	Generalized Minimal Residual method
JRE	Laufzeitumgebung von Java	Java runtime environment
LM	Verfahren zur Funktionsminimierung	Levenberg-Marquardt
MLP	mehrschichtiges Netz aus Perzeptronen	Multi-Layer Perceptron
N	Normalverteilung	normal distribution
NN	Neuronales Netzwerk	Neural Network
NEAT	evolutionäres Verfahren zur Entwicklung von neuronalen Netzen	NeuroEvolution of Augmenting Topologies
$\mathbb{NP}$	Komplexitätsklasse der Algorithmen mit nicht-deterministischer Laufzeit	$\mathbb{NPTIME}$
o.B.d.A.	ohne Beschränkung der Allgemeinheit	without loss of generality, w.l.o.g.

Abkz.	Begriff oder Beschreibung	
	deutsch	**englisch**
$\mathbb{P}$	Komplexitätsklasse der Algorithmen mit polynomieller Laufzeit	$\mathbb{PTIME}$
PCA	Hauptkomponentenanalyse	Principle Component Analysis
PVM	Server-/Client-Serviceanwendung für verteiltes Rechnen	Parallel Virtual Machine
RBF	radiale Basisfunktion	Radial Basis Function
RMSE	Gütekriterium	Root Mean Squared Error
SDS	evolutionäres Verfahren zur Funktionsminimierung	Stochastic Downhill Simplex
SLN	einschichtiges neuronales Netz	Single-Layer Network
SOM	selbstorganisierende Karte	Self-Organizing Map
SR	evolutionäres Verfahren zur Funktionsminimierung	Stochastic Reflections
SUSE	Software- und System-Entwicklung	SuSE Linux
SVD	Verfahren zur Matrixzerlegung und zur Lösung eines linearen Gleichungssystems	Singular Value Decomposition
TS	Verfahren zur Entwicklung von neuronalen Netzen	Tiger Search
UID	eindeutige Identifikation	Unique ID
VARMAX	ein Modell zur Zeitreihenanalyse	Vector AutoRegressive Moving Average model with eXogenous variables
VCD	Kapazitäts- oder Mächtigkeitsmaß einer Funktionenklasse	Vapnik-Chervonenkis Dimension
XML	hierarchische Darstellung strukturierter Daten in Form von Textdateien	eXtensible Markup Language
XO	Kreuzung zweier Lebewesen	crossover
XOR	exklusives Oder	eXclusive OR

Algorithmenverzeichnis

Tabellenverzeichnis

1 Einleitung

Neuronale Netzwerke repräsentieren einen Versuch biologisches Lernen nachzubilden und stellen gleichzeitig eine Repräsentation sowie ein Verarbeitungsschema von Information dar. Für die angestrebte Analyse dieses Ansatzes und der konstruktiven Algorithmen[1] NEAT, EANT und TS werden in Abschnitt 1.1 kurz abstrakte Informationsmengen eingeführt, bevor Abschnitt 1.2 die allgemeinen Lernparadigmen vorstellt und Neuronale Netzwerke in diesen Kontext einordnet. Die Aufgabenstellung in Abschnitt 1.3 beschreibt das Ziel im Umgang mit neuronalen Netzen und fixiert Rahmen sowie Aufbau der Arbeit.

1.1 Information

Der Informationsbegriff ist schwer zu fassen, und die darstellungsunabhängige und kontextfreie Definition einer atomaren Informationseinheit ist bislang mangels einer übergeordneten (transzendenten!?) Basis nicht gelungen. In der in Shannon (1948) begründeten Informationstheorie und in der mehrfach begründeten Algorithmischen Informationstheorie - siehe Solomonoff (1964a,b), Kolmogorov (1965), Chaitin (1966), Chaitin (1969) und Zvonkin u. Levin (1970) - wird daher stets eine kontextbasierte Darstellung betrachtet. Erstere ist dabei auf Informationskodierung und -übertragung ausgerichtet und basiert auf dem (statistischen) Begriff der Entropie[2]. Letztere beschäftigt sich mit Komplexität und Berechenbarkeit der Kodierung bzw. Darstellung einer Information. Beide Theorien sind also für den Umgang mit Information relevant, bieten aber kaum einen Ansatz für die Analyse der Effektivität einer Informationsgewinnung und der Effizienz einer Informationsverarbeitung.

Als Kompromiss (für den Kontext der Neuronalen Netzwerke) wird deshalb die darstellungsunabhängige Repräsentation von Information als kontextabhängige und kleinste unteilbare (atomare) Eigenschaft eines Gegenstandes abstrakt definiert. Damit wird eine Analyse auf Basis von Informationsmengen, als möglichst redundanzfreien Mengen aus atomarer Information, machbar. Eine interessante Arbeit zu diesem Thema findet sich übrigens in der Dissertation Flueckiger (1995). Der hier gewählte Ansatz ergänzt also die Informationstheorie um das Konzept der abstrakten Informationsmenge.

Dazu stellt sich im Kontext der Neuronalen Netze die Frage, welche Informationsmenge diesen zum Lernen vorgelegt wird. Klassischerweise ist dies eine Menge von (Eingabe-) Vektoren, an denen das Problem der atomaren Informationseinheit gut veranschaulicht werden kann.

[1] Sofern nicht anders beschrieben, stehen die Algorithmenbezeichnungen NEAT und EANT im Folgenden synonym für die Implementierung NEAT4J bzw. die Weiterentwicklung EANT2.

[2] Die Entropie ist ein Maß (der Unordnung), das die kleinste zu kommunizierende Informationsmenge beschreibt, mit der ein Informationsungleichgewicht zwischen einem Sender und einem Empfänger ausgeglichen werden kann.

Ein Vektor besteht aus einer geordneten, endlichen Menge aus reellen Zahlen. Während die gängige Arbeitsweise diesen Vektor als atomare Informationseinheit betrachtet, soll dies nun abstrahiert geschehen. Das soll bedeuten, dass tatsächlich davon ausgegangen wird, dass Information in diesem Vektor enthalten ist. Unklar bleibt jedoch, welche Darstellung diese hat, und folglich auch, wie mit ihr optimal umzugehen ist. Anstatt den Vektor wie üblich komponentenweise dem Netz zu übergeben, ist ebenso die Übergabe eines Teils seiner charakteristischen[3] (u.U. binär kodierten) Eigenschaften z.B. in Form von Vorzeichen, Ordnung, paarweiser oder mehrfach auftretender Eigenschaften, Größenordnungen usw. möglich. Entsprechende Sensoren könnten eine Repräsentation menschlicher Sinnesorgane darstellen.
Im Rahmen der Arbeit aber findet sich die Betrachtung abstrakter Informationsmengen und deren effektiver Erzeugung und effizienter Verwendung auf Basis von Eingabevektoren wieder. Ferner wird an einigen Stellen ein Effizienzmaß als informationsbasiertes Vergleichskriterium für Algorithmen angedacht, welches zukünftige Arbeiten motivieren könnte.

1.2 Lernen

Lernen ist als genereller Sinn und Zweck eines jeden Algorithmus auffassbar, wobei allgemein zwischen drei disjunkten Lernparadigmen unterschieden wird:

- Überwachtes Lernen (supervised learning) erfordert einen Lehrer, der eine Menge aus Paaren von Eingabe und Sollausgabe vorgibt. Das Lernziel besteht jeweils in der Minimierung der Differenz (des Residuums) von geforderter und tatsächlicher Ausgabe.

- Unüberwachtes Lernen (unsupervised learning) erfordert eine Menge von Daten ohne weitere Vorgaben. Das Ziel wird durch die Daten vorgegeben und besteht im Auffinden von Strukturen (z.B. Clustern), sofern für diese ein geeigneter Fixpunkt existiert.

- Verstärkendes Lernen (reinforcement learning) erfordert eine kontextabhängige Maßfunktion. Diese bewertet den Kontext in Abhängigkeit einer Aktion und gibt eine positive oder negative Rückmeldung, so dass gelernt werden kann.

Werden in diesem Kontext die von einem Algorithmus benutzten Daten als Informationsmenge interpretiert, so vergrößert er diese unter Hinzunahme von Hilfsmitteln[4] und erlernt dabei gemäß seiner Konstruktion einen Lösungvorschlag für die

[3] Dies ist als Analogon zum Begriff suffizienter Statistiken gemeint.
[4] Dies können beispielsweise Neuronale Netzwerke sein, die mit der gleichen Zielsetzung ihrerseits Propagierungs- und Aktivierungsfunktionen als Hilfsmittel einsetzen.

gestellte Aufgabe. Aus diesem Grund wird die Sichtweise mit abstrakten Informationsmengen als ein mächtiges Werkzeug bei der Analyse von Algorithmen bzw. Lernprozessen angesehen und dementsprechend verwendet.
Ferner bildet Lernen eine Obermenge für allgemeine Optimierungsverfahren. Beispielsweise entspricht das Überwachte Lernen dem Verfahren der kleinsten Quadrate, eine Iteration entspricht dem Durchlauf eines (u.U. schrittweise entstehenden) Datensatzes und die Anwendung von Bootstrapping einer randomisierten Iteration. Keine Ausnahme in dieser Anschauung des Lernens sind Konstruktion und Training eines Neuronalen Netzwerkes.
Die Verwendung eines neuronalen Netzes kann dabei über seine Darstellung als Funktion[5] f, die aus Daten x Information erzeugt, welche als Funktionswert $f(x)$ an den Ausgabeneuronen abzulesen ist, motiviert werden. Sein hauptsächlicher Vorteil manifestiert sich in der Mehrfachbaumstruktur, die algorithmisch direkt veränderbar ist. Diese Spezialisierung gegenüber gewöhnlichen (unzugänglichen) Funktionen stattet neuronale Funktionen mit neuen Eigenschaften aus. Wird z.B. ein Neuronales Netzwerk trainiert bzw. funktional optimiert, finden sich anschließend bestimmte Informationen leicht zugänglich in seiner Netzstruktur wieder. Somit erschließt sich bei der Optimierung einer neuronalen Funktion deren Struktur als neue Dimension. Damit können (Lern-) Algorithmen entworfen werden, denen theoretisch der gesamte Funktionenraum als Lösungsraum zur Verfügung steht.
Die nahezu unbegrenzte Plastizität solcher Funktionen gestaltet den Umgang mit ihnen allerdings schwierig, denn mit jedem Parametervektor, der fast zwangsläufig mit einer Vergrößerung der Funktion hinzukommt, steigt die Varianz einer Lösung[6], vergleiche auch Bias-Varianz-Dilemma in Bishop (2004) oder Duda u. a. (2004). Die üblicherweise benutzten Netze werden deshalb auf bestimmte starre (teilweise biologisch motivierte) Komponenten festgelegt[7]. Die Suche nach besseren und allgemeineren Komponenten prägt daher jede Abhandlung über NN.

1.3 Aufgabenstellung

Die Arbeit soll eine konstruktive Auseinandersetzung mit (parameterisierten) Verfahren zur Entwicklung neuronaler Topologien nachvollziehen und aufzeigen, an welchen Punkten deren Verbesserungen möglich oder Grenzen erreicht sind. Das allgemeine Ziel stellt dabei die Konstruktion eines Lernalgorithmus dar, der zu einer Aufgabe ein neuronales Modell (Netzwerk) entwickelt, das diese und möglichst auch ähnliche Aufgabenstellungen löst.

[5] Der abstrakte Syntaxbaum einer gewöhnlichen Funktion ist anschaulich mit einem einfachen Netzwerk vergleichbar.

[6] Dieses Phänomen kann höchstens asymptotisch mit der Größe des zum Training verwendeten Datensatzes kompensiert werden, welcher aber oftmals begrenzt ist.

[7] In angelernten Netzwerken ergibt sich das Minimum der neuronalen Funktion komponentenweise pro Ausgabeneuron als der kleinste Wert, der an einem Ausgabeneuron ankommen oder von diesem berechnet werden kann. Bei typischen Aktivierungsfunktionen mit beschränktem Bild genügen zu dieser Bestimmung die Gewichte der eingehenden Verbindungen. Das Maximum läßt sich analog bestimmen.

Grundlegend werden in Kapitel 2 die erforderlichen Definitionen und ausgewählte Interpretationen gegeben. In Kapitel 3 erfolgt anschließend die Betrachtung einer effizienten Repräsentation von (veränderbaren) Neuronalen Netzwerken in Form einer geeigneten Datenstruktur und ihrer Operatoren. Bevor in Kapitel 4 ein zusammenfassender Überblick über die im Weiteren relevanten Gegenstände und Teilgebiete (anderer Disziplinen) gegeben wird.
Daran knüpft Kapitel 5 mit der detaillierten Einführung einer randomisierten Methode (CMA-ES) zur problemorientierten Optimierung eines Netzes mit statischer Struktur an. Zusätzlich werden alternative Optimierungstechniken angesprochen und kurz vorgestellt. Die dynamische Entwicklung der Netzstruktur findet dann in Kapitel 6 auf Basis evolutionärer Vorgehensweisen statt. Dort werden die konstruktiven Algorithmen NEAT und EANT sowie ein neuer Ansatz (TS) eingeführt.
Für diese sowie andere (neuronale) Lernalgorithmen wird danach in Kapitel 7 ein neuer Benchmark vorgeschlagen und auf dessen Grundlage in den Kapiteln 8 und 9 die Analyse der zuvor eingeführten Verfahren präsentiert. Die Diskussion der betrachteten Algorithmen und Untersuchungen geschieht dabei jeweils zusammenfassend am Ende eines Kapitels.
Abschließend wird in Kapitel 10 eine Schlussbetrachung in großem Rahmen mit Zusammenfassung und Ausblick auf zukünftige Problemstellungen bzw. Forschungsgegenstände gegeben.

2 Neuronale Netze

Der Versuch, die Funktionsweise des Gehirns zu verstehen und nachzubilden, ist alt und umfasst bis zum heutigen Tag eigentlich nur das Konzept der Neuronalen Netzwerke. Dieses besteht rudimentär aus kleinen funktionalen Einheiten (Neuronen), die über gewichtete Leitungen (Verbindungen) miteinander verbunden sind und zusammen das Netzwerk ergeben. Dabei werden mittlerweile anhand sukzessiv verallgemeinerter Neuronen drei Generationen unterschieden.
Die erste Generation basiert dabei auf einem binären Neuron names McCulloch-Pitts-Zelle und wird von dem Neurophysiologen und Kybernetiker Warren McCulloch (1899-1969) sowie dem Logiker und kognitiven Psychologen Walter Pitts (1923-1969) in der wegweisenden Arbeit McCulloch u. Pitts (1943) eingeführt. Lernalgorithmen für die Netzwerke dieser ersten Generation werden jedoch erst später entwickelt, so dass ein Netzwerk vorerst von außen zu designen ist. Erst der kognitive Psychologe Donald O. Hebb (1904-1985) erschafft in dem Werk Hebb (1949) die erste Lernregel zur Selbstmodifikation eines Neuronalen Netzwerkes.
Daraus entwickelt der Psychologe und Informatiker Frank Rosenblatt (1928-1969) in Rosenblatt (1958, 1962) die zweite Generation von Neuronen names Perzeptron mit den notwendigen Eigenschaften, um relativ flexible Lernalgorithmen zu ermöglichen. Zwar können einzelne Perzeptrone das einfache XOR-Problem[8] nicht lernen, sind aber dennoch so flexibel, dass sie bis heute die Grundlage der meisten neuronalen Netze bilden. Sie stellen auch die Grundlage der in dieser Arbeit betrachteten Netzwerke dar.
Ein interessanter Vorschlag für eine dritte Generation wird in Gerstner u. Kistler (2002) - abermals als eine Verallgemeinerung der vorhergehenden Generation - gemacht. Dieser erweitert die Informationskodierung eines Neurons um die Modellierung von dessen Feuerrate, der eine latent große Bedeutung innerhalb biologischer neuronaler Netze zugeschrieben wird.

2.1 Kanonische Darstellung

Ein Neuronales Netzwerk in kanonischer Darstellung ist ein um neuronale Einheiten erweiterter Mehrfachbaum (insbesondere eine Art spezieller abstrakter Syntaxbaum), wie er in Abbildung 1 in Kombination mit einem neuronalen Netz dargestellt wird. Die folgenden Definitionen formalisieren die Gegenstände der Betrachtung in (Top-Down-Manier).

[8] Das XOR-Problem ist eine klassische Aufgabe für Überwachtes Lernen und besteht im Lernen der binären XOR-Funktion:

$$f : \mathbb{B}^2 \to \mathbb{B},\ (x_0, x_1) \mapsto \begin{cases} 0 & \text{für } x_0 = 0 \text{ und } x_1 = 0 \\ 1 & \text{für } x_0 = 0 \text{ und } x_1 = 1 \\ 1 & \text{für } x_0 = 1 \text{ und } x_1 = 0 \\ 0 & \text{für } x_0 = 1 \text{ und } x_1 = 1 \end{cases}$$

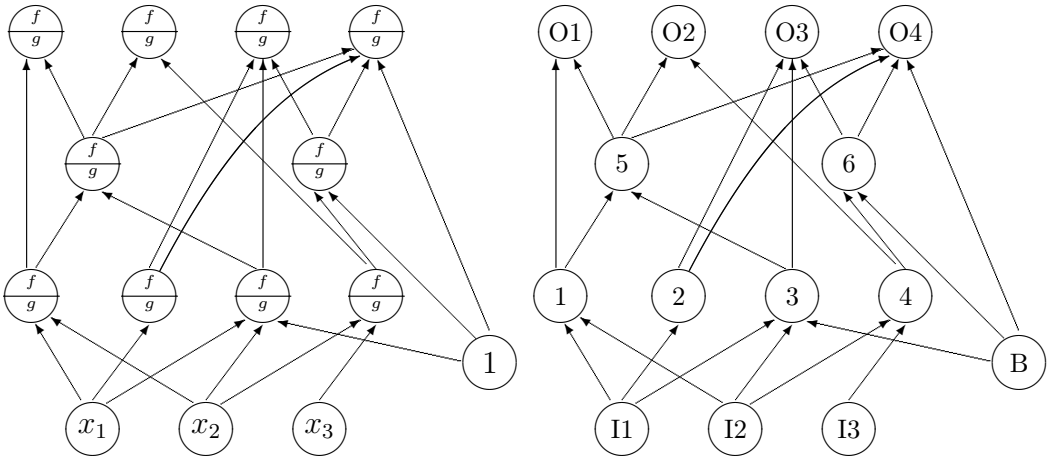


Abbildung 1: Beispiel eines ungewichteten Neuronalen Netzwerkes in der Darstellung eines Mehrfachbaumes mit und ohne Identifikatoren

Neuronales Netzwerk (Neuronale Funktion):
Ein Neuronales Netzwerk ist ein gewichteter (V, E)-Mehrfachbaum ohne Mehrfachkanten mit Neuronen als Knoten von unterschiedlicher Tiefe und Verbindungen als Kanten. Es besitzt eine feste Anzahl von Eingabe- und Ausgabeneuronen, variabel viele verdeckte Neuronen und Verbindungen sowie ein Biasneuron. Insbesondere ist nicht erforderlich, dass ein Knoten eine eingehende oder ausgehende Verbindung hat. Formal ist ein Netzwerk h durch folgende Parameter definiert:

- Dimension der Eingabeschicht sowie Dimension der Ausgabeschicht
- Neuronen (Propagierungsfunktionen, Aktivierungsfunktionen)
- Verbindungen (Gewichtsvektor)
- Struktur (Mehrfachbaum)

Wird wie üblich bei einer Funktion ceteris paribus nur die Eingabe betrachtet, dann ist ein Neuronales Netzwerk (siehe Abbildung 1) eine (vektorielle, multivariate) neuronale Funktion $h : \mathbb{R}^m \to \mathbb{R}^n$, welche den Urbildvektor $x \in \mathbb{R}^m$ auf den Bildvektor $h(x) \in \mathbb{R}^n$ abbildet.
Ein Netzwerk heißt zudem *voll verbunden*, wenn jedes Neuron einer Schicht mit jedem Neuron der nächsten Schicht verbunden ist. In Netzwerken ohne verdeckte Schicht existieren somit Verbindungen von jedem Eingabeneuron zu jedem Ausgabeneuron.

Schichten:
Eine Schicht wird nicht, wie bei MLPs üblich, als Menge der Knoten gleicher Tiefe, sondern als Menge gleichartiger Neuronen definiert. Entsprechend definiert werden die Begriffe Eingabeschicht, verdeckte Schicht und Ausgabeschicht.

Tiefe:

Die Tiefe eines Knotens ist definiert als die maximale Anzahl an Kanten, über die der Knoten von einem bestimmten Knoten aus erreicht werden kann. In einem Neuronalen Netzwerk bezieht sich die Tiefe eines Knotens auf dessen Erreichbarkeit von der Eingabeschicht aus bemessen.

Ausgabeneuron (Ausgabe):

Ein Ausgabeneuron ist ein spezielles Neuron, dessen Aktivierungswert einen indexfixierten Eintrag des Ergebnisvektors (Ausgabe) eines Neuronalen Netzwerkes darstellt.

Verdecktes Neuron:

Ein verdecktes Neuron ist kein Eingabe- oder Ausgabeneuron. Die verdeckten Neuronen sind zusammen mit den Verbindungen typischerweise die einzigen variablen Komponenten eines Netzes.

Eingabeneuron (Eingabe):

Ein Eingabeneuron ist ein spezielles Neuron, dessen Verhalten der bedingten Funktion $\forall c \in \mathbb{R}\, \forall x : \ f(x; c) = c$ entspricht und dem die Konstante c vorgegeben werden kann. Es besitzt keine eingehenden Kanten und propagiert als Aktivierungswert c stets den ihm zugewiesenen (indexfixierten) Eintrag eines Eingabevektors (Eingabe).

Biasneuron (Bias):

Ein Biasneuron existiert o.B.d.A. genau einmal in einem Neuronalen Netzwerk und entspricht der konstanten Funktion $\forall x : \ f(x) = 1$. Es gehört keiner der definierten Schichten an und stellt einem Netz den Aktivierungswert 1 zur Verfügung.

Neuron (Knoten):

Sei $i \in \mathbb{N}_{\leq |V|}$ der eindeutige Identifikator eines Neurons, und sei $m_i \in \mathbb{N}_{\leq |E|}$ die Anzahl seiner eingehenden Verbindungen. Ein Neuron ist eine (bedingte) Funktion, eine Berechnungseinheit eines Neuronalen Netzwerkes und ein Knoten in einem Graphen. Es besteht aus einer Propagierungsfunktion $f_i : \mathbb{R}^{m_i} \times \mathbb{R}^{m_i} \to \mathbb{R}$ und einer Aktivierungsfunktion $g_i : \mathbb{R} \to \mathbb{R}$, deren Hintereinanderausführung $g(f(x))$ seine Funktionalität ergibt. Im Folgenden wird daher die Schreibweise „(Propagierungs-, Aktivierungsfunktion)-Neuron“ benutzt, um ein Neuron zu charakterisieren.

Aktivierungsfunktion (Aktivierungswert):

Eine Aktivierungsfunktion $g : \mathbb{R} \to \mathbb{R}$ ist eine Funktion und Teil eines Neurons. Ihr Urbild besteht aus dem Funktionswert einer Propagierungsfunktion. Ihr Bild ist ein reellwertiger Skalar, das als Aktivierungswert bezeichnet und entlang aller ausgehenden Verbindungen des zugehörigen Neurons propagiert wird. Manchmal wird eine Aktivierungsfunktion ferner als bedingte Funktion definiert, um ihre Eigenschaften anpassen zu können.

Propagierungsfunktion (Propagierungswert):
Eine Propagierungsfunktion $f : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}$ ist eine Funktion und Teil eines Neurons. Ihr Urbild besteht aus dem Vektor der Gewichte und dem Vektor aller propagierten Aktivierungswerte der eingehenden Verbindungen. Ihr Bild ist ein reellwertiger Skalar und wird als Propagierungswert bezeichnet. In seltenen Fällen wird eine Propagierungsfunktion zudem als bedingte Funktion definiert, um ihre Eigenschaften anpassen zu können.

Verbindung (Kante):
Eine Verbindung ist eine Kante eines Graphen. Sie besitzt neben Ursprungs- sowie Zielknoten ein Gewicht $w \in \mathbb{R}$ und den Aktivierungswert des Ursprungneurons. Verbindungen können Schleifen beliebiger Länge im Netzwerk bilden, sofern dies zugelassen oder rekurrent[9] definiert wird. Eine ungewichtete Verbindung besitzt ein konstantes Gewicht, z.B. $w = 1$.

(V, E)-Mehrfachbaum (Polytree):
Ein Mehrfachbaum ist ein gerichteter, azyklischer und (V, E)-Graph (DAG), bestehend aus der Knotenmenge V der disjunkten Vereinigung der Eingabeknoten V_{INPUT}, der verdeckten Knoten V_{HIDDEN} und der Ausgabeknoten V_{OUTPUT} sowie einer Menge E von (gerichteten) Kanten, welche je zwei Knoten miteinander verbinden, und hat folgende Eigenschaften:

1. Eingabeknoten besitzen keine eingehenden Kanten.
2. Optional: Ausgabeknoten besitzen keine ausgehenden Kanten.

Die Knotenmengen V_{INPUT}, V_{HIDDEN} und V_{OUTPUT} werden auch als Eingabeschicht (input layer), verdeckte Schicht (hidden layer) und Ausgabeschicht (output layer), die Knoten selbst als Neuronen, bezeichnet. Sei $m := |V_{\text{INPUT}}|$ und sei $n := |V_{\text{OUTPUT}}|$.

2.2 Funktionsweise

Die Berechnung eines künstlichen Neuronalen Netzwerkes verläuft von der Eingabeschicht zur Ausgabeschicht und (vermutlich) im Gegenteil zum biologischen Neuronalen Netzwerk nicht kontinuierlich, sondern schrittweise. Für eine Auswertung des Netzes erhält jedes Eingabeneuron initial eine bestimmte Komponente des Eingabevektors $x \in \mathbb{R}^m$ vorgegeben und propagiert diese entlang seiner ausgehenden Kanten. Die nachfolgenden Neuronen (in der verdeckten Schicht oder der Ausgabeschicht) berechnen und feuern dann ihrerseits ihren Aktivierungswert, wenn an jedem ihrer Eingänge ein Signal anliegt. Dadurch entsteht bei der Berechnung eines Eingabesignals entsprechend der Tiefenstruktur des Netzwerkes eine stufenweise Ausbreitung des Eingabesignals. Diese (Vorwärts-) Propagierung endet, wenn an

[9] Eine rekurrente Verbindung liefert den letzten Aktivierungswert ihres Ursprungsneurons und entspricht damit dem Lag-Operator des Zeitreihenkontexts.

allen Eingängen der Ausgabeschicht ein Signal anliegt und die Ausgabeneuronen ihren Aktivierungswert errechnet haben. Ist das NN zyklisch oder existieren rekurrente Verbindungen, muss die Berechnung wiederholt werden, bis das Netzwerk einen stabilen Zustand erreicht hat. Je nach Kontext kann die Berechnung (z.B. aus Effizienz- oder Komplexitätsgründen) aber auch abgewandelt erfolgen, wie Kapitel 3 aufzeigt.
Ein analytischer Blick auf die Komponenten eines Neuronalen Netzwerkes offenbart zudem einige Interpretationen, anhand derer ein tieferes Verständnis aufgebaut werden kann. Den Interpretationen ist jedoch immer mit Vorsicht zu begegnen, weil die mit ihnen einhergehenden Abstraktionen die betrachteten Gegenstände oft stark vereinfachen.
Im Folgenden sei o.B.d.A. $t \in \mathbb{N}$ stets geeignet gewählt, und $w, x \in \mathbb{R}^t$ seien Vektoren. Folgende grundlegende Eigenschaften bzw. Informationen bietet ein künstliches Neuronales Netzwerk:

- Die Kanten eines neuronalen Netzes tragen zu seiner Funktionsweise mit ihrer Existenz als Topologiebeschreibung und mit ihrer Informationsmenge (Herkunft und Gewicht) als Funktionsbestandteil bei. Die Kanten können somit rudimentär als biologische Nervenfaser zwischen biologischen Nervenzellkörpern interpretiert werden.

- Die Knoten der Eingabeschicht stellen lediglich die Dimension des Eingaberaumes als Topologiebeschreibung und die Komponenten des aktuellen Netzeingabevektors als Funktionsargumente zur Verfügung. Die Eingabeknoten können somit grob als spezielle biologische Nervenzellkörper interpretiert werden.

- Das Biasneuron ist die (eindeutige) Quelle des konstanten 1-Signals. Es ist bislang biologisch nicht zu interpretieren und wird benutzt, um z.B. die lineare Propagierungsfunktion $\sum_{j=\mathbf{1}}^{m_i} x_j w_j$ zu einer affin-linearen Funktion $\sum_{j=\mathbf{0}}^{m_i} x_j w_j$ mit $x_0 = 1$ umzuformen.

- Die Knoten der verdeckten Schicht beeinflussen einerseits über ihre Anzahl als Topologiebeschreibung die theoretische Plastizität des Netzes und andererseits über ihre Propagierungs- und Aktivierungsfunktion dessen funktionale Eigenschaften.

- Die Knoten der Ausgabeschicht liefern mit ihrer Anzahl als Topologiebeschreibung die Dimension des Ausgaberaumes und können ebenfalls über ihre Propagierungs- und Aktivierungsfunktion die Funktionalität des Netzes beeinflussen.

- Die Wahl von Propagierungsfunktion $f : \mathbb{R}^{m_i} \times \mathbb{R}^{m_i} \to \mathbb{R}$ und Aktivierungsfunktion $g : \mathbb{R} \to \mathbb{R}$ ist bis auf die Sicherstellung der Hintereinanderausführbarkeit $g(f(x))$ und deren Restriktion auf Urbild- und Bilddimension frei. Es muss allerdings bemerkt werden, dass mangels Entscheidungskriterien in

den meisten Neuronalen Netzwerken nur ein einzelner Neuronentyp eingesetzt wird.

- Typischerweise wird als Propagierungsfunktion f das Skalarprodukt

$$< x, w >= \sum_{j=1}^{m_i} x_j w_j$$

von Gewichts- und Eingabe-/Signalvektor, also die gewichtete Summe des Signalvektors, benutzt. Damit läßt sich die Propagierungsfunktion als Linearkombination der eingehenden Signale oder als momentanes Erregungspotenzial einer biologischen Nervenzelle interpretieren. Weitere bekannte Propagierungsfunktionen sind z.B. die euklidische Norm des Abstands von Gewichts- und Signalvektor in RBF-Netzen oder das große Produkt $\prod_{j=1}^{m_i} x_j w_j$ aus Gewichts- und Signalvektor in Boolschen Netzen.

- Als Aktivierungsfunktion g wird zumeist eine Funktion eingesetzt, welche den propagierten Wert $f(x, w)$ in einer nichtlinearen Art abbildet[10]. Dies ist üblicherweise entweder eine sigmoide Funktion oder der Tangenshyperbolikus, welche die s.g. Kübelfunktion approximieren können, die in biologischen Nervenzellen bei ausreichender Erregung für die Impulsweitergabe zuständig ist. Damit läßt sich die Aktivierungsfunktion schlicht als einfache Funktion oder approximiative Aktivierung einer biologischen Nervenzelle interpretieren.

Ein allgemeines Entscheidungskriterium für die optimale Wahl von Netzwerkstruktur oder Propagierungs- und Aktivierungsfunktionen bzw. Neuronen existiert bislang nicht. Abschnitt 2.3.4 zeigt jedoch, wie beispielsweise eine pragmatische Forderung an die neuronale Funktion die Wahl bestimmter Neuronen begründen kann. Ansonsten besteht - wie bei anderen Optimierungsproblemen auch - eine Möglichkeit darin, Gütekriterien (siehe Abschnitt 4.6) zum Vergleich verschiedener Netzwerkmodelle zu Rate zu ziehen.

2.2.1 Skaleneffekte und Eingabedaten

Die übliche Art und Weise einem Neuronalen Netzwerk eine Eingabe zu präsentieren, ist die komponentenweise Identifikation des Eingabevektors mit den Eingabeneuronen. Der Wert des i-ten Eingabeneurons entspricht dann dem i-ten Wert im Eingabevektor. Diese am häufigsten anzutreffende Art der Eingabe forciert allerdings das Problem des Skaleneffektes.
Dieser ist eine Eigenschaft einer Aktivierungsfunktion und existiert, falls diese einen endlichen Grenzwert besitzt, also der Funktionswert für eine monoton wachsende oder fallende Reihe reeller Zahlen ab einem bestimmten Index nahezu identisch ist.

[10] Würde eine lineare Aktivierungsfunktion im gesamten NN verwendet, wäre das NN eine lineare Funktion, welche im Allgemeinen unzureichend plastisch ist. Siehe auch Abschnitt 2.4.

Für eine monoton steigende ($\forall x_0, x_1 : x_0 \leq x_1 \Rightarrow f(x_0) \leq f(x_1)$) Aktivierungsfunktion f mit beschränktem Bild zeigt das Beispiel der tanh-Funktion in Gleichung (1) den Skaleneffekt[11].

$$\forall a \geq 10 : \tanh(a) \approx 1 \quad \text{und} \quad \forall a \leq -10 : \tanh(a) \approx -1 \tag{1}$$

Ein $(< \cdot, \cdot >, \tanh)$-Neuron wird also für alle Propagierungswerte größer als ± 10 praktisch (insbesondere aufgrund der endlichen numerischen Genauigkeit eines Computers) denselben Aktivierungswert berechnen, was einem direkten Informationsverlust entspricht.
Existieren Neuronen mit Skaleneffekt in einem Netzwerk und wird wie üblich eine unbeschränkte Propagierungsfunktion verwendet, kann dieser Effekt prinzipiell im gesamten Netzwerk auftreten. Konstruktionsbedingt reagiert ein Neuronales Netzwerk dabei jedoch besonders sensitiv auf seine Eingabeschicht. Zur Begrenzung der Skaleneffekte bzgl. einer (unbekannten) Netzeingabe werden aus diesem Grund auch transformierte Eingabedaten verwendet.
Diese zweite Art der Eingabe besteht in einer Transformation des Eingabevektors[12], z.B. durch komponentenweise Abbildung mit einer Aktivierungsfunktion, bevor dieser den Eingabeneuronen wie bei der ersten Variante übergeben wird. Die Begrenzung der Skaleneffekte auf die Eingabeschicht funktioniert, weil die größere Anzahl der ihr nachfolgenden Neuronen somit nicht durch die Eingabedaten beeinträchtigt werden kann.
Eine Lösung für das Problem der Skaleneffekte besteht indes in der Verwendung unbeschränkter Aktivierungsfunktionen. Diese führen jedoch mit steigender Tiefe eines Neurons zu numerischen Problemen mit großen Zahlen, siehe dazu auch Abschnitt 2.4. Eine andere Lösung stellen hingegen zyklische (und damit nicht monotone) Aktivierungsfunktionen dar. Für das Beispiel der Sinusfunktion verhindert ihr 2π-langer Zyklus anschaulich den Skaleneffekt, obwohl ihr Bild $[-1, 1]$ ebenfalls beschränkt ist.
Eine Möglichkeit, um den Skaleneffekt zumindest beeinflussen zu können, ist zudem die Skalierung der verwendeten Aktvierungsfunktion. Abbildung 2 zeigt u.a. am Beispiel des Tangenshyperbolikus, wie sich die Grenze des Skaleneffektes verschieben läßt, und Abschnitt 9.5 zeigt Ergebnisse zu einigen skalierten Aktivierungsfunktionen im Kontext eines Algorithmus zur Konstruktion neuronaler Netzwerke.
Unterdessen müssen auftretende Skaleneffekte nicht zwingend zu einem numerischen Problem führen, wenn eine entsprechende Aktivierungsfunktion, wie z.B. der Tangenshyperbolikus oder die Sigmoidfunktion, eingesetzt wird. Unter Umständen fallen diese Effekte im Kontext einer Aufgabestellung sogar überhaupt nicht auf. Bei einer Gewichtsoptimierung stellen sie dagegen einen Grund dar, den initialen Gewichtsvektor - den Skaleneffekt neutralisierend - auf 0 zu setzen. Häufig stellt

[11] Das Phänomen der Skaleneffekte ist zum Beispiel der Grund für die „Flat Spots“ in Fahlman (1988, Seite 8).

[12] Diese Vorverarbeitung der Eingabe ist eigentlich nicht Teil eines künstlichen Neuronalen Netzwerkes und sollte daher konzeptionell getrennt (in einer vorgeschalteten Schicht) stattfinden.

sich dann ein Lernergebnis ein, das den Skaleneffekt zumindest für die präsentierten Daten vermeidet.

2.2.2 Skalierung der Ausgabe

Dem allgemeinen Verständnis nach ist anzunehmen, dass sich eine plastische Funktion aufgrund ihrer Plastizität einem geeigneten Kontext (z.B. der Lösung einer Aufgabe) hinreichend gut anpassen kann.
Dies gilt für Neuronale Netzwerke allerdings nur unter der Voraussetzung, dass die Ausgabeschicht dies zulässt und nicht in ihrem Bild, wie z.B. die Sigmoidfunktion auf das Intervall $]0, 1[$ oder der Tangenshyperbolikus auf $]-1, 1[$, beschränkt ist. Genügt das Bild der Ausgabeneuronen nicht den Anforderungen, kann die Ausgabe beispielsweise durch Multiplikation mit einem Skalar skaliert oder durch Abbildung mit einer geeigneten Funktion transformiert werden[13], wie z.B. im Fall des Benchmarks in Abschnitt 7.1.

2.3 Sichtweisen

Neuronale Netzwerke sind sehr flexibel einsetzbar, woraus sich viele Sichtweisen und Interpretationen ergeben, von denen die verallgemeinernden benutzt werden können, um Möglichkeiten zur neuronalen Optimierung aufzuspüren.
Zu Gunsten einer konzeptionellen Separierung der einzelnen Netzwerkschichten wird dabei vorgeschlagen, in allen Neuronen der Ausgabeschicht das Skalarprodukt $< \cdot, \cdot >$ als Propagierungsfunktion und die Identität id als Aktivierungsfunktion zu verwenden, und somit o.B.d.A. die Ausgabe auf eine Linearkombination von Eingabe- und verdeckten Neuronen zu reduzieren. Bei dieser Aufteilung kann die Eingabeschicht als Sensorbereich, die verdeckte Schicht als neuronale Funktionalität und die Ausgabeschicht als Optimierer aufgefasst werden.
Bislang ist jedoch keine Anschauung bekannt, die zu einem einheitlichen Ansatz oder einer generischen Vorgehensweise führt. Ob z.B. eine Schicht kontextabhängig arbeiten sollte oder nicht, ist deshalb eine offene Frage und variiert über die Lernalgorithmen.

2.3.1 Erste Interpretation

Gegeben sei ein Neuronales Netzwerk, dessen verdeckte Schicht aus ($< \cdot, \cdot >$, nichtlineare Aktivierungsfunktion)-Neuronen und dessen Ausgabeschicht aus ($< \cdot, \cdot >$, id)-Neuronen besteht. Fasst man die verdeckten Neuronen als (eine Auswahl von) Basisfunktionen eines geeigneten (unendlich-dimensionalen) Vektorraums auf, so formen Struktur und Gewichte des Netzwerkes eine Funktion bzw. einen Vektor in diesem Funktionalraum.

[13] Wie bei der Transformation der Eingabe ist die Nachverarbeitung eigentlich nicht Teil des neuronalen Konzeptes und sollte daher getrennt (in einer nachgeschalteten Schicht) stattfinden.

Die Propagierungsfunktion eines Nicht-Eingabeneurons bildet dabei aus den Aktivierungswerten der Vorgänger eine Linearkombination, und die Aktivierungsfunktion bestimmt die näheren Eigenschaften des Funktionenraumes. Eine ähnliche Betrachtung ist eine Krylow-Sequenz in einem Krylowraum[14], welcher jedoch u.a. Verschaltelungskomplexität und Nichtlinearität fehlen.
Diese Sichtweise beschreibt ein NN somit als eine Verflechtung transformierter Linearkombinationen in einem speziellen Funktionenraum.

2.3.2 Zweite Interpretation

Gegeben sei ein Neuronales Netzwerk, dessen verdeckte Schicht aus (Distanzfunktion $||x - w||_2$, RBF)-Neuronen und dessen Ausgabeschicht aus ($< \cdot, \cdot >$, id)-Neuronen besteht. Fasst man die verdeckten Neuronen als m_i-dimensionale (in die Struktur des NN eingebettete) RBF-Neuronen $r : \mathbb{R}^{m_i} \times \mathbb{R}^{m_i} \to \mathbb{R}^+$ auf, so kann das resultierende neuronale Netz beispielsweise als selbstorganisierende Karte (SOM) erkannt werden.
Wird das Netzwerk trainiert, lernt die verdeckte Schicht die optimalen Positionen ihrer m_i-dimensionalen Neuronen und die Ausgabeschicht die optimale Linearkombination. Etwas eingeschränkt durch die nicht-negative Ausgabe der verdeckten Neuronen kann hier ebenfalls die Sichtweise aus Abschnitt 2.3.1 herangezogen werden.
Diese Sichtweise zeigt die Flexibilität des neuronalen Konzeptes anhand der Überführbarkeit verschiedener neuronaler Konzepte[15] in die kanonische Darstellung aus Abschnitt 2.1.

2.3.3 Dritte Interpretation

Gegeben seien ein Neuronales Netzwerk und ein Datenquelle. Wird diese als Informationsquelle aufgefasst, kann das Verhalten des Netzes als eine Methode aus dem Bereich des „Data Minings“ interpretiert werden. Das Netzwerk ähnelt einer zu trainierenden Intelligenz, die mit bekannten Daten eine Aufgabe lösen muss. Für das Netz ist dabei ein Datum zugleich Eingabe und einzig verfügbare Information, mit der eine Teilaufgabe gelöst werden muss.
Genauer betrachtet beschränkt sich die verfügbare Information auf die Komponenten und die Ordnung des Eingabevektors, wobei das Netzwerk diese Informationsmenge durch seine Struktur vergrößert und als Muster in Form der Aktivierungswerte seiner Neuronen darstellt. Existiert dann für jede Eingabe ein genügend

[14] Seien $m, n \in \mathbb{N}$, sei $A \in \mathbb{C}^{n \times n}$ eine quadratische Matrix und $q \in \mathbb{C}^n$ ein Spaltenvektor. Ein Krylowraum $\mathcal{K} \subset \mathbb{C}^n$ ist definiert als die lineare Hülle der Reihe (Krylow-Sequenz):

$$\mathcal{K} := \mathcal{K}_m(A, q) = \operatorname{span}\left\{q, Aq, \ldots, A^{m-1}q\right\}$$

[15] Weitere Konzepte, wie z.B. endliche Automaten, sind einfach in Neuronale Netzwerke transformierbar.

einzigartiges und geeignetes Aktivierungsmuster im Netzwerk, kann damit die Aufgabe gelöst werden.
Diese Sichtweise beschreibt ein NN als Medium, das Information erzeugt und benutzt und damit einen funktionalen Informationsspeicher bildet.

2.3.4 Vierte Interpretation

Gegeben sei ein Neuronales Netzwerk, dessen verdeckte Schicht aus (Skalarprodukt, nichtkonstante und zyklische sowie beschränkte Funktion)-Neuronen und dessen Ausgabeschicht aus ($< \cdot, \cdot >$, id)-Neuronen besteht. Ein solches Netzwerk kann im Sinne einer mehrdimensionalen Polynomfunktion oder eines Wavelets auch als eine Art formbarer plastischer Wellenfunktion aufgefasst werden.
Bei einer Funktionsapproximation wird dann derjenige Parametersatz gesucht, der die neuronale Funktion am besten durch gegebene Stützstellen verlaufen läßt. Soll sich die Approximation dabei besonders glatt und nur wenig oszillierend bzw. schwach schwankend verhalten[16], sollte sie bekanntermaßen[17] auf ebenfalls möglichst glatten Funktionen - wie z.B. dem Tangenshyperbolikus (vergleiche auch Polynominterpolation) - aufbauen. Durch das Training eines Netzes wird diesem also die zur Aufgabe passende Schwingung beigebracht.
Diese Sichtweise beschreibt ein NN somit als allgemeinen (multivariaten) Funktionsapproximator mit dem Ziel eines perfekten Interpolationsverhaltens.

2.3.5 Fünfte Interpretation

Wird ein Problem bestehend aus bekanntem Urbild (Eingabe) und u.U. unbekanntem Bild (Referenz bzw. Sollausgabe) als Zuordnungsproblem (Klassifikationsproblem) aufgefasst, beschreibt ein Neuronales Netzwerk eine Relation, die einer Eingabe eine (approximative) Lösung bzw. eine Ausgabe zuordnet.
Zusammen mit der üblichen Definition einer Funktion als Relation zwischen zwei Mengen stellen Neuronale Netzwerke dann - z.B. analog zu Polynomfunktionen - eine Funktionenklasse dar.
Diese Sichtweise beschreibt ein NN somit als Repräsentant einer speziellen Funktionenklasse.

2.4 Alternative Neuronen

Das Perzeptron aus Rosenblatt (1958) mit seiner zweistufigen Funktionsweise bildet den Ausgangspunkt der Modellierung eines Neurons. Für ein Neuronales Netzwerk wird jedoch gängigerweise nur ein einziger Neuronentyp verwendet, da keine allge-

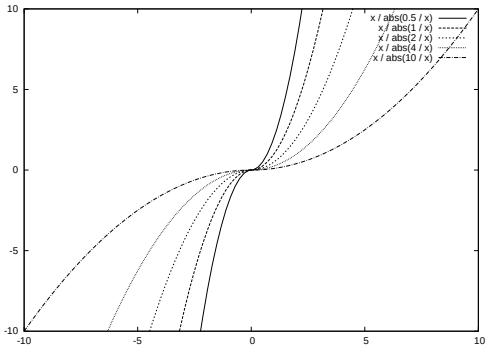
[16] Für das Ziel einer guten Interpolationseigenschaft sind dies typische Anforderungen an eine Funktionsapproximation.
[17] Diese Vorgehensweise ist für mehrdimensionale Daten bislang nur empirisch fundiert und nicht allgemein erforscht.

AbsAZ	:	$\frac{x}{\left\lvert\frac{z}{x}\right\rvert}$	SinZZ	:	$\sin(\frac{x}{z}) \cdot z$
AbsBZ	:	$\frac{x}{\left\lvert\frac{z}{x}\right\rvert} + x$	SinZ	:	$\sin(x) \cdot z$
SigmoidZ	:	$\frac{1}{\left(1+e^{(-z \cdot x)}\right)}$	TanhZZ	:	$\tanh(\frac{x}{z}) \cdot z$
SinCosZ		$\sin(x) \cdot \cos(x) \cdot z$	TanhZ	:	$\tanh(x) \cdot z$

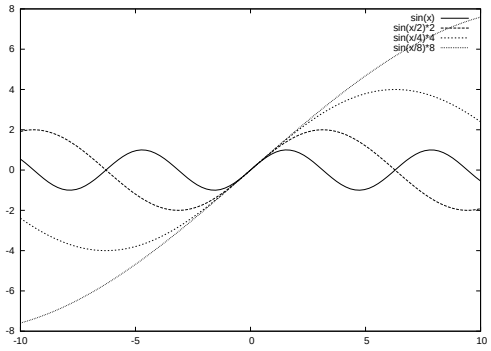
Tabelle 2: Beispiele ausgewählter Aktivierungsfunktionen

meinen Kriterien für die Wahl mehrerer Typen bekannt sind. Die Neuronenwahl stellt deshalb einen Metaparameter[18] eines neuronalen Netzes dar.
Das übliche Neuron besitzt als Propagierungsfunktion das leicht interpretierbare Skalarprodukt und als Aktivierungsfunktion eine Sigmoidfunktion oder den Tangenshyperbolikus. Die Wahl der Propagierungsfunktion wird dabei mit der Aggregation der eingehenden Signale und die Wahl der Aktivierungsfunktion mit der Approximationsfähigkeit der Kipp-Kübel-Sichtweise, welche die Reizweiterleitung eines biologischen Neurons beschreibt, begründet.
Neben den beiden genannten Aktivierungsfunktionen existieren aber auch noch weitere. Einige dieser Funktionen werden beispielhaft in Tabelle 2 angeführt und in Abbildung 2 veranschaulicht. Die Variable $z \in \mathbb{R}^+$ steht für einen Skalierungsfaktor. Wie in Tabelle 2 dargestellt, sind Aktivierungsfunktionen häufig monoton steigende, nichtlineare und oftmals beschränkte Funktionen. Diese Eigenschaften sind im Rahmen Neuronaler Netzwerke bis auf die Nichtlinearität jedoch kaum erforscht. Lineare Aktivierungsfunktionen werden dabei ausgeschlossen, weil sonst die gesamte neuronale Funktion zu einer linearen degeneriert und dann u.a. ihre wesentliche Eigenschaft der Nichtlinearität verliert. Die Eigenschaft der Beschränkung hat dagegen eher den praktischen Hintergrund, dass die Zustände eines physikalischen Netzes ansonsten mit steigender Tiefe explodieren bzw. numerisch instabil werden können. Die Eigenschaft der Monotonie entspricht ferner einer erwünschten Simplizität bzw. der Forderung nach guten Interpolationseigenschaften, siehe Abschnitt 2.3.4.
Für eine monotone Neuronenfunktion besteht zudem ein Zielkonflikt zwischen der Funktionsbeschränkung durch einen Limes und der Vermeidung von Skaleneffekten. Für eine (nicht monotone) oszillierende Funktion existiert dieser Zielkonflikt hingegen nicht. Für Neuronale Netzwerke mit beschränkten Neuronenfunktionen ist außerdem eine endliche Abschätzung der internen Zustände und bei zusätzlich beschränkter Eingabe sogar die Konstruktion eines Netzwerkes mit vorgegebenem maximalem Skaleneffekt möglich.

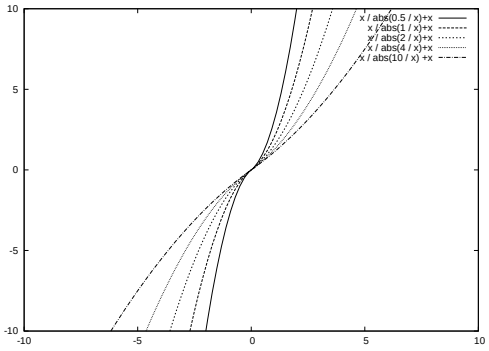
[18] Ein Metaparameter ist ein (latenter) Parameter, der einem Modell vorausgesetzt wird. Metaparameter werden entweder nicht als solche (an-) erkannt oder sind noch nicht ausreichend untersucht.



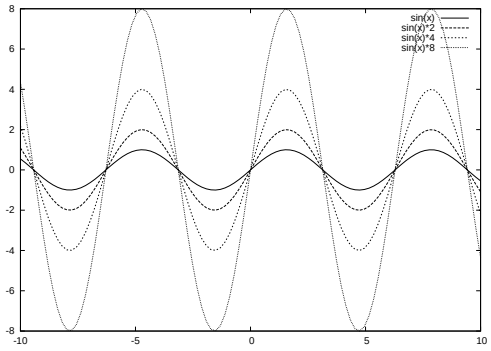
Skalierte AbsA-Funktionen



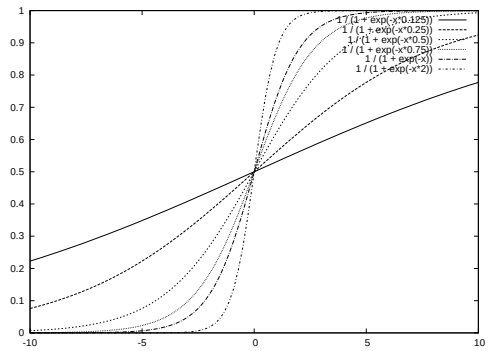
Skalierte SinZZ-Funktionen



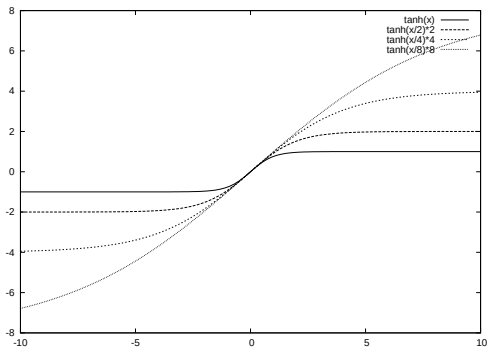
Skalierte AbsB-Funktionen



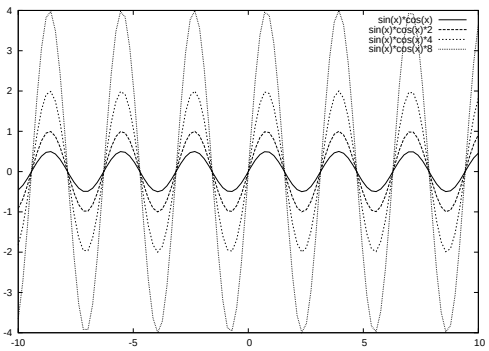
Skalierte SinZ-Funktionen



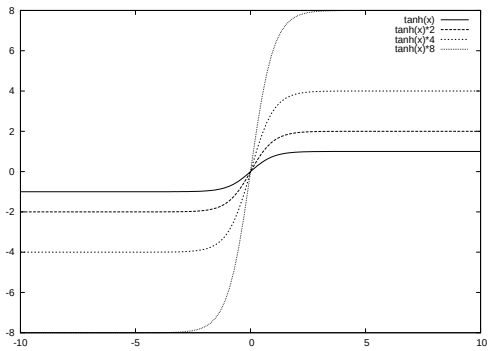
Skalierte Sigmoid-Funktionen



Skalierte TanhZZ-Funktionen



Skalierte SinCos-Funktionen



Skalierte TanhZ-Funktionen

Abbildung 2: Funktionsplots ausgewählter Aktivierungsfunktionen

3 Datenstrukturen

Die Implementierung eines Neuronalen Netzwerkes ist auf vielfältige Weise realisierbar, daher ist eine sorgfältige Konzeption der Datenstrukturen unerläßlich. Der hier gegebene stichpunktartige Überblick über ein klassisches Pflichtenheft für eine neuronale Datenstruktur umfasst typischerweise nicht alle Punkte, die Berücksichtigung finden sollten, sondern nur die aus der Benutzerperspektive sichtbaren Eigenschaften, weil dort Effizienz- und Verwendbarkeitsaspekten oft nicht ausreichend Aufmerksamkeit gewidmet wird[19].
Die folgenden Ausführungen beziehen sich auf die Referenzimplementierung in Abschnitt 3.1. Zusätzlich werden zwei Alternativimplementierungen vorgeschlagen, um effizientere Varianten aufzuzeigen und die mögliche Vielfalt zu demonstrieren. Angesichts der intensiven Nutzung dieser Datenstruktur in diversen Algorithmen ist die Geschwindigkeit ihrer Operatoren allerdings als kritisch zu betrachten. Die Referenzimplementierung des LinearGenomes wird aus diesem Grund in der Programmierpsprache C/C++ vorgenommen. Die Implementierung der Alternativen in der Programmiersprache Java hat dieser Wahl gegenüber entwicklungs- und wartungstechnische Vorteile, ist aber erfahrungsgemäß bei rechenintensiven Algorithmen etwas langsamer.
Anstatt ein vollständiges Pflichtenheft für die Entwicklung einer neuronalen Datenstruktur in Anlehnung an Balzert (1996, 1998) zu geben, wird lediglich ein pragmatischer Überlick über die wesentlichen Sollanforderungen vorgenommen. Das Lastenheft wird an dieser Stelle ohne Verlust von Anschauung vernachlässigt.
Die Besonderheit der geforderten Datenstruktur ist dabei ihre Flexibilität, weil sie dynamisch veränderbare Neuronale Netzwerke aufnehmen können muss. Dies erfordert komplexere Datenstrukturen als sie für Netzwerke mit statischer Topologie notwendig sind, welche z.B. für Implementierungen des Backprop-Algorithmus oder Realisierungen in FPGAs eingesetzt und zumeist als Adjazenzmatrix kodiert werden.
Der kurze Überblick der Anforderungen besteht aus der folgenden pragmatischen Auswahl von Punkten eines (imaginären) Pflichtenhefts:

1. Die Entwicklung soll das KISS-Prinzip (keep it short and simple) berücksichtigen.

2. Die Datenstruktur muss die in Tabelle 3 beschriebenen Basisoperatoren bereit stellen, um genügend Flexibilität zu gewährleisten. Höhere Operatoren sollen dann aus den Basisoperatoren kombiniert werden, wie z.B. AddSubNetwork, um ein neues Teilnetz in ein bestehendes Netzwerk einzufügen.

3. Ein existentes Netz soll vollständig speicher- und wiederherstellbar sein, so dass Ergebnisse permanent zugänglich gemacht werden können.

[19] Zum Beispiel wird ein einfaches Feld kaum der Mehrfachbaumstruktur eines Neuronales Netzwerkes gerecht. Insbesondere in Hinsicht auf das Laufzeitverhalten sollten mehr Informationen als nur dieses einfache Feld bereit gehalten werden.

Operator	Beschreibung
(Konstruktor)	Konstruktion eines NN mit fester Eingabe- und Ausgabedimension
AddNode	Hinzufügen eines Neurons
AddLink	Hinzufügen einer Verbindung
RemoveNode	Entfernen eines Neurons
RemoveLink	Entfernen einer Verbindung
FindNode	Suche nach einem Neuron
FindLink	Suche nach einer Verbindung
SetWeights	Setzen der Gewicht des NN
GetWeights	Abfrage der Gewichte des NN
GetNumberOfWeights	Abfrage der Anzahl der Gewichte
GetNumberOfInputs	Abfrage der Anzahl der Eingabekomponenten
GetNumberOfOutputs	Abfrage der Anzahl der Ausgabekomponenten
SetInput	Setzen des Eingabe des NN
GetOutput	Abfrage der Ausgabe des NN
Evaluate	Abfrage der Ausgabe des NN als Funktion seiner Gewichte

Tabelle 3: Beispielhafte Basisoperatoren einer Datenstruktur für eine flexible Implementierung eines Neuronalen Netzwerkes

4. Die Datenstruktur soll immun gegen Veränderungen sein, welche die Struktur zerstören. Entsprechende Absicherungen müssen in die Mutatoren eingebaut werden.

5. Beim Umgang mit einem Neuronalen Netzwerk muss stets dieselbe Indizierung der Eingabeneuronen und der Einträge des Eingabevektors vorliegen. Gleiches gilt für die Indizierung der Ausgabeneuronen.

6. (Optional) Die Komponenten des Neuronalen Netzwerkes sollen mit BOT-Logik ausgestattet sein, damit diese weitgehend eigenständig z.B. als Prozess auf verteilten Systemen oder als Thread arbeiten können.

Die Quellcodes der im Folgenden aufgeführten Implementierungen befinden sich auf dem beiliegenden Datenmedium (Seite 150).

3.1 Referenzimplementierung: LinearGenome

Die an dieser Stelle vorgestellte Datenstruktur LinearGenome (Revision 326) wird in Kassahun u. Sommer (2005) entworfen und bildet die Grundlage des EANT-Algorithmus (siehe Abschnitt 6.3). Sie ist in der Programmiersprache C++ implementiert und basiert in entfernter Analogie zum biologischen Genom auf einem Vektor (std::Vector). Dieser nimmt Einträge vom Typ des abstrakten Objektes Node (ein einzelnes Gen) auf, welches ein Container mit einer Doppelbedeutung ist

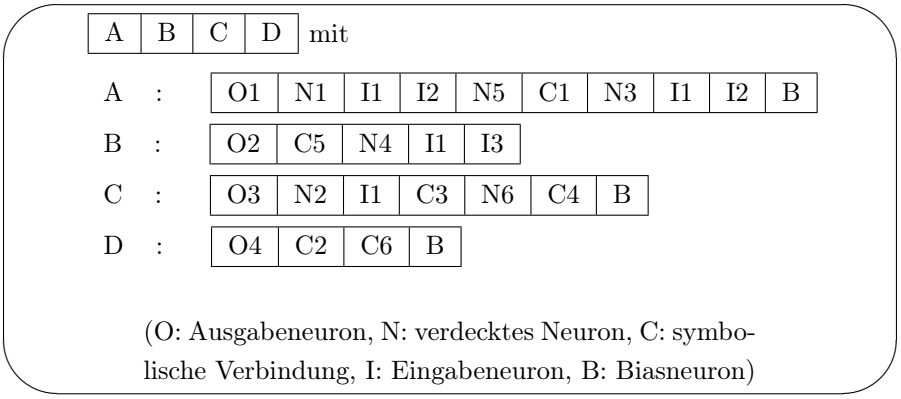


Abbildung 3: Vereinfachtes Beispiel der Datenstruktur LinearGenome

und eine Verbindung sowie deren abstrakte Zielkomponente, enthält. Zudem ist jeder Node mit einer eindeutigen Identifikationsnummer (UID) ausgestattet, um ihn eindeutig ansprechen zu können.
Mögliche Komponenten erweitern den abstrakten Node, um ihre minimale Funktionalität, d.h. sie besitzen keine Referenz zu anderen Komponenten. Die verfügbaren Komponenten sind das Biasneuron (Bias), Neuronen (Neuron), symbolische Verbindungen (Connection) und rekurrente Verbindungen (Recurrent), wobei letztere derzeit nicht verwendet werden. Im Umgang mit den Genen des Genoms ist daher immer eine Fallunterscheidung vorzunehmen und zu berücksichtigen, dass der Zugriff auf andere Komponenten nur über die Datenstruktur selbst durchführbar ist.
Der Aufbau der Datenstruktur erfolgt ausgehend von den Ausgabeneuronen rekursiv. Dazu besitzt ein Neuron an Stelle i im Vektor die Information über die Anzahl m_i seiner eingehenden Verbindungen und kann diese als Komponenten $[i+1, i+m_i-1]$ des Vektors ansprechen. Abbildung 3 zeigt diese Datenstruktur beispielhaft für das NN aus Abbildung 1. Tabelle 4 enthält die für das lineare Genom gültigen Operator-Laufzeiten in Abhängigkeit von dessen Größe, die als die Anzahl der enthaltenen Verbindugnen (Kanten des repäsentierten NN $|V|$) definiert ist. Dabei wird der zugrunde liegende Vektor als Feld angesehen, und es werden Schutzmechanismen berücksichtigt, die sicherstellen, dass die Repräsentation des NN nicht durch Veränderungen seiner Darstellung im Vektor korrumpiert wird.
Es besteht ferner die Möglichkeit, Instanzen der Datenstruktur inklusive aller transienten internen Zustände XML-basiert zu speichern und zu laden.

Evolution der Referenzimplementierung

Die Arbeit an der Datenstruktur LinearGenome (und damit einhergehend an EANT) im Zuge dieser Arbeit besteht aus Verbesserungen, Ergänzungen und Fehlerkorrekturen an der Datenstruktur, die hier kurz zusammengefasst werden:

- Signifikante Laufzeitverbesserungen werden in verschiedenen Teilen der Funktion LinearGenome.GetResponseOfSubNetwork realisiert. Für ein Neuron wer-

Operator	**Laufzeit**	**Bemerkungen**
AddNode	$O(1)$	Vektor basiert auf doppel-verlinkter Liste
AddLink	$O(\|V\|)$	Sicherheitsüberprüfung: Zykelerkennung
RemoveNode	$O(\|V\|)$	Sicherheitsüberprüfung: Suche nach bzw. Austausch mit Verbindungen
RemoveLink	$O(1)$	-
FindNode	$O(\|V\|)$	-
FindLink	$O(\|V\|)$	-
SetWeights	$O(\|V\|)$	-
GetWeights	$O(\|V\|)$	-
GetNumberOfWeights	$O(1)$	Rückgabe der Größe des Node-Vektors eingehender Verbindungen
GetNumberOfInputs	$O(1)$	-
GetNumberOfOutputs	$O(1)$	-
SetInput	$O(1)$	-
GetOutput	$O(\|V\|)$	Suche nach den Ausgabeneuronen
Evaluate	$O(\|V\|)$	Separate Sortierung nach jeder Strukturänderung

Tabelle 4: Operator-Laufzeiten der Datenstruktur LinearGenome

den die Teilnetzwerke der eingehenden Verbindungen nun nicht mehr wiederholt kopiert und stets erneut rekursiv ausgewertet. Durch die Einführung eines eindeutigen Identifikators für jede Evalution einer Netzeingabe können nun wiederholte Berechnungen vermieden werden. Das Entfernen der unnötigen Kopieraktionen verbessert zusätzlich die Laufzeit, siehe dazu auch Abschnitt 3.4.

- Die Erkennung von Zykeln beim Einfügen neuer Verbindungen vermeidet nun eine unendliche Rekursion in einer Auswertungsroutine des LinearGenome, die sonst bei der Verwendung in EANT zu erheblichen Problemen führt. Ein anderer Ansatz, der Zykel erlaubt, befindet sich in Abschnitt 3.3.

- Eine Vergleichbarkeit zweier Datensturkturen des Typs LinearGenome, die einige Methoden in EANT erfordern, wird durch die Sortierung der Node Container auf Basis des BubbleSort-Algorithmus in $O(n^2)$ direkt auf der Datenstruktur realisiert. Beginnend an der Ausgabeschicht erfolgt dazu rekursiv für jedes Teilnetz eine Sortierung nach Containertyp und UID (jeweils für Neuronen, Verbindungen, rekurrenten Verbindungen und Biasverbindungen), die nun einen komponentenweisen Vergleich erlaubt.

- Die Korrektur eines Indexfehlers infolge der Fehlbenennung einer Argumentvariablen in der Funktion LinearGenome.RemoveNodes ermöglicht nun das korrekte Austauschen eines zu löschendes Teilnetzes mit der ersten auf dieses verweisenden Verbindung. Damit wird ein Segmentierungsfehler bei der Programmausführung als Absturzursache von EANT in Testläufen behoben.

Operator	Laufzeit	Bemerkungen
AddNode	$O(1)$	-
AddLink	$O(\|E\|)$	Sicherheitsüberprüfung: Zykelerkennung
RemoveNode	$O(1)$	-
RemoveLink	$O(1)$	-
FindNode	$O(\|V\|)$	-
FindLink	$O(\|E\|)$	-
SetWeights	$O(\|E\|)$	-
GetWeights	$O(\|E\|)$	-
GetNumberOfWeights	$O(1)$	-
GetNumberOfInputs	$O(1)$	-
GetNumberOfOutputs	$O(1)$	-
SetInput	$O(1)$	-
GetOutput	$O(1)$	-
Evaluate	$O(\|E\| + \|V\|) = O(\|V\|)$	Separate Sortierung nach jeder Strukturänderung

Tabelle 5: Operator-Laufzeiten der Alternativimplementierung

3.2 Alternativimplementierung: Kanonisches Netzwerk

Alternativ zur Referenzimplementierung kann ein Neuronales Netzwerk auch ohne (vermutete) Ähnlichkeit zu einem biologischen Genom implementiert werden.
Mit dem Mehrfachbaum als Grundlage bietet sich hierfür eine Adjazenzmatrix oder eine Adjazenzliste an. Die inflexible Adjazenzmatrix ist jedoch für eine flexible Repräsentation eines Neuronalen Netzes ungeeignet, weil für jedes Hinzufügen oder Entfernen eines Neurons, die Matrix im Speicher mit Aufwand $O(n^2)$ neu aufgebaut werden muss. Tabelle 5 zeigt die Operator-Laufzeiten für eine auf Adjazenzlisten basierende neuronale Datenstruktur. Die Implementierung benutzt separate Neuronenlisten für Eingabe-, verdeckte und Ausgabeschicht sowie eine weitere Liste für Verbindungen. Außerdem wird die aktuelle Netzstruktur für Vorwärts- und Rückwärtspropagierung linearisiert, wozu ein Feld mit Referenzen zu den Neuronen in DFS-Reihenfolge von Eingabe- oder Ausgabeschicht ausgehend angelegt wird.
Es zeigt sich, dass die Operator-Laufzeiten aufgrund der zahlreichen Listen etwas schneller als die der Referenzimplementierung implementierbar sind. Ein weiterer Vorteil dieser Implementierung ist ihre simple Handhabbarkeit, da sie direkt die Anschauung eines Netzes als Mehrfachbaum bzw. Graph umsetzt.

3.3 Alternativimplementierung: Chaotisches Netzwerk

Obige Implementierungen können ohne großen Aufwand so modifiziert werden, dass sie Zykel im Netzwerk zulassen, ohne bei der Auswertung in Endlosschleifen zu geraten. Damit kann, sofern eine Relation zu biologischen neuronalen Netzen aufrecht erhalten werden soll, ein vermutlich etwas realitätsnäheres Netz konstruiert wer-

den. Hierzu wird eine Konstante $b \in \mathbb{N}$ global festgelegt und jedes Neuron b-mal ausgewertet. Ist b größer als die Anzahl aller Neuronen, dann erhält jedes Neuron am Ende dieser Einpendelphase (Burn-In) gültige Eingabesignale. Auf diese Weise kann bei genügend großer Einbrennphase die Wirkung von Zykeln im Netzwerk beobachtet und benutzt werden.
Die Operatorlaufzeit von AddLink sinkt dabei aufgrund der wegfallenden Zykelerkennung auf $O(1)$, während die Laufzeit der Funktion Evaluate mit der Einpendelphase auf $O(b \cdot (|V| + |E|))$ anwächst.

3.4 Auswertungsstrategien

Eine Auswertungsstrategie ist zumeist Teil der Datenstruktur eines Neuronalen Netzwerkes und oftmals an Besonderheiten der Darstellung gekoppelt. Eine konsequente Trennung von Repräsentation und Anwendung hingegen offenbart sofort eine Vielfalt möglicher Auswertungsmethoden, wobei deren Effizienz natürlich abhängig von der Wahl der Darstellung ist. Es zeigt sich, dass z.B. die Vorbereitung (Erstellen einer Neuronenliste mittels DFS für jedes Ausgabeneuron und Entfernen von Redundanz) der kanonischen Implementierung für die letzten beiden Auswertungsstrategien weniger aufwendig als die Implementierung der Datenstruktur LinearGenome ist.
Quantitativ betrachtet ist der wesentliche Aspekt aber nicht der Zeitaufwand infolge der Komplexität einer Vorbereitung, sondern die Laufzeit einer Auswertung. Die folgenden Strategien stellen fortschreitend effizientere Vorgehensweisen dar.

Die einfachste Strategie benutzt eine Rekursion (keine Iteration) und sucht beginnend bei jedem Ausgabeneuron nach Eingabesignalen von Neuronen. Wird ein solches gefunden, so wird die Auswertung mit diesem als Argument (rekursiv) aufgerufen. Dies geschieht solange, bis die Eingabeschicht erreicht ist, welche keine Eingabesignale hat. Im Verlauf dieser Strategie kann es leicht zu Mehrfachauswertungen von Neuronen kommen, weswegen sich folgende Modifikation anbietet.

Die zweite Strategie benutzt ebenfalls eine Rekursion (keine Iteration) und inkrementiert als erstes den globalen Evalutionsidentifikator, bevor sie für jedes Ausgabeneuron nach Eingangssignalen von Neuronen mit nicht aktuellem (lokalem) Evalutionsidentifikator sucht. Wird ein solches Neuron gefunden, wird die Auswertung mit diesem als Argument (rekursiv) aufgerufen. Damit ist gewährleistet, dass keine Neuronen mehrfach ausgewertet werden. Es gibt jedoch eine noch effizientere Strategie.

Die dritte Strategie benutzt eine geordnete (linearisierte) Form eines Neuronalen Netzwerkes, welche für jedes Netz generiert werden kann. Dazu wird das Netzwerk wie in der zweiten Strategie dargestellt durchlaufen und ein Stack nimmt die Reihenfolge der auszuwertenden Neuronen auf. In umgekehrter Reihenfolge kann das

neuronale Netz nun so lange ohne weitere Rekursionen ausgewertet werden, bis es strukturell mutiert wird. Erst dann muss die lineare Reihenfolge erneut ermittelt werden. Diese Strategie trägt bei vielen Auswertungen nochmals erheblich zu einer höheren Gesamtgeschwindigkeit bei.

Die vierte Strategie ist bis auf die Kodierung des neuronalen Netzes in Assembler[20] mit der dritten Strategie identisch und erhöht die Geschwindigkeit erneut, sofern der verwendete Kompilierer nicht selbst eine derartige Optimierung vornimmt oder diese wegoptimiert. Analog können weitere Teile des Algorithmus kodiert und beschleunigt werden.

[20]Diese Form wird im EANT-Algorithmus als BinaryGenome genutzt.

4 Synopsis

Wie in Abschnitt 2.3 bemerkt, sind Neuronale Netzwerke mathematische Funktionen, deren Training als Funktionsapproximation unter der Nebenbedingung einer Funktionsinterpolation verstanden werden kann. Dieser Interpretation (und der Aufgabenstellung in Abschnitt 1.3) folgend, soll im Kontext der evolutionären Entwicklung Neuronaler Netzwerke ein Netz (Modell[21]) gefunden werden, dass eine gegebene Lernaufgabe optimal approximiert oder sogar löst und dabei als Nebenbedingung im Hinblick auf Interpolationseigenschaften glatte (Basis-) Funktionen (Neuronen) einsetzt sowie eine minimale Netzstruktur aufweist. Das resultierende neuronale Netz kann dann als angelernter funktionaler Informationsspeicher interpretiert werden, der optimalerweise Lösungen für alle Probleme der durch die Daten repräsentierten Problemklasse enthält.

Anstelle der minimalen Netzstruktur sind allerdings auch andere oder zusätzliche Nebenbedingungen denkbar. Für die Interpolationseigenschaft eines Neuronalen Netzwerkes bietet sich z.B. an, die schlecht interpretierbare Nebenbedingung der Netzgröße durch ein besser interpretierbares Gütekriterium wie die Generalisierung zu ersetzen.

Die folgenden Abschnitte bilden entsprechend und gemäß der Aufgabenstellung eine Übersicht der zur Entwicklung einer neuronalen Topologie relevanten Gegenstände.

4.1 Suchraum und Komplexität

Der Suchraum für ein Neuronales Netzwerk wird durch dessen Parameter aufgespannt. Für ein typisches Neuronales Netzwerk umfasst dieser die Anzahl der Neuronen in den drei Schichten, die Wahl der Gewichte und die Auswahl von Propagierungsfunktion sowie Aktivierungsfunktion. Ein Blick auf die dabei auftretenden Dimensionen zeigt die Komplexität. Siehe dazu auch Fluch der Dimensionaliltät in Bishop (2004) oder Duda u. a. (2004).

Zur Veranschaulichung empfiehlt sich die Adjazenzmatrix[22] des (V, E)-Mehrfachbaumes des Netzes. Für $n \in \mathbb{N}$ Eingabeneuronen, $d \in \mathbb{N}$ verdeckte Neuronen und $m \in \mathbb{N}$ Ausgabeneuronen hat diese quadratische[23] Matrix $|V| = n + d + m$ Spalten bzw. Zeilen und $|V|^2$ Einträge, wobei die Anzahl der Matrixeinträge der maximalen Anzahl an Kanten entspricht. Für das Training des Netzwerkes sind somit bis zu $|V|^2$ Gewichte zu optimieren. Soll entsprechend der Aufgabenstellung auch die Netzstruktur entwickelt werden, so kann für eine fixierte Neuronenanzahl mit Hilfe

[21] Die Suche nach einem Neuronalen Netzwerk ist als allgemeine Modellsuche für einen Datzensatz bzw. für eine gegebene Aufgabe aufzufassen.

[22] In dieser Nachbarschaftsmatrix steht ein Eintrag für Gewicht und Existenz einer Kante. Eine Kante wird dabei normalerweise als existent betrachtet, wenn ihr Matrixeintrag ungleich 0 ist.

[23] Die Kanten des Mehrfachbaums sind gerichtet, weswegen die Matrix nicht symmetrisch ist.

Neuronenanzahl k	0	1	2	3	4	5	6	7	8	9	10	11	...
Kantenkombinationen	0	0	2	6	14	30	62	126	254	510	1022	2046	...
2^k	0	2	4	8	16	32	64	128	256	512	1024	2048	...

Tabelle 6: Abschätzung möglicher Kantenkombinationen

der Landausymbole die obere Schranke[24]

$$\sum_{i=1}^{|V|^2} \binom{|V|^2}{i} \in \Theta(2^{|V|} - 2) \tag{2}$$

der zu testenden Kombinationen von Kanten (ohne Zurücklegen) angegeben werden, wie Tabelle 6 veranschaulicht. Gleichung (2) zeigt, dass bereits die Aufzählung aller Netzstrukturen in Abhängigkeit der Neuronenanzahl exponentiell viel Zeit beansprucht. Die Laufzeit erhöht sich weiter, wenn zusätzlich die $|E|$ Kantengewichte zu optimieren sind, was minimal in $O(|E|)$ realisierbar ist. Die Suche nach einer optimalen neuronalen Lösung für ein Problem liegt somit insgesamt in $\mathbb{EXP}$ und kann praktisch in annehmbarer Zeit nur approximiert werden.
Dennoch ist dazu kein FPTAS (Fully Polynomial Time Approximation Scheme), mit dem eine bestimmte Approximationsgüte in polynomieller Laufzeit in $\mathbb{P}$ parameterisierbar wäre, oder polynomieller Algorithmus bekannt. Die üblichen oder praktikablen Approximationen für ein Suchschema liegen demnach bislang in $\mathbb{NP}$.
Es wird angemerkt, dass allgemein und insbesondere beim Umgang mit diesen Techniken separable Probleme stets separiert werden sollten, um Teilprobleme mit geringerer Problemdimension und Komplexität zu erhalten, die meistens besser lösbar sind. Wie die Abschätzung in Gleichung 2 ferner verdeutlicht, ist es dabei angebracht, die Sicht auf ein neuronales Netz bis mindestens auf die Ebene von Neuronen, Kanten und Gewichten zu differenzieren, um Teilaspekte untersuchen und abschätzen zu können.

4.2 Online- und Offlineverfahren

Ein wesentliches Unterscheidungsmerkmal bei Algorithmen, die mit Daten arbeiten, ist ihre Fähigkeit, bei Vorlage neuer Daten ihren Zustand aktualisieren zu können oder neu erlernen zu müssen. Erstere werden Online-, letztere Offlineverfahren genannt. Der prinzipielle Unterschied läßt sich an der einfachen adaptiven bzw. autoregressiven (Online-) Gleichung

$$x_{t+1} = \alpha \cdot x_t + \beta_t \cdot y_t \tag{3}$$

[24] Obwohl die Funktion exakt abgeschätzt werden kann, müssen nicht alle Kanten zulässig sein. Dies gilt beispielsweise für Kanten zu Eingabeneuronen.

mit $x_t, y_t \in \mathbb{R}$ f.a. $t \in \mathbb{N}_0$ und $\alpha, \beta \in]0,1[$ veranschaulichen, denn sie ist äquivalent der (Offline-) Gleichung

$$x_{t+1} = \alpha^{t+1} \cdot x_0 + \sum_{i=1}^{t+1} \beta^i \cdot y_{t-i} \,, \tag{4}$$

in welcher die Koeffizienten β_i eine exponentielle Glättung der y-Reihe erzeugen.
Während sich Onlineverfahren wie Gleichung (3) für jedes Datum t eines Datensatzes mit einem Iterationsschritt Δx_t aktualisieren, können Offlineverfahren dies nur für einen vollständigen Datensatz (Batchmode).
Die Gleichungen zeigen ferner, dass manchmal sogar aus einem Offline- ein Onlineverfahren konstuiert werden kann, indem eine (adaptive) Iterations- oder Rekursionsvorschrift[25] für das Verfahren gefunden wird.
Sofern beide Varianten existieren, hängt die Wahl der Variante dann vom Anwendungsgebiet ab, denn Onlinalgorithmen sind zwar weniger rechenaufwendig, aber aufgrund endlicher Adaptionsgenauigkeit oft nicht so präzise wie ihre Offlinevarianten[26]. Ein endliches Adaptionsverhalten ist jedoch z.B. bei einer Positionsverfolgung durchaus erwünscht, wohingegen es bei stochastischen Algorithmen umgekehrt u.U. wünschenswert ist, nicht mehrfach das Gleiche zu tun, sondern den nächsten Schritt mit einer Datenbank (offline) zu verifizieren.

4.3 Formalisierung

Zur Handhabung eines Neuronalen Netzwerkes sind bedingte Funktionen ein praktisches Mittel, um Metaparameter zu formalisieren oder bestimmte Parameter einer Funktion zu fixieren. Seien $m, n \in \mathbb{N}$ Dimensionen, sei $x \in \mathbb{R}^m$ ein Vektor, sei $k \in \mathbb{N}_{\leq m}$ ein Index und sei $h : \mathbb{R}^m \to \mathbb{R}^n$ eine Funktion. O.B.d.A. seien ferner die Vektoren $\theta_0 := x_1, \cdots, x_{k-1}, x_m \in \mathbb{R}^k$ und $\theta_1 := x_k, \ldots, x_{m-1} \in \mathbb{R}^{m-k}$ definiert.
Werden nun die Parameter in θ_1 für h fixiert, so heißt h die auf θ_1 bedingte Funktion $h : \mathbb{R}^k \times \mathbb{R}^{m-k} \to \mathbb{R}^n$ und formal wird $h(\theta_0; \theta_1)$ notiert. Als Komponenten von θ_0 und θ_1 sind dabei alle (Meta-) Parameter von h ohne Zurücklegen kombinierbar.
Bedingte Funktionen werden im Rahmen Neuronaler Netzwerke für die Modellierung einer Blackbox (siehe Abbildung 4) benutzt, um zusammen mit einer Bewertungsfunktion (siehe Abschnitt 4.6) der Ausgabe Verstärkendes Lernen umzusetzen. Die Sicht auf ein Netzwerk kann damit bespielsweise auf die neuronale Funktion $h(w; x)$ der Netzgewichte w bedingt auf die aktuelle Netzeingabe x begrenzt werden.

[25] Eine Iteration ist eine verschacheltungsfreie Rekursion, die somit ohne Verwendung eines Kellers (Stacks) auskommt.

[26] Beispielsweise verhält sich die Adaption der Abstiegsrichtung im Gauss-Newton-Verfahren bei der Minimierung einer Residuenquadratsumme analog zum Training eines Adalines (Widrow u. Hoff (1960), Perzeptron mit Rückkopplung bzw. rekurrenten Verbindungen) mittels μ-LMS-Algorithmus (online), bei dem die Konvergenz für große Schrittweiten μ nicht sichergestellt ist. Im Gegensatz hierzu konvergiert die lineare Regression (offline) auf Basis der Cholesky-Zerlegung oder des CGNE-Algorithmus immer zu einer Lösung.

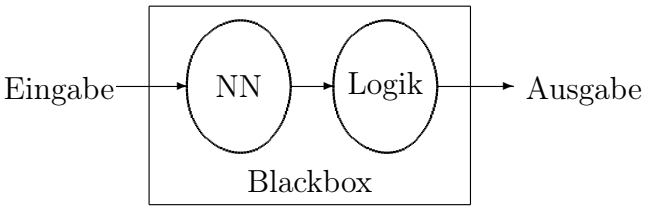


Abbildung 4: Blackbox und Blackboxlogik

Ist dabei nicht das Neuronale Netzwerk allein Gegenstand der Betrachtung, sondern vielmehr eine andere Funktion bzw. Logik, welche die Ausgabe des Netzes als Eingabe erhält, wird von einer Hintereinanderausführung oder praktisch von einer Steuerung gesprochen. Dies trifft zum Beispiel auf eine neuronale Robotersteuerung zu (siehe Kapitel 7). Eine solche Verschachtelung kann analog zu Abbildung 4 ebenfalls als Blackbox dargestellt werden, da von außen nur Ein- und Ausgabe erkennbar sind.
Ferner kann Überwachtes Lernen direkt in Verstärkendes Lernen überführt werden, wenn als Logik bzw. Bewertungsfunktion ein Fehlerfunktional zum Einsatz kommt. Dazu wird z.B. eine geeignete Norm der Differenz von Soll- und Istausgabe über den Trainingsdatensatz aufsummiert und als Fehlerwert zurückgegeben[27].

4.4 Lokale und globale Optimierung

Die Begriffe „lokale Optimierung“ und „globale Optimierung“ werden häufig gegeneinander abgegrenzt, obwohl für letzteren oftmals keine Definition gegeben wird. Im Weiteren werden deshalb folgende Definitionen verwendet[28]:

- Das Attribut „lokal“ bedeutet, dass ein Minimierungsverfahren nicht aus einem lokalen Minimum heraus kommen kann.
- Das Attribut „global im Kontext deterministischer Verfahren“ bedeutet, dass ein Minimierungsverfahren aus einigen lokalen Minima heraus kommen kann[29].
- Das Attribut „global im Kontext stochastischer Verfahren“ bedeutet, dass ein Minimierungsverfahren jeden Punkt eines diskreten Suchraumes mit einer Wahrscheinlichkeit > 0 oder abzählbar unendlich viele Punkte eines stetigen Suchraumes erreichen kann.

Die Definition der Globalität für stochastische Verfahren soll insbesondere der Balancierungs-Gleichung (6) (Detailed-Balance-Equation) in Abschnitt 4.4.2 Rechnung tragen, die bei der Suche nach einer optimalen (diskreten) Struktur eines

[27] Siehe dazu auch Kapitel 6, insbesondere Fußnote 63 (Seite 49).
[28] Während dabei konstruktiv mit endlichen und abzählbar unendlichen Menge umgegangen werden kann, gilt dies bislang nicht für überabzählbar unendliche Mengen. Für letzte ist lediglich eine statistische Handhabung bekannt.
[29] Dies trifft z.B. für Verfahren auf Basis der Hessematrix oder den Downhill-Simplex-Algorithmus zu.

neuronalen Netzes beachtet werden sollte. Diese drückt die Forderung aus, dass wenn schon nicht überabzählbar viele Punkte erreicht werden können, trotzdem möglichst viele ausgewogen verteilte erreichbar sein sollten.

4.4.1 Markovketten

Sei S ein abzählbar unendlicher Zustandsraum und $n \in \mathbb{N}$. Dann heißt eine (Zeit-) Reihe von Zufallszahlen $(X_n)_{n\in\mathbb{N}}$ über S Markovkette, wenn für alle (Zeit-) Punkte $t \in \mathbb{N}_{<n}$ und alle Zustände $i \in S$ die Bedingung

$$P\left(X_{t+1} = i_{t+1} | X_t = i_t, X_{t-1} = i_{t-1}, \ldots, X_0 = i_0\right) = P\left(X_{t+1} = i_{t+1} | X_t = i_t\right)$$

erfüllt ist. Die Wahrscheinlichkeit einen Zustand im Zeitpunkt $t+1$ zu erreichen, ist also nur vom Zustand im Zeitpunktes t abhängig.
Ist diese Wahrscheinlichkeit ferner nicht vom Zeitpunkt t abhängig, so heißt die Markovkette homogen, und die Transitionsfunktion für $t \in \mathbb{N}_{<n}$ Schritte kann als Transitionsmatrix, mit den Übergangswahrscheinlichkeiten

$$P_t\left(i, j\right) := P\left(X_t = j | X_0 = i\right)$$

als Einträgen, angegeben werden.
Eine homogene Markovkette heißt irreduzibel[30], wenn immer alle Zustände mit einer Wahrscheinlichkeit größer Null erreichbar sind:

$$\forall i, j \in S\, \exists m \in \mathbb{N} :\ P_m\left(i, j\right) = P\left(X_m = j | X_0 = i\right) > 0$$

Zum Ausschluß eines deterministischen Verhaltens der Kette wird weiter die Periode eines Zustandes als der größte gemeinsame Teiler der Schrittzahl

$$z_i := \left\{t \geq 0 :\ P_t\left(i, i\right) > 0\right\}$$

definiert, mit der ein Zustand wiederholt erreichbar ist, und eine Markovkette heißt genau dann aperiodisch, wenn $\forall i \in S :\ z_i = 1$ gilt.
Während im Fall endlicher Markovketten die Eigenschaften irreduzibel und aperiodisch ausreichen, um die Existenz einer Grenzverteilung für alle Zustände (Gleichung 5) sicherzustellen, muss für abzählbar unendliche Ketten ein positiver Limes sichergestellt werden.
Dazu wird eine irredizibele Markovkette genau dann rekurrent genannt, wenn es einen Zustand[31] gibt, der bei unendlich vielen Schritten (Zufallsexperimenten) unendlich oft erreicht (gezogen) wird:

$$\exists i \in S :\ \sum_{t=0}^{\infty} P_t(i, i) = \infty\,,$$

[30] Die Differenzierung verschiedener Zustandsklassen in reduzierbaren Markovketten erfolgt o.B.d.A. analog.
[31] Gilt diese Aussage, so gilt sie sogar für alle Zustände.

andernfalls heißt sie transient. Ferner muss zwischen positiv-rekurrenten

$$\forall i, j \in S : \lim_{t \to \infty} P_t(i, j) > 0 \tag{5}$$

und null-rekurrenten ($\forall i, j \in S : \lim_{t \to \infty} P_t(i, j) = 0$) Ketten unterschieden werden. Ist eine abzählbar unendliche Markovkette aperiodisch und positiv-rekurrent, dann existiert eine (eindeutige, stationäre) Grenzverteilung π auf S. Es gilt dann außerdem

$$\forall j \in S : \sum_{i \in S} \pi(i) P_1(i, j) = \pi(j) \ ,$$

und die Markovkette heißt ergodisch[32], vergleiche Gilks (1995).

4.4.2 Detailed-Balance-Equation

Im Kontext Neuronaler Netzwerke kann (mit der Notation aus Abschnitt 4.4.1) die s.g. Detailed-Balance-Equation

$$\forall i, j \in S : \pi(i) P_1(i, j) = \pi(j) P_1(j, i) \tag{6}$$

benutzt werden, um die Ausgewogenheit der Wahrscheinlichkeit des Hinzufügens und Entfernens einer Verbindung bzw. Kante im Graphen des Netzes darzustellen. Finden diese Aktionen zu einem Zustand des Netzes nicht ausgewogen statt, wird die Suche im Suchraum der Netzstruktur verzerrt und schöpft nicht die vorhandenen Möglichkeiten (Informationen) aus.
Wird die Suche hingegen als homogene, irreduzibele, aperiodische und positiv-rekurrent Markovkette konstruiert, garantiert Gleichung (6) die vollständige Bewegungsfreiheit im Suchraum. Als Vorlage einer derartigen Konstruktion bietet sich der bekannte Metropolis-Hastings-Algorithmus aus Metropolis u. a. (1953) und Hastings (1970) an.

4.5 Abbruchbedingungen

Lernalgorithmen sind fast immer iterativ, indem sie einen Lösungsvorschlag durch wiederholtes Durchlaufen eines Datensatzes so lange verbessern, bis dieser gut genug ist. Als Abbruchbedingung einer Iteration kommen dabei viele Entscheidungskriterien in Frage. Tabelle 7 listet die gebräuchlichsten auf. Stets wird in disjunktiver Verknüpfung der Bedingungen je ein Wert gegen ein Kriterium getestet, z.B. $err_i < stop$ oder $iter_i < maxiter$. Die Iteration endet, sobald eine Bedingung erfüllt ist.
Im Regelfall werden als Abbruchbedingungen die maximale Anzahl an Iterationsschritten sowie die maximale Toleranz für die Differenz zweier aufeinanderfolgender Funktionswerte vorgegeben. Sollen hingegen zwei iterative Verfahren verglichen

[32] Etwas Übergeordnetem (in diesem Fall der Grenzverteilung) folgend.

Kriterium \ Wert	Zahl	Differenz	Varianz (Volumen)
maxfevals	fevals_i	-	-
maxiter	iter_i	-	-
maxtime	$(\text{time}_i - \text{time}_0)$	-	-
tolx	x	$\|\|x_{i-t} - x_i\|\|$	$\text{Var}(x_{i-t+1,}\ldots, x_i)$
tolfun	$f(x_i)$	$\|\|f(x_{i-t}) - f(x_i)\|\|$	$\text{Var}(f(x_{i-t+1})\ldots, f(x_i))$
update	-	-	$\text{Var}(\{v_1, \ldots, v_t\})$
stop	err_i	$\|\text{err}_{i-t} - \text{err}_i\|$	$\text{Var}(\text{err}_{i-t+1}\ldots, \text{err}_i)$

Seien $i, t \in \mathbb{N}$ Iterationsindizes, $k, m, n \in \mathbb{N}$ Dimensionen, $f : \mathbb{R}^n \to \mathbb{R}^m$ eine Funktion, $x \in \mathbb{R}^n$ und $v \in \mathbb{R}^k$ Vektoren.

Tabelle 7: Typische Abbruchbedingungen

werden, beschränkt sich die Wahl zumeist auf die maximale Anzahl an Funktionsauswertungen oder die maximale Zeit.
Die Möglichkeit, eine maximale Toleranz für verfahrenspezifische Fehler zu setzen, wird durch die auf Verbesserungen der Lösung beschränkte Berechnung der Varianz vervollständigt und oft anstelle der Varianz für mehrdimensionale Daten das Volumen berechnet.
Genügen diese Abbruchbedingungen nicht, müssen dem Verfahren weitere Bedingungen eingebaut werden. Im CMA-ES-Algorithmus (Abschnitt 5.2) sind dies u.a. Bedingungen an die numerischen Eigenschaften der Kovarianzmatrix. In EANT (Abschnitt 6.3) trifft dies z.B. auf die Differenzenbildung (Stagnationserkennung) für Funktionswerte zu.

4.6 Gütekriterien

Die Betrachtung der Güte einer Lösung[33] eines Lernalgorithmus erfolgt normalerweise in Form von Güte-, Informations- bzw. Modellselektionskriterien[34] oder im evolutionären Kontext als Fitnesswert. Diese Kriterien ermöglichen den Vergleich unterschiedlicher Modelle auf Basis einer Kennzahl[35] und sind idealerweise allgemein und modellunabhängig definiert.
Für einen Vergleich neuronaler Netze existiert als Qualitätskriterium dabei bislang neben der Netzgröße nur die Performanz bzw. Fitness bzgl. einer Aufgabenstellung. Letztere ist dabei das übliche Mittel.
Ist eine erweiterte Qualitätsbetrachtung erwünscht, so kann die Generalisierung als Güte einer Lösung auf anderen als nur den Trainingsdaten untersucht werden. Die-

[33] Je nach Lernaufgabe ist dies einfach der Parametervektor eines Modells oder aber das Modell, z.B. in Form eines Neuronalen Netzwerkes, selbst. Eigentlich sind dabei jedoch Modell und Parameter als Einheit zu sehen, so dass ein veränderter Parameter stets ein neues Modell ergibt.

[34] Dies sind geeignete Funktionen, die einen interpretierbaren Skalar zurückliefern.

[35] Beispiel von Gütekriterien sind das AIC (Akaike Information Criterion) zur Selektion verschieden parameterisierter ARMA-Modelle, die Funktionswerte von Maximum-Likelihood-Schätzungen oder der Fitnesswert des Benchmarks in Kapitel 7.

se ist als Maß des Auswendiglernens (negativ) oder des Interpolationsvermögens (positiv) einer Lösung interpretierbar und wird meistens anhand einer Kreuzvalidierung (k-Cross-Validation) untersucht. Sie wird jedoch normalerweise nur für bewährte Verfahren betrachtet, da sie als Verallgemeinerung eines Gütekriteriums für einen einfachen Qualitätstest nur einen erhöhten Rechenaufwand verursacht.

4.6.1 Generalisierung

Die Generalisierung beschreibt den grundsätzlichen Unterschied zwischen einem Lookuptable und einem Modell. Während der Lookuptable[36] nichts außer bekannten Informationen enthält, versucht ein Modell diese zusätzlich zu erklären. Aus diesem Grund steht der Lookuptable auch als klassisches Beispiel für Auswendiglernen, wohingegen einem Modell stets ein gewisses Interpolationsvermögen unterstellt wird. Erfasst dann ein Modell (Neuronales Netzwerk) die seiner Informationsmenge (Datensatz) zugrunde liegende Funktion vollständig, so wird von einer optimalen Generalisierung gesprochen. Diese Suche nach einem perfekten Modell ähnelt dabei der Gralssuche.
Insbesondere für Neuronale Netzwerke existiert jedoch noch ein weiteres Unterscheidungsmerkmal in Form der Dualität von funktionaler und imparativer (objektorientierter) Programmierung. Im Gegensatz zum Lookuptable basiert ein neuronales Netz nicht auf einer assoziativen Speichertabelle, sondern auf einer Funktion[37] und kodiert sämtliche Relationen in den funktionalen Netzkomponenten. Allerdings ist die Konstruktion einer solchen (neuronalen) Funktion als Informationsspeicher ein größtenteils ungelöstes Problem.

4.6.2 Kreuzvalidierung

Die Methode der Kreuzvalidierung teilt einen Datensatz zuerst in $k \in \mathbb{N}_{\geq 2}$ Teilmengen so (beliebig) auf, dass jede der k Mengen möglichst gleich groß ist. Danach wird für das gegebene Modell (z.B. ein neuronales Netz) mit den Daten aus $k-1$ dieser disjunkten Teilmengen (Trainingsmenge) trainiert bzw. optimiert bzw. geschätzt und anschließend mit den Daten der verbleibenden Teilmenge (Testmenge) anhand eines Gütekriteriums getestet. Dieses Vorgehen wird für jede der k Teilmengen als Testmenge wiederholt und danach eine mittlere Güte berechnet.
Bezüglich des Datensatzes und dessen Aufteilung ist dann die gemittelte Güte als Generalisierung des Modells interpretierbar, wenn Trainings- und Testmenge ausreichend allgemeine und homogene Daten enthalten und das Größenverhältnis beider ihrem Kontext (z.B. Experiment, Umwelt oder Realität) angemessen ist. Der Parameter k sollte dabei mit Blick auf die Größe von Trainings- und Testmenge gewählt

[36] Ein beliebtes Beispiel hierfür ist der Chinesische Raum (Blackbox), in den man einen englischsprachigen Satz hinein gibt und eine Antwort erhält. Dabei erfolgt die Antwort jedoch nicht als Reaktion auf die Eingabe, sondern wird aus einer Tabelle abgelesen.

[37] In diesem Zusammenhang läßt sich Q-Learning als objektorientierter Ansatz des neuronalen Konzepts interpretieren.

werden. Eine Anwendung der Kreuzvalidierung mit der Wahl k = Datensatzgröße in Form von In-Sample- und Out-Sample-Fehlerraten findet sich oft bei der Untersuchung von Vorhersagemodellen. Diese Wahl von k ist oft sehr zeitaufwendig, garantiert aber die Unabhängigkeit von der Aufteilung des Datensatzes.

4.7 Chaos

Chaos und wird gängigerweise durch die Eigenschaften deterministisch, nichtlinear, stark parametersensitiv, aperiodisch und beschränkt definiert. Es existieren allerdings auch andere Definitionen, weil es bislang keiner Einigung über das Wesen einer chaotischen Funktion gekommen ist. Dabei ist besonders die Beschränkung einer chaotischen Funktion auf ein fixiertes Intervall umstritten, die eigentlich unnötig, aber in physikalischen Systemen als Stabilisierung durchaus erwünscht ist.
Prinzipiell umschreibt Chaos das Verhalten eines latenten nichtlinear deterministisch dynamischen Systems, dessen Komplexität alle bekannten Möglichkeiten einer Modellierung überschreitet. Beispiele von bekannten Modellierungen eines eindimensionalen Chaos hingegen sind die Funktionen $x_{n+1} = \sin(\pi \cdot 2^{x_n} \cdot x_n)$ und $f(x;c) = \sin(\pi \cdot 2^x \cdot c)$ mit Startwert $x_0 \neq 0$ bzw. Konstante $c \neq 0$.
Ferner können chaotische Funktionen u.a. auch zur Erzeugung von Pseudozufallszahlen genutzt werden, wobei jedoch ein kausaler Zusammenhang zwischen Chaos und Zufall bislang ungeklärt ist[38]. Dennoch ist zumindesten eine statistische Beschreibung chaotischer Prozesse möglich.
Im Falle eines Neuronalen Netzwerkes mit deterministischen, nichtlinearen, beschränkten Aktivierungsfunktionen kann ein chaotisches Verhalten anschaulich über dessen Topologie begründet werden. Wird eine Eingabe eines Neurons variiert, so hat dies Auswirkungen auf alle ihm nachfolgenden Neuronen. Diese Variation setzt sich entsprechend fort und führt dann zum bekannten Schmetterlingseffekt[39] in Form einer stark parametersensitiven und vermutlich aperiodischen Ausgabe. Die Beschränkung der Aktivierungsfunktionen scheint dabei mit Blick auf die viablen Funktionen AbsAZ und AbsBZ aus Abschnitt 2.4 nicht unbedingt erforderlich, aber in Bezug auf eine praktische physikalische Umsetzung in einem System (z.B. einem künstlichen neuronalen Netz) dennoch häufig notwendig (s.o.) zu sein.

4.8 Konditionierung

Die Konditionierung einer Aufgabe beschreibt das technische Problem, überabzählbar große Wertebereiche auf Basis der endlichen Wertedarstellung in einem Computer zu verarbeiten.
Während bei linearen Funktionen die Kondition anhand ihrer Matrixdarstellung berechnet werden kann, wird bei nichtlinearen Funktionen gängigerweise die Hessematrix als Basis der Konditionsbestimmung benutzt. Die Hessematrix einer neuro-

[38] Dies ist ein Analogon zur Frage, ob $\mathbb{NP} \equiv \mathbb{P}$ gilt.
[39] Dieser Effekt beschreibt das Unvermögen die atmosphärischen (vergleiche auch Lorenz-Attraktor) Auswirkungen des Flügelschlags eines Schmetterlings vorherzusagen.

nalen Funktion ist jedoch analytisch unbekannt und eine Approximation über das Verfahren der finiten Differenzen aufgrund der chaotischen Eigenschaften generell numerisch instabil. Für ein Neuronales Netzwerk kann demnach keine bekannte Konditionszahl berechnet werden.
Es ist jedoch zu erwarten, dass die schlechte Konditionierung zum Vorschein kommt, wenn in ihm aufgrund seiner chaotischen Natur die kleinsten und größten in einem Rechner repäsentierbaren Zahlen auftreten. Dieses Problem ist auch für viele Lernalgorithmen zu beobachten, vergleiche z.B. Abschnitt 8.1.
Eine allgemeine Möglichkeit zur Umgehung des Konditionsproblems existiert allerdings in Form der Simulation einer beliebig präzisen Arithmetik mit Hilfe von ganzen Zahlen. Hier wird das Problem aber auf die Laufzeit übertragen, welche im Fall einer schlechten Konditionierung überproportional zur benötigten Genauigkeit ansteigt. Aus diesem Grund wird eine derartiger Testaufbau hier vermieden.

5 Funktionsoptimierung

Die allgemeine Funktionsoptimierung beschreibt Verfahren, mit denen Lösungen für funktional darstellbare Probleme gefunden bzw. erlernt werden können und wird normalerweise als Funktionsminimierung auf Grundlage eines Gütekriteriums formuliert. Entsprechend sollte ein solcher Algorithmus möglichst kontextfrei entworfen werden, um robust gegenüber verschiedenen Daten bzw. Aufgabenstellungen zu sein. Tabelle 8 führt die gängigsten Klassen dieser Algorithmen auf[40].

Informationsgewinnung	**Informationsverarbeitung**	**Beispiele**
deterministisch		
Grid	Abtastung eines Hypergrids	Gridsearch
Hessematrix	Nullstellensuche in der ersten Ableitung	LM, Trust Region, DFP, BFGS
Gradient	Suche nach minimalem Punkt in Abstiegsrichtung	Armijo-Goldstein, Wolfte-Powell
Simplex	Suche nach Abstiegsrichtung mittels Polytop	Directional Search, Downhill Simplex
stochastisch		
Punkt	Randomisierte Suche	Stochastic Tunneling, Simulated Annealing
Stichprobe	Evolutionäre Suche	CMA-ES, Stochastic Reflections, Stochastic Downhill Simplex

Tabelle 8: Verfahren zur multivariaten Funktionsminimierung

Insbesondere für deterministische Verfahren ist hervorzuheben, dass sie oftmals aufgrund ihrer Informationsgewinnung nicht so robust sind wie stochastische Optimierungsalgorithmen. Während stochastische Algorithmen Stichproben benutzen können, um Statistiken zu bilden, ist dies bei deterministischen Algorithmen nur approximativ möglich. Das Simplexverfahren ist hierfür ein schönes Beispiel, wenn der Simplex als Stichprobe betrachtet wird.
Die im Rahmen Neuronaler Netzwerke auftretenden Probleme sind jedoch fast immer so beschaffen, dass nur stochastische bzw. evolutionäre Algorithmen brauchbare Lösungen liefern. Aus diesem Grund werden im Folgenden deterministische Algorithmen nicht betrachtet. Ferner sind stochastische Optimierungsalgorithmen wie Simulated Annealing fast immer in eine Untergruppe der evolutionären Algorithmen einzuordnen, weswegen die evolutionäre Terminologie meistens präferiert wird, obgleich dies die zugrunde liegende Statistik verdeckt.
Nach einem einführenden Abschnitt wird zuerst der evolutionäre CMA-ES-Algorithmus vorgestellt, der den zentralen Bestandteil des EANT-Algorithmus in Kapi-

[40] Auf eine feinere Differenzierung nach weiteren Kriterien (ableitungsfrei, restringiert, linear, Laufzeit etc.) wird für diese kurze Übersicht verzichtet.

tel 6 darstellt. Dann folgen die stochastischen Algorithmen Stochastic Reflections und Stochastic Downhill Simplex als Alternativen. Sie alle sind für NN dem bekannten ableitungsbasierten BFGS-Algorithmus auf Grundlage der Arbeiten Broyden (1970a,b), Fletcher (1970), Goldfarb (1970) und Shanno (1970) überlegen[41], weil die Berechnung finiter Differenzen zur Approximation von Ableitungen mit hohen Laufzeiten und bei chaotischen Funktionen mit unlösbaren numerischen Instabilitäten verbunden ist.

5.1 Evolutionäre Optimierung

Die evolutionäre Optimierung ist eine der wenigen relativ allgemeinen Möglichkeiten, eine Lösung für ein beliebiges Problem zu entwickeln. Wie auch andere Funktionsoptimierungen wird sie normalerweise und o.B.d.A. als Funktionsminimierung auf Basis eines Gütekriteriums formuliert. Dabei gehören die evolutionären Verfahren zu den stochastischen Algorithmen, die vor allem bei komplizierten oder unstetigen Funktionen[42] zum Einsatz kommen. Sie stellen die bislang allgemeinste und zumeist langsamste Art und Weise eines Suchmusters dar, denn oftmals bieten sich spezialisiertere Verfahren an, welche dann eine probleminhärente Informationsmenge ausnutzen und deshalb schneller sind.
Evolutionäre Algorithmen bilden rudimentär eine biologische Evolution nach, deren Ziel die Lösung eines gegebenen Problems ist. Dazu wird eine Menge von Individuen (die Bevölkerung) aus Lösungsvorschlägen erzeugt, welche zumeist (dem biologischem Genom nachempfunden) linear dargestellt werden (vergleiche Abbildung 3, Seite 19). Die Bevölkerung wird anschließend rundenbasiert so verändert, dass genau die Individuen evolutionär bevorteilt werden, die bezüglich des Problems die besten Lösungen liefern. Guten Individuen wird das Überleben oder die Zeugung von Nachkommen erleichtert, schlechten hingegen erschwert.
Codebeispiel 1 zeigt den typischen Ablauf eines solchen Verfahrens. Die evolutionären Operatoren Selektion, Rekombination und Mutation bilden die wesentlichen Schritte der biologischen Evolution abstrakt nach und werden wie folgt interpretiert:

- Der Selektionsoperator selektiert Individuen für den Evolutionsprozess. Er ist als Gewichtung interpretierbar, bei der das Gewicht Null erlaubt ist. Wird ein Indivuduum nullgesetzt und somit dem Evolutionsprozess entzogen, geht damit allerdings Information, die kostenaufwendig gewonnen wird, verloren. Der Selektionsoperator muss daher mit Bedacht eingesetzt werden.

[41] Testergebnisse zum Vergleich von BFGS und CMA-ES etc. werden nicht explizit präsentiert, sind jedoch mit der Software auf dem beiliegenden Datenmedium nachprüfbar oder auch in Hansen u. Kern (2004) nachzulesen.
[42] Praktisch bestehen dann Probleme bei der Informationsgewinnung oder -verarbeitung aufgrund zu hoher Problemdimension (Dimension des Urbildraums der betrachteten Funktion$\gg 50$) oder Zeitkosten für die Berechnung einer Ableitung.

Pseudocode 1 Beispiel eines evolutionären Algorithmus

```
Setze (Problem-) Dimensionen.
Setze Bevölkerungsgröße.
Erzeuge initiale Individuen.
Bestimme die Fitnesswerte.
WHILE (kein Abbruch) {
  Erzeuge neue Individuen mittels:
    Selektion, Rekombination und Mutation
  Bestimme die Fitnesswerte etc.
  Aktualisierung der Suche.
}
Ergebnis: das beste Individuum
```

- Der Rekombinationsoperator kombiniert mehrere Individuen zu einem neuen. Dazu benötigt er eine Anleitung, wie ein Nachkomme entstehen kann, der dem Evolutionsprozess hilfreich ist. Ohne eine derartige Anleitung kann der Operator nicht zielgerichtet eingesetzt werden und ist nutzlos oder sogar kontraproduktiv.

- Der Mutationsoperator gleicht dem Vorgehen einer stochastischen Suche. Er führt pro Individuum mindestens eine Mutation durch und sollte dafür den gesamten Suchraum verwenden können. Sein Einsatz unterliegt ansonsten keiner Einschränkung.

Für jedes evolutionäre Verfahren ist essentiell wichtig, dass die Operatoren die angestrebte Globalität (siehe Abschnitt 4.4) nicht beeinträchtigen und ihr Verhalten zielgerichtet und der Suche nach einer Lösung dienlich ist. Manche evolutionäre Algorithmen, wie z.B. NEAT, berücksichtigen diese Anforderungen jedoch bereits in ihrer Konzeption nicht, so dass ihr Anspruch angezweifelt werden muss.
Zur Abgrenzung der verschiedenen evolutionären Ansätze werden diese ferner in Klassen unterteilt[43], von denen Tabelle 9 die wichtigsten enthält. Die im Folgenden benutzten Optimierungalgorithmen gehören entweder der Gruppe der evolutionären Strategien oder einer Mischung aus erster und evolutionärer Programmierung an.

5.2 CMA-ES

Der CMA-ES-Algorithmus (Covariance Matrix Adaption - Evolutionary Strategy) wird in Hansen u. Ostermeier (1996) vorgestellt und ist eine evolutionäre Strategie zur Minimierung von Funktionen $f : \mathbb{R}^n \to \mathbb{R}$ auf Basis einer adaptiven Approximation der Hessematrix.

[43] Es existieren zahlreiche weitere Gruppen evolutionärer Algorithmen, zwischen denen die Grenzen nicht scharf definiert sind und auch keine Inklusionsordnung definierbar ist. Die Gruppe der Evolutionären Programmierung ist beispielsweise nicht eindeutig von der Gruppe der Evolutionären Strategien zu trennen.

EA (Evolutionärer Algorithmus)	Algorithmus auf Basis der Operatoren Mutation, Rekombination und Selektion mit der biologische Evolution zum Vorbild (Beispiel: Ant-Colony)
EP (Evolutionäre Programmierung)	Evolutionärer Algorithmus (unter anderem) ohne Einsatz eines Rekombinationsoperator (Beispiel: Stochastic Reflections)
ES (Evolutionäre Strategie)	Evolutionärer Algorithmus in (Eigen-) Abhängigkeit von dynamisch im Evolutionsprozess vaiierenden Größen (Beispiel: CMA-ES)

Tabelle 9: Klassifizierungen evolutionärer Algorithmen

Pseudocode 2 Ablauf des CMA-ES-Algorithmus

```
Initialisiere CMA-ES.
WHILE (kein Abbruch) {
  Ziehe eine Bevölkerung aus der Stichprobenverteilung.
  Berechne die Fitnesswerte der Bevölkerung.
  Aktualisiere die Stichprobenverteilung.
  Teste die (numerischen) Abbruchbedingungen.
}
Ergebnis: das beste Individuum
```

Seine Vorgehensweise basiert auf einer statistischen Informationsgewinnung mittels wiederholter Stichproben im Parameterraum der zu minimierenden Funktion. Aufgrund des dafür verwendeten Adaptionsprozesses ist er bei kleinen Problemdimensionen langsamer als existierende Verfahren. Erst bei höher dimensionalen Problemen zeigt sich seine Überlegenheit gegenüber anderen Algorithmen, wie z.B. dem bekannten BFGS-Algorithmmus. Dabei ist jedoch neben dem Vergleichskriterium der Zahl der Funktionsevaluationen auch das der Laufzeit zu berücksichtigen. Obwohl letztere in Abhängigkeit der Problemdimension n bereits durch einen Trick[44] von $O(n^3)$ auf $O(n^2)$ reduziert wird, ist diese immer noch relativ hoch. Aus diesem Grund existieren, allerdings zu Lasten des Umfangs der Informationsgewinnung und folglich des Ergebnisses, schnellere Varianten, siehe Igel u. a. (2006) oder Fußnote 45, die hier aber nicht eingehender betrachtet werden.

5.2.1 Vorgehensweise

Informationsgewinnung und Vorgehen des CMA-ES-Algorithmus basieren grundlegend auf der Idee einer gewichteten empirischen Schätzung einer Normalverteilung. Die Elemente x_i der Stichprobenmenge bilden dabei die Bevölkerung (mit Umfang $\lambda \in \mathbb{N}$) im Rahmen des evolutionären Verfahrens. Sei $n \in \mathbb{N}$ die Problemdimension. Zu Beginn wird für einen Startpunkt $x_0 \in \mathbb{R}^n$ (der selbst nicht ausgewertet wird)

[44] Der Trick besteht in der Aktualisierung der Kovarianz nur alle $\frac{n}{10} > 1$ Iterationsschritte.

eine n-dimensionale Standardnormalverteilung $N \sim (m_0, C_0)$ mit $m_0 = x_0$ und $C_0 = I_{n \times n}$ benutzt, um eine initiale Stichprobe zu ziehen.
Die einzelnen Stichprobenelemente werden anhand ihres Funktionswertes (Fitnesswertes) $f(x_i)$ so sortiert, dass Elementen mit guter Fitness ein hohes Gewicht zufällt. Danach wird aus den besten μ Stichprobenelementen (Selektion) ein neuer gewichteter empirischer Mittelwert m_1 (Rekombination) berechnet und nachfolgend unter Verwendung zweier Evolutionspfade ps, pc und eines Skalierungskoeffizienten σ darauf folgend die neue empirische Kovarianz C_1 (Rekombination). Anschließend werden die Elemente der nächsten Stichprobe (Generation) anhand der neuen Normalverteilung $N \sim (m_1, C_1)$ gezogen[45] (Mutation des Mittelwertes). Bei dieser Vorgehensweise stellt die gewichtete Schätzung der Kovarianzmatrix eine effektive Möglichkeit dar, ein zielgerichtetes Suchgebiet im Parameterraum aufzuspannen. Die Matrix kann ferner geometrisch als n-dimensionaler Spat[46] interpretiert werden, der sich in Richtung der hoch gewichteten Funktionswerte am stärksten ausdehnt. Wird anstelle des Spats die Normalverteilung betrachtet, so ergibt sich eine Ellipse von vergleichbarer Form. Die alleinige Schätzung der Kovarianz genügt jedoch nicht, um den Algorithmus effizient zu gestalten. Ergänzend werden die in den Unterabschnitten erläuterten Techniken eingesetzt. Dabei spielen Adaptionstechniken, siehe Einleitung dieses Kapitels, eine entscheidende Rolle.
Das iterative Vorgehen wird so lange fortgesetzt, bis ein Abbruchkriterium aus Abschnitt 4.5 erfüllt ist, die Kondition der Kovarianzmatrix einen Maximalwert überschreitet oder die Standardabweichung für eine Komponente des Stichprobenmittles (oder für einen Eigenraum der Eigenwertzerlegung) einen Schwellenwert unterschreitet.
Tabelle 10 listet die im Weiteren verwendeten CMA-ES-Variablen und ihre Beschreibungen auf, während der Pseudocode 2 dessen Vorgehensweise veranschaulicht.

[45] Eine Möglichkeit, aus einer n-dimensionalen Normalverteilung $N(0, \Sigma)$ zu ziehen, besteht in der Cholesky-Zerlegung der (semi-positiv-definiten) Kovarianzmatrix $\Sigma = L \cdot L^T$ in $O(n^2)$ und der anschließenden Linksmultiplikation einer standardnormalverteilten n-dimensionalen Zufallsvariablen $X \sim N(0, I)$ mit L. Sei $Y := L \cdot X$. Dann gilt:

$$Var(Y) = E(Y^2) - \underbrace{E^2(Y)}_{=0} = E((L \cdot X)(L \cdot X)^T) = L \cdot \underbrace{E(X \cdot X^T)}_{=1} \cdot L^T = \Sigma$$

Möchte man jedoch die Basis der Kovarianzmatrix verändern, so muss auf die PCA bzw. Eigenwertzerlegung $\Sigma = B \cdot D \cdot B^T$ in $O(n^3)$ zurückgegriffen werden, wobei B eine orthogonale ($B^{-1} = B^T$) Transformations- und D die Diagonalmatrix der Basisvektoren ist. Sei $D = B^T \cdot \Sigma \cdot B$ die Eigenwertzerlegung von Σ, und sei nun $Y := B \cdot \sqrt{D} \cdot Z$ mit $Z := B^T \cdot X$. Dann gilt $Var(Z) = E(Z \cdot Z^T) = E(B^T \cdot X \cdot X^T \cdot B) = I$ und damit auch:

$$Var(Y) = E((B \cdot \sqrt{D} \cdot Z) \cdot (B \cdot \sqrt{D} \cdot Z)^T) = E(B \cdot \sqrt{D} \cdot \underbrace{Z \cdot Z^T}_{=1} \cdot \sqrt{D}^T \cdot B^T) = B \cdot D \cdot B^T = \Sigma \quad (7)$$

Zugleich ist Gleichung 7 der Hauptbestandteil einer Hauptkomponentenanalyse von Σ.

[46] Auch bekannt als Parallelotop, Parallelepiped oder Paralellflach.

Variable	Beschreibung	Wertebereich
λ	Bevölkerungsgröße	$\mathbb{N}$
n	Dimension des Urbildes der betrachteten Funktion (Problemdimension)	$\mathbb{N}$
m	Mittelwert	$\mathbb{R}^n$
C	Kovarianzmatrix	$\mathbb{R}^{n \times n}$
ps	Evolutionspfad der Schrittweitenkontrolle	$\mathbb{R}^n$
σ_0	Initialer Skalierungskoeffizient der Kovarianzmatrix	$\mathbb{R}^+$
pc	Evolutionspfad des Kovarianz-Rank-1-Updates	$\mathbb{R}^n$
μ	Anzahl der besten Individuen zur Berechnung des neuen Mittelwertes und des Rank-μ-Updates	$\mathbb{N}_{\leq \lambda}$
w	Skalierte Gewichtskoeffizienten zur Berechnung des neuen Mittelwertes und des Rank-μ-Updates	$]0,1[^{\mu}$
c_c	Koeffizient der Adaption des Evolutionspfades für das Rank-1-Update der Kovarianz	$[0,1]$
c_s	Koeffizient der Adaption des Evolutionspfades für die Schrittweitenkontrolle	$[0,1]$
d_s	Dämpfungsfaktor in der Berechnung der Schrittweite	$[0,1]$
c_{cov}	Koeffizient der Kovarianzadaption	$[0,1]$
μ_{eff}	Kennzahl der Gewichtungsart der Rekombination	$[1,\mu]$
μ_{cov}	Gewichtungskoeffizient zwischen Rank-1- und Rank-μ-Update bei der Adaption der Kovarianzmatrix	$\mathbb{R}_{\geq 1}$

Tabelle 10: CMA-ES-Variablen

5.2.2 Schrittweitenkontrolle

In einem evolutionären (stochastischen) Funktionsminimierungsalgorithmus gibt es keine direkte Abstiegsrichtung wie bei der klassisch gradientenbasierten Funktionsminimierung, so dass auch nicht gezielt nach einer optimalen Schrittlänge gesucht werden kann. Ein Evolutionspfad ist ein Ansatz, um trotzdem eine Schrittlänge, genauer eine Schrittweite, benutzen zu können.

Der Evolutionspfad $ps \sim N(0, C)$ ist definiert als die Summe aller mit zugehöriger Schrittweite und Kovarianz standardisierten Mittelwertdifferenzen. Er gibt die kumulierte Länge und Richtung aller getätigten dekorrelierten und deskalierten Schritte[47] beginnend bei x_0 an und stellt eine sehr nützliche Informationsquelle dar. Seine Berechnung wird adaptiv vorgenommen, so dass rückwärts gerichtete Schritte

[47] Der Evolutionspfad ist aufgrund von Dekorrelation und Deskalierung standardnormalverteilt.

nicht sofort eine große Auswirkung haben:

$$ps_{i+1} = \underbrace{(1-c_\sigma)\cdot ps_i}_{\text{Adaption}} + \underbrace{\sqrt{c_\sigma\cdot(2-c_\sigma)\cdot\mu_{eff}}\cdot}_{\text{Normierung auf } N(0,C)} \underbrace{C_i^{-\frac{1}{2}}}_{\text{Dekorrelation}} \cdot \underbrace{\frac{1}{\sigma_i}\cdot(m_{i+1}-m_i)}_{\text{Update}}$$

Gleichwertig zu klassischen Minimierungsalgorithmen kann - nun im stochastischen Kontext - die Weite des nächsten Schrittes bzw. die erwartete Lage des nächsten Mittelwertes optimiert werden. Dazu wird die Kovarianz durch die Multiplikation mit dem Koeffizienten[48] $\sigma_i \in [0, 1e6]$ vergrößert oder verkleinert, so dass die Elemente der nächsten Stichprobe dementsprechend weiter voneinander entfernt oder näher zusammen liegen. Die Lage des nächsten Mittelwertes ist somit über die Elemente der nächsten Stichprobe abhängig von der Größe der Einträge der nächsten Kovarianzmatrix.
Ist der Evolutionspfad kurz, weil sich viele der Mittelwertaktualisierungen (s.u.) ausgelöscht haben, wird die Schrittweite verkleinert. Ist der Evolutionspfad dagegen lang, weil die Mittelwertaktualisierungen korreliert sind, wird die Schrittweite vergrößert. Die notwendige Wahlentscheidung wird dabei mit Hilfe des Skalarproduktes getroffen. Dazu wird im CMA-ES-Algorithmus das Skalarprodukt des $N(0,1)$-verteilten Evolutionspfades ins Verhältnis zu dem des Erwartungswertes eines standardnormalverteilten Zufallsvektors gesetzt, und die verwendete, um Adaptionskonstanten ergänzte, Schrittweitenregel ergibt sich zu:

$$\sigma_{i+1} = \exp\left\{\frac{c_s}{d_s}\cdot\left(\frac{||ps_{i+1}||_2^2}{\mathrm{E}\left(||N(0,1)||_2^2\right)}-1\right)\right\}\cdot\sigma_i \tag{8}$$

Die Schrittweite wird also nicht multiplikativ, wie bei klassischen Schrittweitenregeln angepasst[49], sondern exponentiell.

5.2.3 Aktualisierung des Mittelwertes

Die Aktualisierung des (Stichproben-) Mittelwerts besteht in der Berechnung des gewichteten Durchschnitts der $\mu \in \mathbb{N}_{\leq\lambda}$ besten Stichprobelemente

$$\mu_{n+1} = \sum_{i=1}^{\mu} w_i x_i$$

und entspricht im evolutionären Kontext dem gewichteten Mittelwert einer Rekombination der zuvor selektierten besten Individuen. Dabei gilt für die Gewichte $w_i \in \mathbb{R}^+$ sowie die Normalisierungsbedingung $\sum_{i=1}^{\mu} w_i = 1$. Die Kennzahl $\mu_{eff} := \sum_{i=1}^{\mu}\left(\frac{w_i^2}{\sum_{i=1}^{\mu} w_i^2}\right)$ beschreibt ferner die Art der Gewichtung. Für $\mu_{eff} = \mu$

[48] Die von CMA-ES übernommene Notation kann irritieren, weil das Symbol σ der statistischen Standardabweichung in Form eines ausgeklammerten Koeffizienten der Kovarianzmatrix hier als Schrittweite der Kovarianzadaption $\sigma \cdot \Sigma$ verwendet wird.

[49] Die bekannten Linesearchalgorithmen zur Schrittweitenbestimmung bei Gradientenverfahren (beispielweise Wolfe-Powell- oder Armijo-Goldstein-Regel) benutzen zumeist eine Aufweitung um Faktor 2 oder eine Reduktion um Faktor $\frac{1}{2}$ zur Anpassung der Schrittweite.

sind alle Gewichte identisch gleich 1. Für kleinere Werte verläuft sie hingegen exponentiell abfallend. Die Standardwahl der Gewichte ist durch die Gleichung

$$w_i = \log(\mu + 1) - \log(i + 1) \tag{9}$$

gegeben[50], für welche μ_{eff} asymptotisch in μ gegen $\frac{\mu}{2}$ läuft.

5.2.4 Adaption der Kovarianzmatrix

Eine gute Schätzung der Kovarianz benötigt eine große Stichprobenmenge, je größer, desto besser. Jedes einzelne Stichprobenelement stellt jedoch eine Funktionsevaluation dar, die mit Blick auf die Laufzeit vermieden werden sollte.
Im Zuge der Minimierung der notwendigen Funktionsevaluationen wird daher eine Adaption der Kovarianzmatrix samt Adaptionskoeffizienten eingeführt. Sind zwei nacheinander geschätze Kovarianzmatrizen ähnlich, lohnt sich eine Adaption. Andernfalls sind mehrere Adaptionen nötig, um die neue Kovarianzmatrix zu erhalten. Der Adaptionskoeffizient $c_{cov} \in [0, 1]$ legt dabei die Geschwindigkeit der Adaption fest.
Für die Adaption selbst werden die aufsummierten gewichteten Matrizen der äußeren Produkte der $\mu \in \mathbb{N}_{\leq\lambda}$ besten standardisierten Stichprobenelemente benutzt. Dies ist ein Rank-μ-Update bestehend aus der empirischen Kovarianz C_μ dieser gewichteten Elemente:

$$C_{n+1} = (1 - c_{cov}) \cdot C_n + c_{cov} \cdot C_\mu$$

Für den Fall $\mu = 1$ ergibt sich das klassische Rank-1-Update, auf welchem beispielsweise auch der BFGS-Algorithmus beruht. Im evolutionären Kontext entspricht dies ebenfalls Selektion und Rekombination.
Die Effizienz des Algorithmus kann ferner gesteigert werden, wenn die Richtungsinformation des Vektors der Mittelwertdifferenz, welche beim Rank-μ-Update nicht genutzt wird, verwendet wird. Dazu wird ein weiterer Evolutionspfad konstruiert, der jedoch nicht-standardisierte Mittelwertdifferenzen benutzt:

$$pc_{i+1} = \underbrace{(1 - c_c) \cdot pc}_{\text{Adaption}} + \underbrace{\sqrt{c_c \cdot (2 - c_c) \cdot \mu_{eff}}}_{\text{Normierung auf } N(0,C)} \cdot \underbrace{\frac{1}{\sigma_i} \cdot (m_{i+1} - m_i)}_{\text{Update}}$$

Er wird dann über den Balancierungskoeffizifenten $\mu_{cov} \in [0, 1]$ als Rank-1-Update in die Adaptionsgleichung eingebunden. Insgesamt ergibt sich somit folgende Adaptionsgleichung für die Kovarianzmatrix:

$$C_{n+1} = \underbrace{(1 - c_{cov}) \cdot C_n}_{\text{Adaption}} + \underbrace{\frac{c_{cov}}{\mu_{cov}} \cdot \left(pc \cdot pc^T\right)}_{\text{Rank-1-Update}} + \underbrace{c_{cov} \cdot \left(1 - \frac{1}{\mu_{cov}}\right) \cdot C_\mu}_{\text{Rank-}\mu\text{-Update}}$$

[50] Dieser konvexe abfallende Verlauf der Gewichtsreihe wird als super-linear bezeichnet.

5.2.5 Konvergenz und Skalierung

Sobald ein Eintrag des Mittelwertvektors über mehrere Generationen nahezu unverändert bleibt, läßt die Adaption das Verfahren in dieser Dimension konvergieren, wobei im Zuge der Kovarianzadaption gleichzeitig die Varianz dieser Komponente und ihres Suchgebietes im Parameterraum gegen Null geht. Dabei, jedoch auch im normalen Ablauf des Verfahrens, können sich numerische Probleme ergeben, wenn eine genügend große Differenz zwischen minimaler und maximaler Kovarianz bei Berechnungen mit der Kovarianzmatrix zur ungewollten, technisch bedingten Auslöschung von kleinen Zahlen bzw. vielen Nachkommastellen führt.

Dieses Konditionierungsproblem der Kovarianzmatrix taucht im Algorithmus vornehmlich bei der Berechnung der (symmetrischen) Eigenwertzerlegung[51], die zur Erzeugung einer neuen Generation bzw. Stichprobe gebraucht wird, auf. Unabhängig von einer ebenfalls problematischen linearen Abhängigkeit der Matrixspalten, tritt es insbesondere dann (in Abhängigkeit der Problemdimension) in Erscheinung, wenn die Kovarianzen um mehrere Größenordnungen auseinanderfallen. Die numerische Stabilität der Matrixzerlegung ist in einem solchen Fall nicht mehr durch die verwendete lineare Skalierung zu gewährleisten[52] und der Algorithmus beginnt sich merkwürdig zu verhalten oder in den Dimensionen, deren Varianz ausgelöscht wird, zu stagnieren.

Würde hingegen eine nichtlineare Skalierung bzw. Transformation benutzt, könnten zu Abschnitt 2.2.1 vergleichbare Skaleneffekte auftreten. Insofern korrespondiert das Konditionsproblem mit dem Auswahlproblem der Aktivierungsfunktion in einem Neuronalen Netzwerk. Da aber die Zustände und das Verhalten einer Matrix A als linearer Funktion $x \mapsto Ax$ im Gegensatz zu einer neuronalen Funktion stets vollständig bekannt sind, sollte sich eine geeignete bijektive nichtlineare Skalierung bzw. Transformation vermutlich finden lassen, mit welcher das Versagen der linearen Skalierung bei der Kovarianz zumindest abgemildert werden könnte.

Ein Indikator für die Existenz dieses Problems ist das starke Anwachsen des Verhältnisses $\frac{d_{max}}{d_{min}}$ von größtem Eigenwert d_{max} zu kleinstem Eigenwert d_{min} der Eigenwertzerlegung. Dieses Verhältnis ist zugleich das maximale Achsenverhältnis der Hauptkomponentenanalyse (PCA) bzw. der orthogonalisierten Kovarianzmatrix[53]. Steigt es stark an, versagt simultan zunehmend die eingesetzte (lineare) Skalierungstechnik zur numerischen Stabilisierung[54], welche in CMA-ES als Doppelverwendung

[51] Dies gilt für jede Matrixzerlegung, insbesondere auch für die Alternative der Cholesky-Zerlegung.

[52] Die in C geschriebene Variante von CMA-ES benutzt eine lineare Skalierung, um zu kleine Werte zu verhindern, so dass numerische Instabilitäten durch zu große Zahlen entstehen. Die in Java geschriebene Variante verhindert diese jedoch mit einer zweiten linearen Skalierung. Ab einer gewissen Größe der Achsenverhältnisse beginnen sich die beiden Skalierugen jedoch gegenseitig zu bedingen, so dass sie sich aufheben oder eine Oszilliation zwischen ihnen einsetzt. Das Konditionsproblem kann somit in beiden Varianten nicht gelöst werden. (Dabei sind die zugelassenen Minimal- und Maximalwerte Metaparameter.)

[53] Siehe dazu auch Fußnote 45.

[54] Die in der Programmiersprache C verfasste Variante von CMA-ES benutzt eine lineare Skalie-

von σ als Skalierungskoeffizient der Kovarianz und Schrittweite realisiert wird. Da ferner die Eigenwerte der orthogonalisierten Kovarianzmatrix identisch mit den Varianzen der Eigenräume sind, stehen sinkende Werte für ein eingeschränkteres und steigende Werte für ein ausgedehnteres Suchgebiet im jeweiligen Eigenraum.
Als hauptsächliche Abbruchbedingung wird deshalb das varianzverringernde Verhalten der Kovarianzadaption eingesetzt, während weitere Bedingungen z.B. das numerische Achsverhältnis prüfen.

5.3 Stochastic Reflections

Der Algorithmus Stochastic Reflections ist als Demonstration einer evolutionären Strategie mit Laufzeit $O(n)$ zur Minimierung von Funktionen $f : \mathbb{R}^n \to \mathbb{R}$ konzipiert und neben zahlreichen anderen evolutionären Minimierungsalgorithmen während der Analyse des CMA-ES-Algorithmus entstanden.
Seine bewusst einfach gehaltene Vorgehensweise basiert analog zu CMA-ES ebenfalls auf einer statistischen Informationsgewinnung in Form einer Stichprobe. Nach Erzeugung dieser initialen Bevölkerung mittels stochastischer Störung des Startvektors werden die Fitnesswerte berechnet und das beste Individuum bestimmt. In der anschließenden Iteration wird dieses selektiert und gespeichert, bevor die gesamte Bevölkerung mutiert wird. Der randomisierte Mutationsschritt besteht dabei aus einer komponentenweisen (dimensionsweisen) Aktualisierung der Individuen (Lösungsvektoren):

$$\text{Individuum}_j = \text{Letztbester}_j + (\text{Letztbester}_j - \text{Individuum}_j) \cdot \text{N}(0,1) \qquad (10)$$

Gleichung (10) enthält simultan die Rekombination eines Individuums mit dem Letztbesten und die Mutation in Form einer standardnormalverteilten Störung. Die Differenz der j-ten Komponenten zwischen Individuum und dem Letztbesten entscheidet hierbei über die Größe der Varianz[55] der normalverteilten Störung sowie über das Vorzeichen des Veränderungsschrittes. Nach erfolgter Mutation wird dann das schlechteste Individuum durch das letztbeste ersetzt, anschließend das nun beste Individuum selektiert und das Vorgehen wiederholt. Nach einigen Neustarts des Algorithmus an der Stelle des bis dahin gefundenen Minimums terminiert die Iteration mit dem (allzeit-) besten Individuum als Ergebnis, wenn eine Abbruchbedingung[56] (siehe auch Abschnitt 4.5) erfüllt ist. Für diesen Algorithmus wird neben

rung, um zu kleine Werte zu verhindern, so dass in der Folge numerische Instabilitäten durch zu große Zahlen verursacht werden können. Die in Java geschriebene Variante verhindert zudem durch eine zweite lineare Skalierung auch zu große Werte. Ab einer gewissen Größe der Achsenverhältnisse beginnen sich die beiden Skalierungen jedoch gegenseitig zu bedingen, so dass diese sich aufheben oder eine Oszilliation zwischen ihnen einsetzt. Das Konditionsproblem kann in beiden Varianten nicht gelöst werden. (Dabei ist der zugelassene Minimal- bzw. Maximalwert ein Metaparameter.)

[55] Die Schrittweitensteuerung erfolgt also differenziert nach Dimension.

[56] Die minimal zugelassene Varianz der Fitnesswerte in der Bevölkerung beträgt $1E-16$ und die maximale Anzahl an Iterationen 100000.

der maximalen Iterationsanzahl eine minimale Toleranz für die Varianz der Fitness über ein vorzugebendes Zeitfenster (historische Varianz) als Abbruchkriterium definiert.
Trotz der Vorgehensweise ist jedoch eine Zuordnung zur Gruppe der evolutionären Strategien nicht ohne weiteres möglich, da zwar das Verhalten des Algorithmus mit dem Abstand zum letztbesten Individuum dynamisch variiert, dieses aber parameterfrei geschieht und somit die Vorgehensweise insgesamt nicht variiert.

5.4 Stochastic Downhill Simplex

Eine bekannte weitere ableitungsfreie Methode zur Funktionsminimierung ist der in Nelder u. Mead (1965) vorgestellte Downhill-Simplex-Algorithmus. Für eine n-dimensionale Funktion $f : \mathbb{R}^n \to \mathbb{R}$ wird ein $(n+1)$-dimensionaler Simplex (Polytop) erzeugt, dessen Ecken im Kontext evolutionärer Algorithmen auch als Individuen einer Bevölkerung interpretiert werden können. Die Randomisierung der hier verwendeten Variante des Downhill-Simplex besteht in der Konstruktion des Simplex. Dieser wird aus dem Startvektor und n komponentenweise $\mathrm{N}(0, 0.1)$-gestörten Kopien erzeugt.
Zu Beginn eines jeden Iterationsschritts werden bester und schlechtester Eckpunkt und Zentrum des Polytops bestimmt. Anschließend wird der schlechteste Eckpunkt am Zentrum gespiegelt

$$\text{Testpunkt}_i = \text{Zentrum}_i + c \cdot (\text{Zentrum}_i - \text{Ecke}_i)$$

(Reflektion, $c = 1$) und der Funktionswert dieses Testpunkts mit dem des besten verglichen.
Eine Schrittweitensteuerung findet dabei wie folgt statt. Ist der Testpunkt besser, wird ein zweiter Testpunkt als expandierende Reflektion (Expansion, $c > 1$) des schlechtesten Eckpunktes berechnet und dieser durch den besseren der beiden Testpunkte ersetzt.
Ist der Testpunkt schlechter, wird die Anzahl der besseren Punkte bestimmt. Ist die Anzahl echt kleiner als n, so ist der Testpunkt besser als der schlechteste und dieser wird durch den Testpunkt ersetzt. Andernfalls ist der Testpunkt mindestens genauso schlecht wie der schlechteste Eckpunkt, und es wird eine weitere Fallunterscheidung vorgenommen. Ist der Testpunkt schlechter als der schlechteste Eckpunkt, so wird der Eckpunkt durch seine eigene kontrahierende Reflektion (Kontraktion, $c < 1$) am Zentrum ersetzt, andernfalls wird er durch die kontrahierende Reflektion des Testpunkts ersetzt. Als Abbruchkriterium[57] (siehe auch Abschnitt 4.5) kommt in der Regel das Volumen des Simplex zum Einsatz, wobei ein hinreichend kleines Volumen zum Abbruch führt. Wird eine Abbruchbedingung erfüllt, so vollzieht der Algorithmus einen Neustart, indem an der Stelle des aktuell erreichten Minimums ein neuer Simplex erzeugt und das Verfahren wiederholt wird. Der Algorithmus terminiert mit dem (allzeit-) besten Individuum als Ergebnis.

[57] Die minimal zugelassene Größe des Simplexvolumens beträgt $1E-12$ und die maximale Anzahl an Iterationsschritten 100000.

5.5 Diskussion

Werden die Algorithmen der letzten Abschnitte betrachtet, so sind zwischen ihnen neben Parallelen in der Vorgehensweise erhebliche Unterschiede in der Komplexität und damit meistens einhergehend auch in der Laufzeit zu bemerken. Die verschiedenen Algorithmen besitzen dabei vornehmlich unterschiedliche Eigenschaften in Informationserhebung und -nutzung.
Im Gegensatz zu den (klassischen) deterministischen Techniken ermöglichen die stochastische Informationsgewinnung und statistische -verarbeitung auf Basis von Stichproben jedem Algorithmus, der sie einsetzt, eine hohe Robustheit[58], z.B. gegenüber Funktionen mit vielen lokalen Minima oder schroffer Oberfläche. Das derartige (adaptive) Vorgehen in CMA-ES ist diesbezüglich beispielhaft.
Wie aufgrund seiner außergewöhnlichen Konzeption zu erwarten ist, sammelt und nutzt der CMA-ES-Algorithmus dabei in Form der Verwendung von Statistiken zweiter Ordnung (Kovarianz) die größte Informationsmenge unter den betrachteten Algorithmen. Für die (Problem-) Dimension n des Urbildraumes einer Funktion schlägt sich dies allerdings unmittelbar in einer Laufzeit von $O(n^2)$ pro Iterationsschritt nieder[59], so dass die Algorithmen SR und SDS mit einer Laufzeit von $O(n)$ pro Schritt eine Alternative darstellen. Praktisch dürfte CMA-ES diesen Verfahren dennoch - vor allem bei schwierigen Funktionen - überlegen sein, weil sein Minimierungsprozess vorgehensbedingt vermutlich schneller terminiert.
In dieser Hinsicht wird eine differenzierte Effizenzanalyse jedoch dadurch erschwert, dass keine theoretisch fundierten Vergleichs- bzw. Gütekriterien für die (informationsbasierte) Vorgehensweise von Optimierungsalgorithmen (über verschiedene Gattungen hinweg) existieren. Außer einer auf den Iterationsprozess beschränkten Konvergenzaussage ist allgemein kein Maß für den Umgang eines Algorithmus mit Information bekannt. Ein solches Kriterium würde neben der (erwarteten) Laufzeit als Effektivitätsmaß ein wünschenswertes Effizienzmaß für eine bessere Vergleichbarkeit abgeben, die bislang nur über Benchmarks realisierbar ist. Tabelle 11 lis-

Ebene	**Aufgabe**	**Schritt**	**Iteration**	**Evolution**	**Verfahren**
Informations-	initiale	äußere	adaptive	evolutionäre	vorgehens-
menge	erzeugte	gewonnene	momentane	statistische	abhängige
	verwendete	verarbeitete			
Effizienz-	Größe der Infor-	Laufzeit eines	Konvergenzge-	Abdeckung des	Gütekriterium
kriterium	mationsmenge	Schrittes	schwindigkeit	Suchraumes	

Tabelle 11: Effizienzbetrachtung mit Informationsmengen

tet denkbare Informationsmengen auf, die ein solches Effizienzmaß berücksichtigen

[58] Dies gilt auch für die auf Metropolis u. a. (1953) zurückgehende Technik des Simulated Annealing, die insbesondere nur ein einziges Element pro Stichprobe (in Form des Funktionsarguments pro Iterationsschritt) benutzt.

[59] Ohne den Trick, dass die Eigenwertzerlegung der Kovarianzmatrix nur alle $\frac{1}{n}$ Iterationsschritte berechnet wird, liegt die CMA-ES-Laufzeit sogar in $O(n^3)$.

könnte. Gerade für randomisierte bzw. stochastische Algorithmen besteht hier wegen ihres informationsbasierten Ansatzes ein hoher Forschungsbedarf.

6 Neuronale Optimierung

Neuronale Optimierung beschreibt klassisch die gewichtsbasierte Anpassung (Backprop, Blackbox) eines gegebenen Neuronalen Netzwerkes (eines Modells) an eine gestellte Aufgabe im Rahmen eines bestimmten Lernparadigmas. Dieses Vorgehen kann allerdings für die Suche nach einem Modell, das einen Datensatz (Überwachtes Lernen) erklären oder einer Aufgabe nachkommen (Unüberwachtes und Verstäkendes Lernen) soll, zu unflexibel sein, weil die fixierte Struktur eines gegebenen Neuronalen Netzwerkes eine begrenzte Plastizität aufweist und dessen chaotische Eigenschaften besonders für große Strukturen die optimale Wahl der Gewichte mit bekannten Mitteln zunehmend unwahrscheinlich werden lassen.
Aus diesem Grund wird mittlerweile die (Modell-) Suche auf den Raum der Netzstruktur ausgedehnt und nach geeigneten kleinen Strukturen mit nur schwachen chaotischen Eigenschaften gesucht. Unter Beibehaltung der Optimierung der Netzgewichte werden dabei zur Strukturentwicklung hauptsächlich evolutionäre additiv-subtraktive Methoden (NEAT, EANT) eingesetzt, weil bislang kaum alternative Konstruktionsmöglichkeiten (TS) mit annehmbarer Laufzeit bekannt sind. Insbesondere erweist sich die Existenzaussage in Hornik u. a. (1989) bzw. Hornik (1990) über Neuronale Netzwerke als universelle Approximatoren[60] für deren praktische Konstruktion als unbrauchbar. Zudem ist unklar, ob die Verwendung mehrschichtiger neuronaler Netze Vorteile gegenüber der Betrachtung einschichtiger Netze hat. Nur die Vermutung, dass mit mehreren Schichten einfacher ein plastisches Verhalten erreichbar ist, liegt aufgrund von mehr möglichen Verbindungen nahe.
Die anvisierte Suche nach einem allgemeinen Verfahren zur Erzeugung einer neuronalen Topologie beschränkt sich mangels Wissens somit vorerst auf die Untersuchung einfachster Vorgehensweisen.
Für bestimmte Einschränkungen dieser Suche sind jedoch bereits empirisch bewährte Verfahren bekannt, die entweder auf besonderen Netzwerken oder speziellen Voraussetzungen basieren, dies sind u.a.:

- Die Perzeptron-Lernalgorithmen nach Rosenblatt (1962), welche nur für ein einzelnes Perzeptron konzipiert sind.

- Der Backprop-Algorithmus für Überwachtes Lernen, welcher $(< \cdot, \cdot >, g)$-Neuronen in der verdeckten und der Ausgabeschicht voraussetzt, wobei die Aktivierungsfunktion g differenzierbar sein muss.

- Der RBF-Lernalgorithmus, welcher mittels k-Means und eines linearen Gleichungssystems nur auf den einschichtigen RBF-Netzwerken arbeitet.

[60] Zusammen mit dem Skalarprodukt als Propagierungsfunktion sind der Aussage nach bereits SLN mit beschränkten, nichtkonstanten und stetigen Aktivierungsfunktionen universelle Approximatoren von stetigen Funktionen auf kompakten Intervallen. Vergleiche dazu auch Abschnitt 2.4.

- Der DCS-Algorithmus, welcher mittels k-Means, einer speziellen Nachbarschaftssuche und eines linearen Gleichungssystems nur auf DCS-Netzen (einer Kombination von SOM und RBF-Netzen) arbeitet.
- Die SOM-Trainingsalgorithmen für Unüberwachtes Lernen, welche mittels „The-Winner-Takes-It-All"-Heuristik und z.B. k-Nearest-Neighbours-Methode nur auf speziellen Netzwerken funktionieren.

Steht hingegen, wie auch bei der allgemeinen Suche, nicht genügend Information zur Auswahl eines speziellen Trainingsalgorithmus zur Verfügung, verbleibt der Entwurf einer problemabhängigen Maßfunktion (Nutzenfunktion) für Verstärkendes Lernen. Mit dieser Bewertungsfunktion kann dann ein Algorithmus auf Grundlage einer Blackbox entworfen werden[61].

6.1 Gewichtsoptimierung

Die populärste Möglichkeit, ein Neuronales Netz (der zweiten Generation) zu trainieren, ist die Optimierung seiner Gewichte. Doch existert bislang kein Netzwerk, dessen Bestandteile zur Selbstmodifikation fähig sind, stattdessen wird die Optimierung stets von einem externen Algorithmus durchgeführt.
Dabei wird zunächst zwischen Überwachtem und Verstärkendem Lernen differenziert, um - der jeweiligen Informationsmenge angemessen - vorgehen zu können. So liegt als spezielle Information entweder ein Residuum aus Soll- und Istwert an jedem Neuron der Ausgabeschicht vor oder (nur) ein Skalar mit kontextabhängiger Interpretation.

6.1.1 Backprop

Bei Überwachtem Lernen und fixierter Netzstruktur werden die Gewichte des Neuronalen Netzwerkes üblicherweise mit dem Algorithmus Backprop trainiert. Er wird in Rumelhart u. a. (1988) vorgestellt und basiert auf der rückwärts gerichteten Propagierung des quadratischen Fehlers der aktuellen Netzausgabe. Dieses Residuum wird von der Ausgabe- bis zur Eingabeschicht durchgereicht, wobei für jedes Neuron ein lokaler Fehler berechnet und vom weiter zu reichenden Residuum subtrahiert wird. Die Gewichte der eingehenden Verbindungen der Neuronen werden dann so verändert, dass der lokale Fehler und in der Folge der globale Fehler in Form des Residuums an der Ausgabeschicht minimiert wird. Die einzigen Voraussetzungen dieses Ansatzes sind Skalarprodukte als Propagierungsfunktionen und die Differenzierbarkeit der Aktivierungsfunktionen.
Die ebenfalls bekannte Variante Quickprop aus Fahlman (1988) und weitere Abkömmlinge[62] werden hier aber zu Gunsten der allgemeinen Blackboxtechnik bzw. des Verstärkenden Lernens nicht eingehender behandelt.

[61] Eine historische Übersicht über zahlreiche konstruktive Verfahren für Neuronale Netzwerke findet sich u.a. in Prechelt (1995).
[62] Der Stuttgarter Neural Network Simulator enthält fast alle bekannten Varianten des Backprop

6.1.2 Blackbox

Seien $r, s, t \in \mathbb{N}$. Bei Verstärkendem Lernen und fixierter Netzstruktur wird das Neuronale Netzwerk üblicherweise (siehe Abschnitt 4.3) als auf die Eingabe $x \in \mathbb{R}^s$ bedingte Funktion des Vektors seiner Gewichte $w \in \mathbb{R}^r$ betrachtet. Das Netz entspricht somit der bedingten Funktion:

$$n : \mathbb{R}^r \times \mathbb{R}^s \to \mathbb{R}^t,\ n(w;\, x) \mapsto y$$

Dabei ist n die Anzahl der Ausgabeneuronen. Seien $a, b \in \mathbb{R}$. Zusätzlich wird eine geeignete Maßfunktion

$$f : \mathbb{R}^t \to \mathbb{R},\ f(y) \mapsto [a, b]$$

definiert, welche die Qualität der Netzausgabe als Lösungsvorschlag anhand der gestellten Aufgabe mit einem Skalar, z.B. $x \in [0, 1]$, bewertet. Die Hintereinanderausführung stellt dann (vergleiche Abschnitt 4.3) eine bedingte neuronale Blackboxfunktion $b(w;\, x_i) \equiv f(n(w;\, x_i))$ dar, die mit Hilfe multivariater numerischer Minimierungsverfahren trainiert bzw. minimiert werden kann. Dies kann online für jedes Datum i als $b(w;\, x_i)$ oder offline über den Datensatz als $\sum_i b(w;\, x_i)$ erfolgen[63]. Dabei ist zu berücksichtigen, dass die Blackboxfunktion aufgrund der Gewichtsanzahl r fast immer hochdimensional ist und in der Nähe eines Optimums eine zumeist stark zerklüftete Oberfläche besitzt. Ein ähnliches Verhalten, das auch in Abschnitt 4.7 antizipiert wird, weisen chaotische Funktionen auf.

6.2 NEAT

Einer der wenigen Algorithmen, welche sowohl Struktur als auch Gewichte von Neuronalen Netzwerken optimieren, ist der NEAT-Algorithmus (NeuroEvolution of Augmenting Topologies). Er wird in Stanley u. Miikkulainen (2002) vorgestellt und verwendet ausschließlich evolutionäre Praktiken. Die hier benutzte Variante[64] ist der Java-basierte NEAT4J-Algorithmus[65], welcher u.a. auf der offiziellen Internetseite zu NEAT neben anderen Implementierungen explizit hervorgebohen wird. Die zugrunde liegende Datenstruktur für Neuronale Netzwerke ist ein Feld aus Genen mit eindeutigen Identifikations- bzw. Innovationsnummern[66]. Es enthält zuerst

---

und ist unter der Internetadresse `http://www-ra.informatik.uni-tuebingen.de/SNNS` zu finden.

[63] Enthält jedes Datum einen Referenzwert z, kann direkt, z.B. mit der bedingten Fehlerfunktion

$$f : \mathbb{R}^t \times \mathbb{R}^t \to \mathbb{R},\ f(y;\, z) \mapsto ||y - z||_2\,,$$

Überwachtes Lernen in Verstärkendes Lernen überführt werden.

[64] Der ursprüngliche NEAT-Algorithmus ist in den Programmiersprachen C und Lisp geschrieben und unter der Internetadresse `http://www.cs.ucf.edu/~kstanley/neat.html` an der University of Central Florida erhältlich.

[65] NEAT4J ist unter der Internetadresse `http://neat4j.sourceforge.net` erhältlich.

[66] Innovationen in Form von Veränderungen an einem Genom bzw. an einem Chromosome werden in einer Datenbank gespeichert, um wiederholte Innovationen verhindern zu können.

die Neuronen und nachfolgend die Verbindungen und ist damit noch einfacher konzipiert als das LinearGenome aus Abschnitt 3.1.
Der Ablauf des NEAT4J-Algorithmus ist in Pseudocode 3 und 4 zu sehen und gestaltet sich wie folgt. Die initiale Bevölkerung besteht aus kleinen, randomisiert generierten Netzwerken ohne verdeckte Schicht. Die benutzte Optimierungtechnik basiert auf einer stochastischen Suche[67] im neuronalen Parameterraum (Struktur und Gewichte).
Nachdem zu Beginn einer Epoche bzw. Generation die Fitness für jede Spezie und die Gesamtfitness berechnet wird, erfolgt die Aufspaltung der Bevölkerung in eine vordefinierte Anzahl von Spezien, so dass je zwei kompatible Individuen derselben Spezie angehören. Diese disjunkte Aufteilung wird über ein Kompatibilitätsmaß vorgenommen, das auf der Länge der Genome, der kumulierten Gewichtsdifferenz und der Anzahl unterschiedlicher Gene aufbaut. Neue Spezien entstehen, wenn keine Zuordnung zu einer existierenden Spezie möglich ist. Bestehende Spezien werden ausgelöscht, wenn sie zu alt oder nach der Aufteilung leer sind und nicht das allzeit beste Individuum enthalten. Die vordefinierte Anzahl von Spezien kann dabei über- oder unterschritten werden und wird durch den Kompatibilitätskoeffizienten gesteuert. Existieren mehr Spezien als gefordert, wird dieser Koeffizient verringert, existieren weniger, wird er vergrößert. Bei einem Wert von 0 wird jedes Individuum zu einer eigenen Spezie.
In einer jeden Epoche werden sämtliche Individuen bis auf das beste erneuert. Dazu wird zuerst die Anzahl der Individuen einer Spezie anhand der relativen Fitness der Spezie so berechnet, dass über alle Spezien gemittelt die vordefinierte Bevölkerungsgröße erreicht wird:

$$\frac{\text{durchschnittliche Fitness der Spezie}}{\text{durchschnittliche Fitness aller Individuen}} \cdot \text{Bevölkerungsgröße}$$

Anschließend werden die vorhandenen Mitglieder einer Spezie nach Fitnesswert sortiert und nur ein vordefinierter Prozentsatz der Besten wird am Leben gelassen. Die Überlebenden werden dann benutzt, um so viele Nachfahren mittels Selektion, Mutation und Rekombination zu zeugen, wie die Spezie definierte Mitglieder besitzt.
Die Selektion der Eltern geschieht jeweils randomisiert, indem zufällig zweimal 2 Überlebende ausgewählt werden und der Beste gewinnt. Die Rekombination bevorzugt ältere Gene bzw. entscheidet zufällig bei gleichem Innovationsalter und übernimmt nur so viele Gene, wie der bessere Elternteil besitzt. Die Mutation erfolgt als letzter Schritt und umfasst sowohl das Verändern von Verbindungs- (Gewicht) und Knoteneigenschaften (Bias und Sigmoidkonstante) als auch das Hinzufügen von Verbindungen und Neuronen. Für jedes Gen des neuen Individuums, ob Verbindung oder Neuron, wird dessen Eigenschaft mit einer vordefinierten Wahrscheinlichkeit

[67] Die Umsetzung eines zu $x\%$ wahrscheinlichen Ereignisses erfolgt - wie üblich - in Form der Ziehung einer gleichverteilten Zufallsvariablen aus dem Einheitsintervall $[0, 1]$. Dabei tritt das Ereignis genau dann ein, wenn der gezogene Wert unterhalb der definierten Wahrscheinlichkeit x liegt. Ein Schwellenwert von 0.33 ergibt somit eine Wahrscheinlichkeit von 33%.

Pseudocode 3 Ablauf des NEAT4J-Algorithmus (Version 1.0), Teil 1

Funktion	**Beschreibung**
`createPopulation`	Starte die Erzeugung einer ersten Bevölkerung.
`.gaPopulationSize`	Hole die vordefinierte Bevölkerungsgröße.
`.createPopulation`	Erzeuge die einzelnen Individuen.
`..initialiseInnovations`	Erzeuge die initialen Netzwerke mit den vordefinierten Eigenschaften.
`..individualFromTemplate`	Transformiere die generischen Netzwerke zu NEAT4J-Objekten.
`.RunEpoch`	Schleife über die Epochen.
`..evaluatePopulation`	Berechne die Fitness der Individuen.
`...evaluate`	Evaluiere alle Individuen (Chromosomen).
`....createNetFromChromo`	Erzeuge NN aus den Chromosomen.
`....execute`	Werte die neuronale Fitnessfunktion aus.
`...updateFitness`	Aktualisiere alle Fitnesswerte.
`..runEvolutionCycle`	Führe einen Evolutionsschritt aus.
`...runEle`	Extinct Life Events: Erhöhe den Kompatibilitätskoeffizienten (Große Werte führen zu weniger, kleine Werte mehr Spezien.) um 5 und töte alle Individuen bis auf die vordefinierte Prozentzahl (wird nicht verwendet).
`...speciatePopulation`	Spalte die Bevölkerung in Spezien auf.
`....addSpecieMember`	Füge ein Individuum zu einer Spezie hinzu...
`.....isCompatable`	...wenn es zum Repräsentanten kompatibel ist, sonst erstelle neue Spezie.
`..findBestChromosome`	Ermittele das beste Individuum und markiere es.
`..removeExtinctSpecies`	Entferne überalterte Spezien.
`..updatePopulation`	Führe die Erneuerung der Bevölkerung durch.
`...spawn`	Erneuere die Bevölkerung.
`....shareFitness`	Berechne die Fitness jeder Spezie.
`.....adjustFitness`	Bevor- oder benachteilige die Fitness der Spezie je nach jung oder alt.
`.....ageFitness`	Berechne den durchschnittlichen Fitnesswert der Spezie.
`....totalAvSpeciesFitness`	Berechne den kumulierten Fitnesswert aller Spezien.
`....validSpecieList`	Validiere alle Spezien: Entferne ausgestorbene Spezien nur, wenn sie nicht den Champion enthalten.
`....calcSpecieOffspringCount`	Berechne die Größe einer neuen Spezie anhand ihrer relativen Fitness: $\frac{avgFitness}{totalFitness} \cdot popSize$
`....specieOffspring`	Erneuere die Spezie: Das beste Individuum bleibt und die Überlebenden zeugen Nachkommen.
`.....produceOffspring`	Generiere neue Individuen.
`......getSurvivalThreshold`	Nur eine vordefinierte Prozentzahl einer Spezie überlebt eine Epoche.
`......cloneChromosome`	Speichere den Besten der Spezie.
`......selectParents`	Selektiere Eltern: Wähle 2x von zwei zufälligen Individuen den Besten aus.
`......crossOver`	Führe ein XO durch: Verwende die jüngsten Gene des Besseren, bei gleichem Innovationsdatum entscheidet der Zufall (50 : 50), die Anzahl der Gene richtet sich nach dem besten Elternteil.
`......mutate`	Mutiere das neue Genom.
`.......mutateLink`	(p1 < pMutate ? (p2 < weightReplace ? rand(±1) : rand(±maxPertub)) und (p1 < pToggle ? deaktiviere bzw. aktiviere die Verbindung)

Pseudocode 4 Ablauf des NEAT4J-Algorithmus (Version 1.0), Teil 2

Funktion	Beschreibung
.....mutateNode	(p1 < pMutate) ? ((sigmoidFactor += rand(±maxPertub)) und (p2 < mutateBias ? (bias += rand(±0.1)))
.....mutateFeature	(p < pMutate) ? ... (wird nicht verwendet)
.....mutateAddLink	(p < pAddLink ? versuche maximal 5x, eine neue Verbindung einzufügen)
.....candidateNodes	Ermittele alle existierenden Knoten.
.....candidateLinks	Ermittele alle existierenden Verbindungen (aktivierte und deaktivierte).
.....linkIllegal	Prüfe, ob die neue, zufällig erzeugte Verbindung noch nicht existiert.
.....submitLinkInnovation	Trage die neue Verbindung in die Innovationsdatenbank ein.
.....updateChromosome	Aktualisiere das neue Chromosom / Genom.
.....mutateAddNode	(p < pAddLink ? füge neuen Knoten ein)
.....candidateLinks	Ermittele alle existierenden Verbindungen (aktivierte und deaktivierte).
.....submitNodeInnovation	Ersetze eine Verbindung durch einen neuen Knoten und Verbindungen von und zu diesem und trage ihn in die Innovationsdatenbank ein.
.....submitLinkInnovation	Trage die neuen Verbindungen in die Innovationsdatenbank ein.
.....findChosenIndex	Ermittele den Index der ersetzten Verbindung, setze den neuen Knoten dorthin und füge die beiden neuen Verbindungen ans Ende des Genoms an.
.....updateChromosome	Aktualisiere das neue Chromosom / Genom.
.....updateDepthInfo	Berechne die Tiefeninformation jedes Gens neu.
.....candidateNodes	Ermittele alle existierenden Knoten.
.....findOutputNodes	Finde die Ausgabeknoten.
.....assignNeuronDepth	Setze die neue Tiefe des Gens.
.....findSourceNodes	Finde alle eingehenden Verbindungen eines Knotens.
.....findNode	Finde einen Knoten.
.....ensureLegalLinks	Entferne rekurrente Verbindungen (falls diese verboten sind).
.....findNode	Finde einen Knoten.
.....updateChromosome	Aktualisiere das neue Chromosom / Genom.
....setExtinct	Werden keine Nachkommen erzeugt, markiere die Spezie als ausgestorben.
...validSpecieList	Validiere alle Spezien: Entferne ausgestorbene Spezien nur, wenn sie nicht den Champion enthalten.
...getCompatabilityChange	Hole den vordefinierten Wert der Kompatibilitätsveränderung.
...setThreshold	Erhöhe oder verringere den Kompatibilitätswert, wenn die Anzahl der Spezien nicht dem vordefinierten Wert entspricht

randomisiert verändert. Ebenso wird ihm mit einer vordefinierten Wahrscheinlichkeit eine neue Verbindung oder ein neues Neuron hinzugefügt, wobei das neue Neuron in eine bestehende Verbindung eingefügt wird[68]. Bestehende Verbindungen oder Neuronen können dabei nicht gelöscht, wohl aber deaktiviert, werden.
Nach einem Evolutionsschritt werden schließlich alle Individuen bis auf den Besten der Eltern getötet. Dieser darf dann als einziger unter den Individuen der nächsten Epoche weiterleben. Mit diesem Vorgehen wird zugleich die stochastische Suche sowohl in Struktur- als auch Gewichtsraum vollzogen.
Der gesamte evolutionäre Prozess wird nun wiederholt, bis eine Abbruchbedingung, wie z.B. eine Mindestfitness, eintritt. Der Beste aus allen Spezien der letzten Epoche bildet dann das Ergebnis.

6.3 EANT

Ein weiterer evolutionärer Algorithmus zur Optimierung von Struktur und Gewichten eines Neuronalen Netzwerkes ist der Algorithmus EANT (Evolutionary Acquisition of Neural Topologies). Er wird in Kassahun u. Sommer (2005); Kassahun (2006) entworfen und in Siebel u. Kassahun (2006) zu EANT2 erweitert. Seine Fortentwicklung erfolgt derzeit an der Christian-Albrechts-Universität zu Kiel, so dass momentan eine Entwicklerversion vorliegt.
EANT ist in der Programmiersprache C/C++ geschrieben und benutzt das in Abschnitt 3.1 vorgestellte LinearGenome als Datenstruktur für die Repräsentation eines Neuronalen Netzwerkes.
Zur Optimierung der Netzstruktur wird eine stochastische Suche eingesetzt, während zur Optimierung der Gewichte eine Blackboxoptimierung auf Basis des CMA-ES-Algorithmus zur Anwendung kommt. Genau betrachtet existieren zwei Varianten des Algorithmus, eine mittels PVM parallelisierte mit separater (numerischer) Gewichtsoptimierung und eine nicht parallelisierte mit integrierter (stochastischer) Gewichtsoptimierung. Letztere ist der historische Ursprung (EANT), der hier nicht weiter betrachtet wird und zur parallelisierten Variante (EANT2) weiterentwickelt wird.
Weiter unterstützt der Algorithmus die Analyse von Verwandtschaftsbeziehungen im evolutionären Prozess. Dafür existieren neben einer fortlaufenden globalen Identifikationsinstanz für jedes Individuum ein Selbst-, ein Eltern- und ein Ähnlichkeitsidentifikator. Stimmt letzterer mit dem Elternidentifikator überein, wurde das Netzwerk zuvor mutiert, ohne ein Neuron zu entfernen oder hinzuzufügen, und das Netz wird als ähnlich zu seinem Elternteil definiert. Stimmt der Ähnlichkeitsidentifikator hingegen mit dem Selbstidentifikator überein, wurde das Netzwerk zuvor neu erzeugt oder ihm ein Neuron entfernt oder hinzugefügt, und es wird als ähnlich zu sich selbst bezeichnet. Die Verwandtschaftsbeziehung in Form des Elternidentifikators wird ferner nicht vererbt, sondern entspricht stets dem Identifikator

[68] Das neue Neuron wird also initial mit einer eingehenden und einer ausgehenden Verbindung ausgestattet.

des direkten Vorfahren oder ist 0 sofern dieser nicht existiert.
Der Ablaufplan des EANT-Algorithmus ist in Pseudocode 5 und 6 zu sehen[69] und beginnt mit der Erzeugung einer initialen Bevölkerung, die sich aus voll oder halbvoll verbundenen Neuronalen Netzwerken mit oder ohne zufällig generierter verdeckter Schicht zusammensetzt.
Im ersten Schritt eines Generationszyklus werden die Gewichte PVM-basiert optimiert. Hierzu wird zwischen dem Modus A, in dem die Anzahl der Individuen verdoppelt, und dem Modus B, in dem die Anzahl verdreifacht wird, unterschieden. Modus B ist die Starteinstellung, und die Wahl des Modus wird nach jeder Gewichtsoptimierung anhand der erzielten Fitnessverbesserung ermittelt. Ist die Verbesserung kleiner als ein vordefinierter Prozentsatz wird Modus B gewählt, ansonsten A. Ferner kann zwischen randomisierten, nullgesetzten oder den aus einer vorherigen Optimierung bereits bekannten Startgewichten gewählt werden, wobei in Modus B in der zweiten Kopie alle Verbindungen zu Neuronen konstant und andere gleichverteilt reinitialisiert werden.
Nach erfolgter Gewichtsoptimierung werden nur so viele Kopien eines Individuums behalten, wie dies die maximale Bevölkerungsgröße zulässt, bevor anschließend anhand von Fitness und Ähnlichkeit weitere Individuen aussortiert werden. Die finale Selektion des ersten Schrittes überleben im Rahmen der vordefinierten Bevölkerungsgröße nur die Besten.
Im zweiten Schritt werden abermals zwei, nun jedoch aus struktureller Veränderung hervorgegangene Mutanten eines Individuums erzeugt. Dazu wird der Kopie eines Individuums entweder ein Neuron inklusive zufälliger eingehender und ausgehender Verbindungen (ein Teilnetzwerk) oder eine einzige Biasverbindung hinzugefügt oder eine Verbindung entfernt. Die Kopie eines Individuums wird jedoch nur in den Evolutionsprozess aufgenommen, wenn sie sich nach ihrer Mutation von seinem Elternteil unterscheidet. Trifft dies zu, so wird auch der Ähnlichkeitsidentifikator des Mutanten entsprechend gesetzt. Der zweite Schritt endet je nach Modus mit doppelter oder dreifacher Bevölkerungsgröße, die erst in der nächsten Generation nach der Gewichtsoptimierung wieder reduziert wird.
Der gesamte evolutionäre Prozess wird wiederholt, bis eine Abbruchbedingung wie z.B. ein anvisierter Fitnesswert erreicht wird. Das beste Individuum aus der letzten Generation bildet dann das Ergebnis. Es handelt sich bei EANT also um einen evolutionären Algorithmus, der ohne Rekombination nur mit Selektion und Mutationen auskommt, vergleiche Abschnitt 5.1.

6.4 Evolutionärer Baukasten

Ausgehend von statischer Netzstruktur und Gewichtsoptimierung unter Verwendung numerischer Minimierungsverfahren (wie im Backprop-Algorithmus oder einer Blackboxoptimierung) ist eine Erweiterung um die evolutionäre Optimierung

[69] Die Punkte in der Darstellung stehen für die Verschachtelungstiefe eines Methodenaufrufs. Etweilige Auffälligkeiten im Ablaufplan gehören - analog zum Pseudocode von NEAT - zu EANT hinzu.

Pseudocode 5 Ablauf des EANT-Algorithmus (Revision 326), Teil 1

Funktion	Beschreibung
`initialise_PVM`	Initialisiere die PVM und setze Gewichtsoptimierer.
`InitGeneration`	Setze die Seed-Werte (Uhrzeit) der Zufallszahlengeneratoren.
`SetNoOfInputs`	Setze die Anzahl der Eingabeneuronen.
`SetNoOfOutputs`	Setze die Anzahl der Ausgabeneuronen.
`SetNoOfStructures`	Setze die Bevölkerungsgröße (derzeit genau 30).
`GenerateFirstStructures`	Erzeuge die Bevölkerung: Versuche maximal 20x ein neues Individuum zu generieren oder nimm den letzten Versuch (den ersten bei vollverbundenen Netzwerken).
`.GenerateLinearGenome`	Erzeuge ein randomisert gewichtetes etwas mehr als halbvoll (oder voll) verbundes Individuum.
`.SetFitnessValue`	Evaluiere und speichere den Fitnesswert des Individuums.
`.AddRandomSubNetworkToNeuron`	Füge dem Individuum so viele zufällig erzeugte Teilnetze hinzu, wie verdeckte Neuronen existieren (hier keine).
`.OrderLinearGenome`	Ist das Individuum kein voll verbundenes Netzwerk...
`.RemoveDuplicates`	...sortiere die Gene und entferne doppelte.
`.SetAge`	Setze das Alter des Individuums auf 0.
`.(evolution loop)`	Schleife über die Generationen.
`.ExploitStructures`	Optimiere die Gewichte der Individuen PVM-basiert (oder lokal mit stochastischer Suche).
`..OptimizeWeightsUncorrelatedPVM`	Optimiere PVM-basiert die Gewichte.
`...Insert`	Fertige je nach Modus 2 oder 3 Kopien eines Individuums an.
`...(distribution loop)`	Verteile die Individuen zur Optimierung mittels PVM.
`....SetFitnessValue`	Evaluiere und speichere den Fitnesswert des Individuums.
`....ReinitialiseParametersRandomly`	Kopien mit geradem Index erhalten neue Startgewichte.
`....(pvm loop)`	PVM-Klienten warten auf Optimierungsaufgaben vom PVM-Server.
`.....ReadLinearGenome`	Benutze das empfangene Individuum (LinearGenome).
`.....get_weights`	Ermittele Gewichte und Sigmoidkonstanten (nur Gewichte).
`......OneSubNetworkForward`	Bestimme die Position des ersten Eintrags des nächsten Teilnetzwerkes (mit einem Ausgabeneuron als Wurzel).
`......GetSigmoidTimeConstant`	Ermittele die Sigmoidkonstante, falls gewünscht (nein).
`......GetWeight`	Ermittele das Gewicht eines Gens.
`.....GenerateBinaryGenome`	Erzeuge und benutze ein binäres Genom, falls gewünscht.
`.....CMA-ES_init`	Initialisiere den CMA-ES-Algorithmus.
`.....(optimization loop)`	Schleife über die Gewichtsoptimierung.
`......CMA-ES_SampleDistribution`	Erzeuge eine zufällige, neue CMA-ES-Bevölkerung.
`......CMA-ES_EvaluateSample`	Berechne die Fitnesswerte der CMA-ES-Bevölkerung.
`......CMA-ES_ReestimateDistribution`	Aktualisiere die CMA-ES-Suchregion.
`......CMA-ES_get`	Ermittele die benötigen Funktionsauswertungen.
`......fmin`	Ermittele den besten Allzeitfitnesswert.
`.....evaluate_negative_fitness`	Aktualisiere das beste Individuum.
`.....SaveLinearGenome`	Sende (XML) das optimierte Individuum zum PVM-Server.

Pseudocode 6 Ablauf des EANT-Algorithmus (Revision 326), Teil 2

Funktion	Beschreibung
`...RemoveDoubles`	Lösche die schlechten Kopien eines Individuums.
`...SortStructures`	Sortiere die verbleibenden Individuen nach Fitnesswert.
`...SpawnStructures`	Lösche Individuen mit identischem und ähnlichem Fitnesswert.
`...TrancatePopulation`	Lösche bis zur Bevölkerungsgröße alle bis auf die besten Individuen.
`..SetTryHard`	Wähle zwischen den Modi A und B.
`.SelectBestIndividuals`	Lösche bis zur Bevölkerungsgröße alle bis auf die besten Individuen.
`.ExploreNewStructures`	Erzeuge eine mutierte Bevölkerung.
`..OrderLinearGenome`	Sortiere die Gene der Individuen.
`..(structural mutation loops)`	Versuche max. 100-mal pro Individuum 2 neue zu erzeugen.
`...MutateGenome`	Kopiere und mutiere das Individuum.
`....AddRandomSubNetwork`	Füge mit 70% Wkt. ein neues Teilnetz hinzu.
`.....FindNeuron`	Finde ein Neuron, von welchem eine zufällig ausgewählte Verbindung zu einem Eingabeneuron ersetzt werden soll.
`.....EraseNode`	Lösche die Eingabeverbindung, sofern vorhanden.
`.....AddRandomSubNetworkToNeuron`	Füge dem Neuron ein neues Teilnetz als Eingabe hinzu.
`......RemoveCycles`	Teste auf und entferne gegebenenfalls Zykel.
`......GenerateNeuron`	Erzeuge ein Neuron bzw. Teilnetz mit etwas mehr als der Hälfte der Eingabeneuronen als Eingabe.
`......InsertNodes`	Füge dem ausgewählten Neuron das neue Teilnetz hinzu.
`......UpdateDepth`	Aktualisiere die Tiefe des neuen Neurons.
`......(connection loop)`	Füge das Teilnetz bei 40% der Neuronen mit größerer Tiefe als Eingabe hinzu, falls dadurch keine Zykel entstehen.
`.....RemoveCycles`	Entferne Zykel im Individuum.
`.....SetFitnessValue`	Evaluiere und speichere den Fitnesswert des Individuums.
`....RemoveRandomConnection`	Lösche mit 25% Wkt. zufällig eine Verbindung.
`.....RemoveConnection`	Entferne eine Verbindung mit Rücksicht auf den Gentyp.
`.....SetFitnessValue`	Evaluiere und speichere den Fitnesswert des Individuums.
`....AddRandomBias`	Füge mit 5% Wkt. eine zufällige Biasverbindung ein, sofern noch Neuronen ohne Bias existieren, berechne den Fitnesswert und speichere diesen beim neuen Individuum.
`.....SetNumberOfInputs`	Aktualisiere die Anzahl der eingehenden Verbindungen.
`.....SetFitnessValue`	Evaluiere und speichere den Fitnesswert des Individuums.
`....OrderLinearGenome`	Sortiere die Gene des Individuums.
`....RemoveDuplicates`	Entferne doppelte Gene.
`...CheckInnovation`	Unterscheidet sich der Mutant von seinem Elternteil?
`....SetFeatures`	Setze das Alter des neuen Individums auf 0 und aktualisiere den Ähnlichkeitsidentifikator.
`....push_back`	Füge das neue Individuum der Bevölkerung hinzu.

evolutionäre Operatoren		
Selektion	:	Auswahl der besten Netzwerke für die nachfolgende Mutation
Rekombination	:	Individuum erhält 90% der zufällig ausgewählten Verbindungen des Besten oder jede Verbindung des Besten zu einer vordefinierten Wahrscheinlichkeit
Mutation	:	Zufälliges Hinzufügen oder Entfernen einer maximalen Anzahl an (nicht-rekurrenten) Verbindungen, (verdeckten) Neuronen oder Teilnetzwerken

Tabelle 16: Evolutionäre Operatoren für neuronale Evolution

der Netzstruktur ein sehr naheliegender Gedanke. Diesem folgend wird ein Baukasten zur evolutionären Entwicklung neuronaler Topologien auf Grundlage folgender Komponenten erstellt:

- Datenstruktur: Kanonisches Netzwerk aus Abschnitt 3.2 oder Chaotisches Netz aus Abschnitt 3.3
- Separierung: Trennung zwischen den Gewichts- und Strukturoptimierung, wobei sich für erstere numerische Minimierungsverfahren bewährt haben, für letzte jedoch nur der evolutionäre Ansatz bekannt ist
- Gewichtsoptimierung: CMA-ES, SR, SDS, BFGS o.ä.
- Strukturoptimierung: Selektion, Rekombination, Mutation nach in Tabelle 16 beschriebener Art und Weise

Ebenso wie die Strukturentwicklung in Abschnitt 6.5 stellt der Baukasten einen Versuch dar, u.a. die Art und Weise einer evolutionären Entwicklung der Netzstruktur zu untersuchen. Sein Umfang ist übersichtlich, die Vielfalt der konstruierbaren Algorithmen aber sehr groß, da jede Komponente parameterisiert und vielfältig ausgestaltet werden kann. Der Pseudocode 7 zeigt den unmodifizierten Ablaufplan des Baukastens, der eine Art eine Obermenge der evolutionären Möglichkeiten zur Entwicklung neuronaler Topologien ergibt.

Balancierung

Zur Korrektheit bzw. Ausgewogenheit der benutzten evolutionären Operationen wird bemerkt, dass deren Analyse in Bezug auf Abschnitt 4.4 ein mögliches Ungleichgewicht zwischen dem Entfernen und dem Hinzufügen einer Verbindung aufzeigt. Dieses Problem ist darauf zurückzuführen, dass bei einer Mutation des Netzes eine Verbindung nicht immer gelöscht oder hinzugefügt werden kann.
Versucht ein (randomisierter) Algorithmus in einem Netzwerk eine neue Kante hinzuzufügen, die bereits existiert, misslingt dies. Versucht er eine Kante zu löschen, die nicht existiert, misslingt dies ebenfalls. Dies ist ein Phänomen, das mit Markovketten formalisiert werden kann und welches vielfältig in Modellen mit diskreten oder abzählbar unendlich vielen Zuständen auftritt.

Pseudocode 7 Ablaufplan des Baukastens

Funktion	**Beschreibung**
`setParameters`	Setze die Parameter des Algorithmus.
`setInitialNetwork`	Setze ein vordefiniertes Urspungsnetzwerk.
`setFitnessFunction`	Benutze den Visual Servoing-Test.
`.createInitialPopulation`	Jedes Individuum ist eine Kopie des initialen NN.
`.evaluatePopulation`	Berechne die Fitnesswerte.
`.(generation loop)`	Schleife bis Abbruch.
`.findBest`	Speichere den Besten.
`.(evolution loop)`	Führe einen Evolutionsschritt durch.
`..cloneBest`	Ersetze Individuum durch den Besten.
`..extendSearchSpace`	Falls gewünscht, füge 1 Neuron hinzu.
`..crossover`	Falls gewünscht, rekombiniere Besten und i-ten.
`..addLink`	Füge mit Wkt. $pl1$ maximal $l1$ Verbindung hinzu, ...
`...(checkCycle)`	...aber teste zuvor auf neue Zykel.
`..removeLink`	Entferne mit Wkt. $pl0$ maximal $l0$ Verbindungen.
`..addNode`	Füge mit Wkt. $pn1$ maximal $n1$ Neuronen hinzu.
`..removeNode`	Entferne mit Wkt. $pn0$ maximal $n0$ Neuronen.
`..addSubnetwork`	Füge mit Wkt. $ps1$ maximal $s1$ Teilnetze hinzu.
`...addNode`	Füge ein Neuron hinzu.
`...neuronsWithOutgoingLinks`	Füge randomisiert eingehende Verbindungen hinzu.
`...neuronsWithIncomingLinks`	Füge randomisiert ausgehende Verbindungen hinzu.
`..removeSubnetwork`	Entferne mit Wkt. $ps0$ maximal $s0$ Teilnetze.
`..isConnected`	Teste auf vorhandene Funktionalität des Netzes.
`..minimize`	Optimiere ggf. die Gewichte.
`..setWeights`	Setze ggf. die optimierten Gewichte.
`..evaluate`	Berechne ggf. den Fitnesswert.
`.findWorst`	Ersetze Schlechtesten durch vorherigen Besten.
`.findBest`	Finde den Besten.

Wird dieses Problem nicht erkannt, folgt eine verzerrte Suche, die den Zustandsraum nicht vollständig ausnutzt. Dies passiert beispielsweise, wenn zum Löschen einer Kante auf eine Liste existierender Kanten zurückgegriffen wird, wohingegen zum Hinzufügen einer Kante zwei Neuronen zufällig ausgewählt werden. Während dann das Löschen einer existierenden Kante immer gelingt, gilt dies nicht für das Hinzufügen einer Kante, wenn zwischen den Neuronen bereits eine existiert. In der Folge wird jede genügend lange Evolution auf dieser Grundlage fast nur Netzwerke ohne Kanten hervorbringen. Dieses Ungleichgewicht kann auch nicht durch die Wahl kleiner Wahrscheinlichkeiten für das Löschen und hoher Wahrscheinlichkeiten für das Hinzufügen einer Kante verhindert werden, sondern führt nur zu längeren Laufzeiten, weil Aktionen mit kleinen Wahrscheinlichkeiten seltener eintreten.

Wird das Problem hingegen erkannt, kann es zwar nicht gelöst, aber wenigstens durch die Verwendung derselben Methode zur Kantenwahl beim Löschen und Hinzufügen einer Kante ausbalanciert werden. Dazu empfiehlt es sich einfach, auch zum Löschen einer Kante, zufällig zwei Neuronen auszuwählen, und dann zu versuchen, die Kante zwischen ihnen zu löschen. Damit besteht dann verfahrenstechnische Chancengleichheit. In Netzwerken mit Mehrfachkanten ist diese Lösung für eine Balancierung der Operatoren jedoch nicht möglich.

6.5 Strukturentwicklung

Wie bereits in den Abschnitten 2.2.1 und 2.3.3 beschrieben, erzeugt ein Neuronales Netzwerk je Eingabevektor eine Informationsmenge[70]. Diese ist unter anderem abhängig von den Gewichten und der Struktur des Netzes. Es ist jedoch nicht klar, wie relavent die einzelnen Komponenten eines klassischen Netzes der zweiten Generation für die darin enthaltene Information sind. Um diese Frage zu beantworten, wird ein Netzwerk mit fixierten Gewichten und maximaler Struktur vorgeschlagen und ein Algorithmus[71] entwickelt, der prüft, ob für eine gegebene Aktivierungsfunktion aus allein der Netzstruktur ausreichend Information gewonnen werden kann, um ein Problem zu lösen.

Das Ziel einer neuronalen Struktur ist hier also die Reichhaltigkeit bzw. Vielfalt an Information bzw. Zuständen für eine Eingabe, und aufgrund der chaotischen Eigenschaft der neuronalen Funktion wird angenommen, dass eine solche Informationsmenge entstehen kann[72].

[70] Mit Blick auf die in Abschnitt 2.3.1 benannte Krylovsequenz kann diese für lineare neuronale Funktionen z.B. mit Hilfe der Dimension $d = \dim(\mathcal{K})$ des erzeugten Krylovraumes, für welche $d \leq m \leq n$ gilt, gemessen werden.

[71] Der Algorithmus erhält den Namen Tiger Search (TS), weil auf der Suche nach einer Lösung zum einen das Verfahren durch die unüberschaubare Informationsmenge des Neuronalen Netzes und zum anderen der Benutzer durch die überabzählbare Menge der Metaparameter (Propagierungs- und Aktivierungsfunktionen) tigert, wie dies inzwischen auch der Suche nach einem Tiger in freier Wildbahn entspricht.

[72] Vergleiche hierzu auch die VC-Dimension einer Funktionenklasse.

6.5.1 Interpretation

Ein ungewichtetes Neuronales Netzwerk (mit neuralen, 1-gesetzten Gewichten) ist als Zwischenstufe zwischen biologischen und künstlichen Netzen interpretierbar. Anstatt - wie in künstlichen Netzen der zweiten Generation - die Informationen über die Existenz eines Impulses und dessen Impulsstärke in einem einzigen reellwertigen Gewicht zu bündeln, wird nun allein der Aktivierungswert als Information über einen Impuls betrachtet und von etweiligen Attributen wie beispielsweise der Impulsstärke getrennt. Während für biologische Neuronale Netzwerke dabei eine binäre Informationskodierung (eine Nervenzelle feuert oder feuert nicht) bekannt ist, verallgemeinern künstliche Netze diese jedoch normalerweise auf vielwertige Kodierungen. Das Attribut der Impulsstärke kann dann davon getrennt durch eine Amplitude, eine Feuerfrequenz oder auch in Form einer Gewichtung kodiert werden[73]. Diese Trennung erscheint auch dem Prinzip der Einfachheit (KISS-Prinzip der Informatik) biologischer Komponenten gerechter zu werden, und ähnelt entfernt dem Ansatz der „Spiking Neurons“ in Gerstner u. Kistler (2002).
Der im Folgenden vorgestellte TS-Algorithmus nutzt diese Separierung. Seine Informationsgewinnung basiert auf der exponentiellen Netzstruktur, seine Informationsverarbeitung auf der Berechnung einer Linearkombination als aufgabenabhängiger Gewichtung der gewonnenen Informationsmenge.
Der Ansatz setzt dabei nicht zwingend spezielle Netzwerktypen voraus, sondern kann prinzipiell auf jedes Netz angewendet werden. Die benutzte maximale Netzstruktur ergibt sich lediglich als ein direkter Einstieg in die Suche nach einer geeigneten Struktur.

6.5.2 Netzstruktur

Für die Definition der maximalen Netzstruktur werden Propagierungs- und Aktivierungsfunktion (weiterhin) als Metaparameter betrachtet und als exogen angenommen. Damit verbleibt als einziger Parameter eines ungewichteten Neuronalen Netzwerkes mit maximaler Struktur dessen Tiefe. Diese wird als die maximale Pfadlänge zu einem Knoten ausgehend von der Eingabeschicht definiert, vergleiche Kapitel 2. Für eine fixierte Tiefe $k \in \mathbb{N}$ können dann alle Kombinationen von Verbindungen zu einem Neuron bestimmt werden. Dies sind maximal exponential viele, denn für $n_0 \in \mathbb{N}$ Eingabeneuronen (das Biasneuron zählt dabei ebenfalls als ein, wenn auch nur indirektes, Eingabeneuron) erhält man genau $n_1 := \binom{n_0}{1} + \binom{n_0}{2} + \ldots + \binom{n_0}{n_0-1} + \binom{n_0}{n_0}$ mögliche Kombinationen mit Tiefe 1 und rekursiv fortgesetzt ergibt sich für Tiefe k die Rekursionsformel:

$$n_k := \binom{n_{k-1}}{1} + \binom{n_{k-1}}{2} + \ldots + \binom{n_{k-1}}{n_{k-1}-1} + \binom{n_{k-1}}{n_{k-1}} \tag{11}$$

Wird die maximale Anzahl eingehender Verbindungen für ein Neuron auf maximal $m \in \mathbb{N}$ beschränkt, so reduziert sich die Gleichung (11) auf, die etwas handlichere

[73] Die Verbindungsgewichte in klassischen Netzen der zweiten Generation könnten z.B. als eine einfache Kodierung der Impulsstärke interpretiert werden.

Form:

$$n_k := \binom{\min(m, n_{k-1})}{1} + \binom{\min(m, n_{k-1})}{2} + \ldots + \min(m, n_{k-1}) + 1 \qquad (12)$$

Die Tabellen 33 und 34 im Anhang zeigen, dass sich auch noch für die beschränkte Gleichung (12) pro 10 weitere Verbindungen (additiv) die Anzahl sowohl der Neuronen als auch der Verbindungen um den Faktor 1000 (multiplikativ) erhöht. Dabei existieren stets ungefähr zehnmal mehr Verbindungen als Neuronen.

6.5.3 Algorithmus TS

Der Algorithmus besteht aus drei Stufen und wird als Referenz mit Hilfe der Broccoli-Umgebung[74] in der Programmiersprache Java implementiert. Die erste erzeugt ein exponentielles Netzwerk für eine vordefinierte Neuronenanzahl, das auf Grundlage des kanonischen Netzwerkes aus Kapitel 3 realisiert wird. Statt der Vorgabe einer bestimmten Tiefe wird dazu die rekursive Konstruktion so lange fortgesetzt, bis die gewünschte Anzahl an Neuronen erreicht ist. Die zweite Stufe gewinnt dann aus dem Datensatz alle auf Grundlage des Netzes zugänglichen Informationen, und die dritte erzeugt daraus einen problemabhängigen Lösungsvorschlag. Die Konstruktion der Netzwerke ist dabei prinzipiell austauschbar, erfolgt aber vorerst nur auf Grundlage der maximalen Netzstruktur.

Die Informationsgewinnung erfolgt unabhängig von der vorgeschlagenen exponentiellen Netzstruktur und läßt sich prinzipiell auch auf gewichtete Netzwerke anwenden. Zur besseren Verwendbarkeit der netzinhärenten Informationsmenge wird eine Matrixdarstellung gewählt.
Dazu erhält das Netzwerk nacheinander alle Eingabevektoren eines Datensatzes vorgelegt, und für jedes Neuron wird in einer Matrix A, deren Zeilen den Eingabevektoren und deren Spalten den verdeckten und den Eingabeneuronen entsprechen, der jeweilige Aktivierungswert gespeichert. Die Matrix enthält somit sämtliche aus den Eingabevektoren gewonnene Information, welche durch die Netzstruktur aufgedeckt wird.
Ferner könnte eine Konditionszahl der Matrix A oder eine Entropie als Maß der enthaltenen bzw. kodierten Information definiert werden.

Die Informationsverarbeitung erfolgt nun für eine gegebene Lernaufgabe über die bei der Informationsgewinnung unbedeutende Ausgabeschicht anhand einer (u.U. affinen) Linearkombination aus den Knoten der Eingabe- und der verdeckten Schicht. Diese ergibt sich analog zu einer Linearen Regression für jedes (vergleiche Gleichung 13) Neuron der Ausgabeschicht als die Lösung des Gleichungssystems

[74] Die Software Broccoli wird maßgeblich vom Autoren entwickelt und ist neben einer Anwendung zur statistischen Aufbereitung, Analyse und Darstellung numerischer Daten, eine Bibliothek für Algorithmen der Mathematik, Informatik und angrenzender Wissenschaftszweige.

$Ax = b$ mit x als Gewichtsvektor und b als Sollausgabevektor des Ausgabeneurons für alle Datenpunkte. Da dieses Gleichungssystem oft unter- teilweise aber auch voll- oder überbestimmt ist, bietet sich als Solver z.B. das Verfahren der Konjugierten Gradienten[75] für Normalengleichungen (CGNE) von Hestenes u. Stiefel (1952) an[76]. Dessen Laufzeit dient glechzeitig als untere Abschätzung der Gesamtlaufzeit und sollte in $O(n \cdot m)$ realisierbar sein, wobei n die Größe des Netzwerkes und m die Größe des Datensatzes darstellen.

6.5.4 Varianten

Die beschriebene Vorgehensweise der Tigersuche erfolgte bislang nur für Überwachtes Lernen als Offlineverfahren. Eine Erweiterung von diesem zum Onlineverfahren ist jedoch möglich, indem z.B. analog zum Adaline aus Widrow u. Hoff (1960) eine Rückkopplung des Fehlers an der Ausgabeschicht vorgenommen wird. Aber auch ein modifizierter Backprop Algorithmus ist denkbar.
Für das Lernparadigma des Verstärkenden Lernens bieten sich zudem zwei Varianten an. Die erste benutzt eine h-dimensionale Fehlerfunktion

$$h = (|V_{\text{OUTPUT}}| \cdot (|V_{\text{INPUT}}| + |V_{\text{HIDDEN}}|)) \ , \tag{13}$$

in der die Koeffizienten der Linearkombination als Parameter einer Blackbox verwendet werden. Damit stellt die erste Variante ein zumeist hochdimensionales (Blackbox-) Optimierungsproblem auf.
Die zweite Möglichkeit ist die in Abschnitt 7.2 zur Berechnung der optimalen Lösung des Benchmarks eingesetzte. Dazu wird das Problem in so viele $|V_{\text{OUTPUT}}|$-dimensionale Teilprobleme separiert, wie Datenpunkte existieren (Größe des Datensatzes) und mit Verstärkendem Lernen für jede Teilaufgabe jeweils die optimale Netzausgabe bestimmt. Diese Sollausgaben können dann offline (anschließend) über alle berechneten Daten oder online (sofort) für jedes einzelne Datum mit Überwachtem Lernen erlernt werden. Dabei wird das Netz jedoch nur noch zur Memorisierung bzw. Generalisierung des Erlernten und nicht zum Lernen selbst gebraucht.

6.5.5 Metaproblem

Auch bei TS besteht das generelle Informationsproblem darin, eine optimale Matrix A zu finden, welche eine gestellte Aufgabe löst und dabei perfekt generalisiert, also die latente Funktion, deren Abtastung der Datensatz darstellt, perfekt approximiert. Dazu sollte das Netzwerk optimalerweise so beschaffen sein, dass keine zwei Knoten das Gleiche tun bzw. keine Redundanz in der verdeckten Schicht auftritt, die u.a. zu einer schlechten Konditionierung von A führt. Ebenfalls könnten auch bestimmte Verbindungen bzw. Kombinationen weggelassen werden.

[75] Nicht zu verwechseln mit dem namensgleichen Minimierungsverfahren für nichtlineare Funktionen.

[76] Der bekannte GMRES-Algorithmus von Saad u. Schultz (1986) wird aus Gründen der Laufzeit nicht verwendet, da dieser in der k-ten Iteration stets k Vektoren abspeichern muss und dies auch durch bedingte Neustarts nicht ausreichend kompensiert werden kann.

Dieses Problem ist in Abschnitt 9.5 Gegenstand einer Untersuchung.
Ferner ist unklar, welche Verschachtelungen bzw. welche Kombinationen von Verschachtelungen viel Information beisteuern und welche nicht. Hier bietet sich womöglich ein evolutionärer Algorithmus an, der im Raum der Funktionsverschachtelungen operiert.

6.6 Diskussion

Als erstes fällt der Anspruch der betrachteten evolutionären Algorithmen, ein Pendant zur biologischen Evolution bzgl. Neuronaler Netzwerke zu sein, auf. Dieser erscheint hauptsächlich aufgrund des Komplexitätunterschiedes ungerechtfertigt, wenn auch im Grundsatz nachvollziehbar. Dabei ist vor allem die häufig genutzte, linear kodierte Darstellung eines Individuums aus zwei Gründen unangebracht. Zum einen verarbeitet die Natur ein Genom vermutlich anders (nichtlinear) als ein evolutionärer Algorithmus, zum anderen besitzt eine lineare Datenstruktur (siehe Kapitel 3) kein optimales Laufzeitverhalten für die typischen Operationen auf einem Neuronalen Netzwerk. Weiter erzeugt die auf ein Gütekriterium ausgerichtete Konstruktion eines künstlichen neuronalen Netzes lediglich ein spezielles Modell, während biologische Netzwerke Lösungsvorschläge für verschiedene Aufgabenstellungen auf Grundlage derselben Basis zustande bringen.
Ungeachtet dieser Diskrepanzen, zeigt sich ein relativ überschaubares Bild der verwendeten Möglichkeiten zur Konstruktion eines neuronalen Netzes. Die Analyse enthüllt dennoch oft eine unausgereifte bzw. unbegründete Vorgehensweise der eingesetzten Operatoren, die eine ausgewogene Suche im neuronalen Parameterraum verhindert. Ob dies im Endeffekt jedoch eine signifikante Auswirkung auf eine Lösung hat, bleibt auch wegen der Größe des Suchraumes vorerst unbeantwortet. Aufgrund der chaotischen Eigenschaft einer neuronalen Funktion und der numerischen Probleme beim Umgang mit dieser ist zu erwarten, dass ein Verfahren eher an dieser Hürde scheitern wird als an der Strukturentwicklung. Voraussichtlich wird damit erst eine direkte Konstruktionsvorschrift, die Beherrschung chaotischer bzw. schlecht konditionierter Funktionen durch ein Optimierungsverfahren oder ein anderes geeignetes Suchverfahren eine allgemein Verwendbarkeit von Neuronalen Netzwerken ermöglichen.

7 Benchmark

In Siebel u. a. (2007) wird ein neuer Benchmark bzw. Test für Verstärkendes Lernen vorgeschlagen, welcher auf einer simulierten Robotersteuerung, z.B. des in Abbildung 5 gezeigten Roboters, beruht und hier noch einmal zusammengefasst wird.

Abbildung 5: Industrieroboter

Ein Roboter, der aus einer räumlich mobilen Kamera besteht[77], soll sich zu einem bestimmen Objekt hin bewegen, welches sich bereits in seinem Blickfeld befindet. Die Testaufgabe besteht in der Minimierung des räumlichen Abstandes in einem einzigen Schritt. Dazu wird bemerkt, dass diese Aufgabe grundsätzlich mit neuronalen Netzwerken ohne Biasverbindungen bzw. Biasneuron gelöst werden kann. Die Angabe der Güte einer Lösung erfolgt o.B.d.A. negativ, so dass höhere Werte besser sind. Das Ziel ist dabei aber vorzeichenneutral stets die Null.
Eine effektive Lösung dieser Aufgabe existiert zwar bereits in Form der iterativen (mehrschrittigen) Gauss-Newton-Methode[78]. Diese muss jedoch für jede Veränderung in Start- oder Zielposition erneut berechnet werden. Ein Neuronales Netzwerk hingegen ist prinzipiell im Stande, Zusammenhänge dergestalt zu erlernen, dass es nach optimalem Training alle Aufgaben dieses Typs löst.
Es wird sich zeigen, dass diese effiziente (einschrittige) Lösung der Aufgabe für Neuronale Netzwerke ausreichend schwierig zu lernen ist, um als Benchmark zu dienen. Gerade im Kontrast zu bekannten Benchmarks wie dem XOR- oder dem (vermeintlich schwierigen) Double-Pole-Balancing-Problem erfordert diese Aufgabe sogar schon für Approximationen der Lösungen komplexe Neuronale Netzwerke und unterscheidet sich damit essentiell von den sehr einfachen Lösungen anderer Benchmarks.

7.1 Details

Die Aufgabe wird in Koordinatenangaben bezüglich des Kamerafeldes des Roboters definiert. Sei $i \in \mathbb{N}_{\leq 4}$. Das anzufahrende Ziel besteht aus vier 2-dimensionalen Objektmarkierungen $p_i^* = (x_i^*, y_i^*) \in \mathbb{R}^2$ auf einem fest stehenden Objekt, welches so positioniert ist, dass sich die Punkte vollständig im Kamerabild des Roboters befinden. Eine Zielmarkierung aus ebenso vielen Punkten in gleicher (geometrischer) Anordnung im Kamerabild gibt die gewünschte Position des Ziels für den Roboter an. Der zu minimierende Fehler wird dann bezüglich des Kamerabildes durch den Abstand der vier Zielmarkierungen $p_i = (x_{i,}, y_i) \in \mathbb{R}^2$ zu ihren Pendants auf dem

[77] Die Kamera kann z.B. am Endeffektor eines Roboterarms montiert sein.
[78] Der Ansatz ist in diesem Kontext auch als Image-Jacobian bekannt.

Eingabe	**Beschreibung**
$\Delta x_1 = \|x_1^* - x_1\|$	Abstand zwischen Zielmarkierung x_1 und Objektmarkierung x_1^* im Kamerabild.
$\Delta y_1 = \|y_1^* - y_1\|$	Abstand zwischen Zielmarkierung y_1 und Objektmarkierung y_1^* im Kamerabild.
$\Delta x_2 = \|x_2^* - x_2\|$	Abstand zwischen Zielmarkierung x_2 und Objektmarkierung x_2^* im Kamerabild.
$\Delta y_2 = \|y_2^* - y_2\|$	Abstand zwischen Zielmarkierung y_2 und Objektmarkierung y_2^* im Kamerabild.
$\Delta x_3 = \|x_3^* - x_3\|$	Abstand zwischen Zielmarkierung x_3 und Objektmarkierung x_3^* im Kamerabild.
$\Delta y_3 = \|y_3^* - y_3\|$	Abstand zwischen Zielmarkierung y_3 und Objektmarkierung y_3^* im Kamerabild.
$\Delta x_4 = \|x_4^* - x_4\|$	Abstand zwischen Zielmarkierung x_4 und Objektmarkierung x_4^* im Kamerabild.
$\Delta y_4 = \|y_4^* - y_4\|$	Abstand zwischen Zielmarkierung y_4 und Objektmarkierung y_4^* im Kamerabild.
$\|\|p_1 - p_3\|\|_2^2 = (x_1 - x_3)^2 + (y_1 - y_3)^2$	Länge der Kreuzdiagonale zwischen den Markierungspunkten p_1 und p_3 im Kamerabild.
$\|\|p_2 - p_4\|\|_2^2 = (x_2 - x_4)^2 + (y_2 - y_4)^2$	Länge der Kreuzdiagonale zwischen den Markierungspunkten p_2 und p_4 im Kamerabild.

Tabelle 17: Eingabedaten bzw. Informationsmenge des Benchmarks

Objekt definiert. Ein Fehler des Roboters von $0 \in \mathbb{R}$ löst also die Aufgabe, welche darin besteht, im Kamerabild die vier Zielmarkierungspunkte mit den vier Objektmarkierungspunkten zur Deckung zu bringen. Wie für Verstärkendes Lernen üblich, wird die Aufgabe als Blackboxfunktion mit dem Fehler als Ausgabe modelliert.

Die neuronale Steuerlogik, welche dem Roboter vorgeschaltet ist, erhält die in Tabelle 17 angegebenen Eingaben. Aus diesen 10 Eingaben muss sie 3 Ausgabewerte berechnen, welche als räumliche Bewegung (Δx, Δy, Δz) zu interpretieren sind. Werden in der Ausgabeschicht Neuronen mit beschränktem Bildraum verwendet, so muss die Ausgabe des Netzwerkes zudem skaliert werden. Dies trifft z.B. für ein Netzwerk mit homogen auf $[-1, 1]$ beschränkten $(< \cdot, \cdot >, \tanh)$-Ausgabeneuronen zu. Der Roboter, der durch das Netz gesteuert wird, bewegt sich dann entsprechend dieses Vektors.

Die Aufgabe kann online einzeln für jedes Paar aus Start- und Zielposition oder offline über den Datensatz dieser Paare erfolgen, vergleiche Kapitel 6. Der hier vorgestellte Benchmark soll jedoch vornehmlich als Offlinevariante eingesetzt werden und wird dementsprechend präsentiert.

7.1.1 Bewertung

Sei $p^* = \{p_1^*, p_2^*, p_3^*, p_4^*\} \in \mathbb{R}^{(4\cdot 2)}$ die Position des anzufahrenden Objektes im Kamerabild, und sei $p = \{p_1, p_2, p_3, p_4\} \in \mathbb{R}^{(4\cdot 2)}$ dessen Zielmarkierung im Kamerabild. Bezüglich des Kamerabildes ist somit p^* variabel und p ist fixiert. Die Anordnung bzw. Zuordnung der Koordinaten innerhalb der Vektoren p und p^* sei in beiden Vektoren o.B.d.A. identisch. Für eine räumliche Position des Roboters wird sein Fehler formal als die Summe der quadratischen Abstände gleich indizierter Markierungen definiert:

$$||p^* - p||_2^2 = \sum_{i=1}^{8} (p_i^* - p_i)^2$$

Zusätzlich wird eine Bestrafungsfunktion $b(p_i^*) \geq 0$ für Situationen, in denen Objektmarkierungen nicht mehr im Kamerabild sichtbar sind, eingeführt[79]. Sei $w \in \mathbb{R}^t$, $t \in \mathbb{N}$, der Parametervektor des Neuronalen Netzwerkes. Für eine gegebene Zielmarkierung p und eine räumliche Roboterposition $s \in \mathbb{R}^3$ lautet dann die formale Fehlerfunktion $err(w)$: im Kamerabild

$$err(w;\, s, p) := ||p^*\,(s, w) - p||_2^2 + b(p^*)$$

Für eine Ausführung ergibt sich p^* dabei in Abhängigkeit von einer gültigen[80] räumlichen Startposition s des Roboters und der Ausgabe w des steuernden neuronalen Netzwerkes. Ziel dieses Benchmarks ist es, den Fehlerwert 0 zu erreichen.

7.1.2 Ablauf

Sei $S \subset \mathbb{R}^3$ eine Menge von räumlichen Startpositionen des Roboters, sei $P \subset \mathbb{R}^{(4\cdot 2)}$ eine Menge von Zielmarkierungen, sei $T := \{(t_0, t_1) \in S \times P :\, t_0 \in S \,\wedge\, t_1 \in P\}$ die Trainingsmenge, und sei $t \in T$. Damit der Test eine gute Generalisierung des Neuronales Netzwerkes ermöglichen kann, wird eine genügend große und geeignet zusammengesetzte Trainingsmenge gewählt. Zudem verwendet der Benchmark ein Lochkameramodell mit $\frac{8}{3}$mm $\times$ 2mm CCD-Sensor und 6.5mm Focus, um Realitätsnähe zu einer handelsüblichen Kamera am Endeffektor des Roboterarms (siehe Abbildung 5) zu simulieren, wozu je nach Wahl der Ausgabeneuronen eine Skalierung der Netzausgabe um Faktor 400 notwendig ist.
Zur Testdurchführung wird für jedes Trainingsbeispiel der Fehler berechnet und der Gesamtfehler aufsummiert. Als Testergebnis wird das arithmetische Mittel über den kumulierten Gesamtfehler definiert, so dass sich die bedingte Fehlerfunktion (Fitnessfunktion) $f(w\,;\, t)$ des Benchmarks wie folgt darstellt:

$$f(w;\, t) := \frac{1}{N} \sum_{n=1}^{N} err(w;\, t_{i,(0,\cdot)}, t_{i,(1,\cdot)})$$

[79] Weitere Nebenbedingungen wie z.B. die Minimalität der Netzstruktur dürfen ebenso in $b(p_i)$ eingehen.
[80] Eine Startposition heißt gültig, wenn alle Objektmarkierungen im Kamerabild zu sehen sind.

Dimension	Eigenwert
0	0.0
1	0.0
2	0.0
3	0.0
4	0.0
5	0.0
6	0.03632
7	0.13389
8	0.70946
9	1.69895

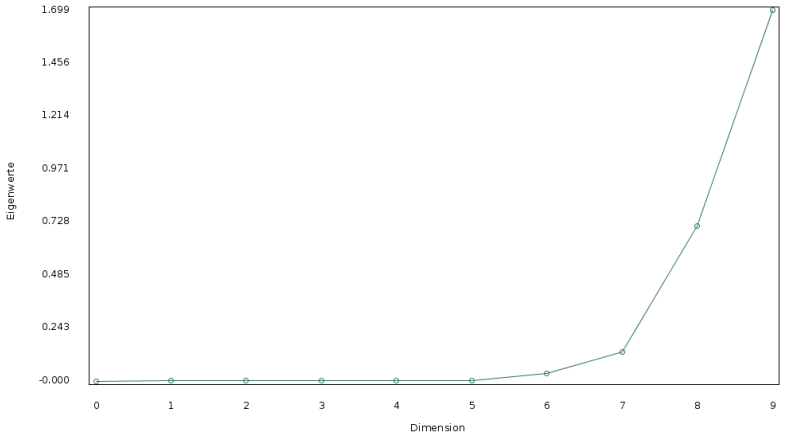


Abbildung 6: Eigenwerte nach PCA des dritten Datensatzes

Ergänzende Verfahren wie eine Vorverarbeitung der Eingabedaten, Boosting oder Gütekriterien wie Kreuzvalidierung sind bislang nicht explizit Teil des Tests, sondern Optionen.
Für die Durchführung der Algorithmenvergleiche in Kapitel 9 wird $|S| = 1023$ gewählt und jeder Startposition eine von $|P^*| = 29$ verschiedenen Objektmarkierungen zugeteilt. Der verwendete Datensatz bzw. die verwendeten Paare aus Startposition und Zielmarkierung befinden sich auf dem Datenmedium (Seite 150) oder können dem Quellcode entnommen werden.

7.2 Eigenschaften und Lösbarkeit

Die Existenz einer optimalen Lösung mit einem Fehler von 0, ist relativ leicht theoretisch einzusehen. Eine optimale Lösung für jedes Datum des Datensatzes wird mit Hilfe des BFGS-Algorithmus ermittelt und befindet sich ebenfalls auf dem Datenmedium im Anhang.
Ferner zeigt die Hauptkomponentenanalyse der Eingabedaten in Abbildung 6, dass die intrinsische Dimension ziemlich genau 4 ist und damit 6 Dimensionen eingespart werden können. Aufgrund der vollbesetzten Transformationsmatrix kann jedoch keine exakte Interpretation der 4 intrinsischen Dimensionen gefunden werden. Die Information der transformierten Daten besteht aber vermutlich aus den x- und y-Koordinaten sowie die beiden Kreuzdiagonalen. Obwohl eine solche Vorverarbeitung prinzipiell möglich ist, wird im Folgenden dennoch keine derartige Transformation der Eingabedaten benutzt.

7.3 Interpretation

Betrachtet man die oben definierte Eingabe des Neuronalen Netzwerkes, so kann dessen Aufgabe als Lernen einer dreidimensionalen Handlung auf Grundlage zwei-

dimensionaler Daten interpretiert werden. Ein Blick auf die Eingabedaten in Tabelle 17 zeigt, dass dies keine triviale Aufgabe ist. So muss das Neuronale Netzwerk aus den zur Verfügung gestellten zweidimensionalen Längen- bzw. Abstandsdaten eine geeignete dreidimensionale Bewegung erlernen. Die Längen der beiden Kreuzdiagonalen bilden dabei die räumliche Tiefeninformation.
Diese Sichtweise beschreibt die Aufgabe also als Lernen einer z-Bewegung aus gegebenen x- und y-Koordinaten und Maßen.

7.4 Erweiterungen

Die Testaufgabe läßt sich modifizieren bzw. verkomplizieren, indem zusätzlich zu den Bewegungen in x-, y- und z-Richtung Drehungen in yaw, pitch und roll ermöglich werden. Entsprechend muss das Neuronale Netzwerk dann die 6 Werte

$$\Delta x, \Delta y, \Delta z, \Delta\text{yaw}, \Delta\text{pitch}, \Delta\text{roll}$$

liefern. Zusätzlich lassen sich natürlich auch die Eingabedaten modifizieren, so dass z.B. die Diagonallängen durch Breite und Höhe des durch die Markierungen aufgespannten Rechtecks ersetzt werden.
Derartige Variationen des Benchmarks bieten vielfältige Möglichkeiten, weitere Probleme mit weit höherer Komplexität zu formulieren.

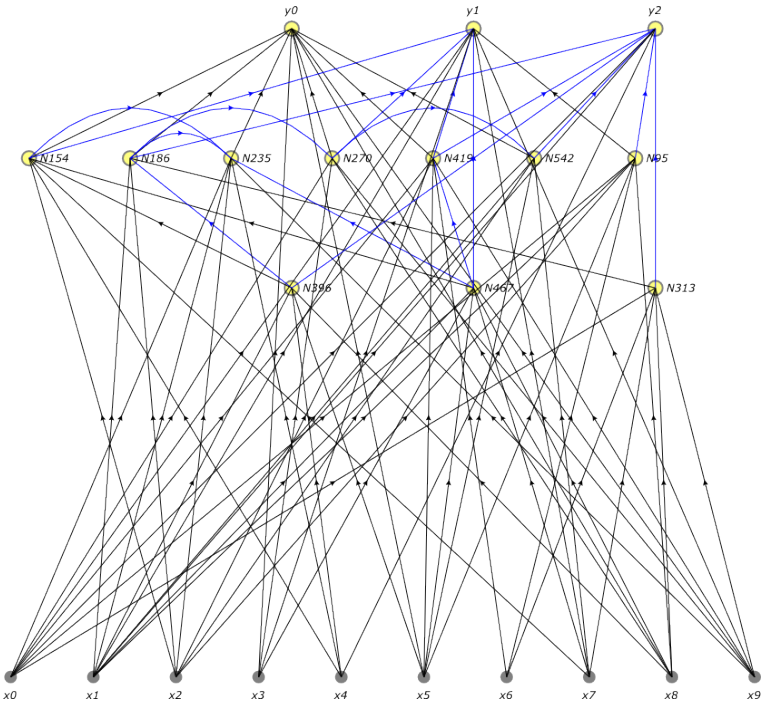


Abbildung 7: Neuronales Netzwerk mit fester Struktur

8 Optimierungsergebnisse

Auf Grundlage des Benchmarks aus Kapitel 7 in seiner Offlinevariante werden die Gewichte eines exemplarischen Neuronalen Netzwerkes[81] mit fester Struktur optimiert, das in Abbildung 7 veranschaulicht ist. Der Testaufbau gleicht dabei dem klassischen Vorgehen bei der Optimierung eines neuronalen Netzes.
Die herangezogenen (stochastischen) Verfahren für Verstärkendes Lernen sind ausschließlich blackboxbasiert (siehe Abschnitt 6.1.2). Ihre Aufgabe besteht darin, eine Funktionsoptimierung umgesetzt als Minimierungsaufgabe mit den Gewichten als Parametervektor und der Ausgabe der Benchmarkfunktion als Informationskriterium durchzuführen. Dazu werden verschiedene Parameterkombinationen getestet, wobei das verfolgte Ziel die Ermittlung von Standardeinstellungen für den Einsatz eines Verfahrens innerhalb des EANT-Algorithmus ist. Die Anzahl der vom Benutzer zwingend zu wählenden Parameter soll ferner minimiert werden, denn eine parameterfreie Version eines Algorithmus stellt dessen anvisierte größtmögliche Generalisierung dar.
Ohne Berücksichtigung der speziellen Parameter eines Verfahrens sind dafür bereits die Startwerte bzw. initialen Gewichte eine wesentliche Wahlmöglichkeit. Deren Nullsetzen kann dabei bayesianisch als Situation, in der keine A-priori-Information

[81] Das verwendete Netzwerk ist ein Ergebnis des EANT-Algorithmus angewendet auf das Visual Servoing-Problem. Sein Gewichtsvektor umfasst 82 Einträge (85 inklusive der optionalen Ausgabegewichte). Es ist bislang ein Fitnesswert von -0.222577 bekannt.

über ein sinnvolles Vorgehen existiert, interpretiert werden, so dass diese Parameter ihren späteren Wert auf eine geeignete Art und Weise während der Optimierung erlernen.
Die stochastischen Eigenschaften der betrachteten Algorithmen verlangen umfangreiche Versuchsreihen, die wegen hoher Laufzeiten selten so umfangreich sind, dass klare (statistische) Aussagen getroffen werden können. Die Auswertung der in den Testreihen gewonnenen Daten, erfolgt graphisch[82] und statistisch[83] in Form von Paneldatensätzen ($\{\text{Parameter } i\} \times \{\text{Parameter } j\}$). Auf eine weiterführende statistische Analyse[84] mit mehr als zweidimensionalen Abhängigkeiten wird vorerst verzichtet.
Zur Anwendung kommen in der Programmiersprache C implementierte und mit dem Compiler GCC (Version 4.1.0) unter Linux (SuSE, 32bit, i686) übersetzte[85] und auf Pentium 4 Rechnern mit mindestens 3Ghz benutzte Versionen der Algorithmen. Zudem wird stets (c.p.) dasselbe Testumfeld benutzt.

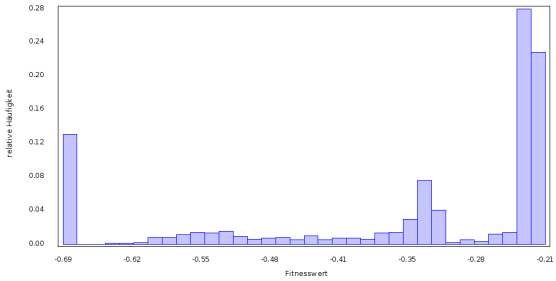
8.1 Test: CMA-ES

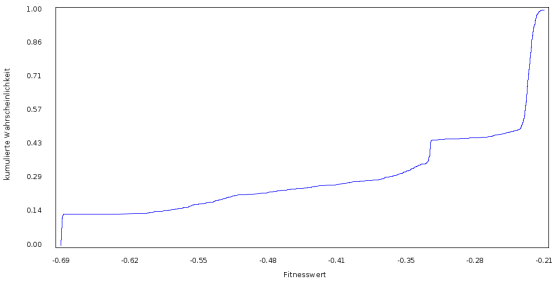
Als Referenz wird die Gewichtsoptimierung des Neuronales Netzwerkes aus Abbildung 7 mit dem CMA-ES-Algorithmus (genauer der in der Programmiersprache C geschriebenen Variante in Version 2.28), vorgestellt in Abschnitt 5.2, getestet.
Der beste Versuch[86] erbringt dabei einen Fitnesswert von -0.20756. Abbildung 8 zeigt Histogramm, CDF und Statistiken der resultierten Fitnesswerte. Die präsentierten Ergebnisse sind unbereinigt.
Das ungewöhnliche Aussehen des Histogramms mit den vielen schlechten Fitnesswerten um -0.69 ergibt sich wider Erwarten nicht durch außergewöhnliche Abbrüche des Algorithmus, die ebenfalls öfters aufgrund zu langer Stagnation in der Fitness oder numerischer Pwurdrobleme bei der Verarbeitung der Kovarianzmatrix zu beobachten sind, sondern durch schlechte Parameterwahlen bzw. Eigenheiten von CMA-ES, siehe unten.

8.1.1 Startwerte und Parameter

Sofern die im Folgenden untersuchten Parameter nicht explizit vorgegeben werden, sind diese optional, weil der CMA-ES-Algorithmus ein parameterfreies stochastisches Verfahren ist, das aus einer gegebenen Problemdimension heuristische Parameterwerte standardisiert ableitet.

[82] Es werden 1- und 2-dimensionale Scatter-, Grid- und Box-&-Whisker-Plots benutzt.
[83] Es werden die typischen Statistiken wie Mittelwert, Standardabweichung, Kovarianz, Minimalwert, 25%-Quantil, Median, 75%-Quantil, Maximalwert und Interquantilabstand verwendet.
[84] Für eine genauere Untersuchung kommen u.a. Kointegrationstests, ANOVA, Kontingenztafeln, VARMAX-Modelle und insbesondere die Faktorenanalyse usw. in Frage.
[85] Für den Compileraufruf zum Übersetzen der Algorithmen werden die folgenden Parameter verwendet: -O3 -march=pentium4 -mfpmath=sse -mmmx -msse -msse2 (-msse3)
[86] Der Test mit dem besten Ergebnis verwendet die Parameter $\sigma_0 = 0.5$, kleine zufällige initiale Gewichte und 170 Individuen.





Mittelwert	−0.3458	beschnittener Mittelwert 1%	−0.3437
Varianz	0.0283	beschnittener Mittelwert 3%	−0.3392
Schiefe	−0.1873	beschnittener Mittelwert 5%	−0.3341
Wölbung	2.7195	beschnittener Mittelwert 10%	−0.3192
1. Moment	0	beschnittener Mittelwert 25%	−0.275
2. Moment	0.0283		
3. Moment	−0.0053	Maximum	−0.20756
4. Moment	0.0022	75%-Quartil (Quantil)	−0.2223
Standardabweichung	0.1683	50%-Quartil (Quantil)	−0.23054
Standardfehler	0.0048	25%-Quartil (Quantil)	−0.43681
Interquartilabstand	0.2145	Minimum	−0.69134

Abbildung 8: Histogramm und CDF aller Fitnesswerte

Für die drei Parameter Startvektor (der initialen Gewichte), Sigma (Skalierung des initialen Suchraumes, ähnlich einer Schrittweitenregelung) und Bevölkerungsgröße (Größe der Stichprobe) werden ohne Vorwissen einige Wertekombinationen getestet, die den zugehörigen Suchraum relativ sinnvoll abdecken sollten. Die Wahlen der verbleibenden CMA-ES-Parameter werden der Standardeinstellung des CMA-ES-Algorithmus überlassen.
Für den Test wird der Metaparameter die maximal erlaubte Anzahl an CMA-ES-Generationen o.B.d.A. auf 25000 festgelegt, um genügend differenzierte Ergebnisse zu erhalten[87]. Aufgrund der dadurch bedingten zeitintensiven Versuche umfasst der zur Verfügung gestellte Datensatz nur 1219 Einträge.

8.1.2 Bemerkungen

Die Betrachtung des CMA-ES-Algorithmus ergibt den Eindruck, dass einige seiner Wesenszüge und Variablen bislang nicht erschöpfend untersucht sind. So sollten beispielsweise die Gewichte w_i, denen eine große Bedeutung bei der Adaption von Mittelwert und Kovarianzmatrix zu kommt, am besten dynamisch zur Laufzeit des Algorithmus bestimmt werden, um z.B. variierende Stichprobeneigenschaften ausnutzen zu können. Ähnlich bieten sich differenziertere Schrittweiten an, die opti-

[87] Eine andere Wahl der maximalen Iterationsanzahl wird in der Diskussion zu diesem Kapitel in Abschnitt 8.5 anhand von Abbildung 19 oder explizit im Test von EANT in Abschnitt 9.3 betrachtet.

mialerweise für jede Dimension separat definiert werden. Zudem ist die Einführung einer separaten Skalierungsvariablen für bessere Analysemöglichkeiten wünschenswert, damit die Schrittweite ihre derzeitige Doppelbedeutung verliert. Ferner sollten Heuristiken - wie z.B. die Konstante μ_{eff} oder der Dämpfungsfaktor d_s - verbessert oder ersetzt werden. Letzteres gilt auch für die Komponenten des Mittelwertvektors, in denen keine oder nur noch kleine Veränderungen stattfinden. Diese sollten dann vernachlässigt und aus der Betrachtung entfernt werden, um keine numerischen Probleme zu verursachen und den Rechenaufwand zu reduzieren. Eine weitere Idee besteht außerdem darin, anstelle der Kovarianzen beispielsweise nur noch die Varianzen zu schätzen, um so den Algorithmus auf $O(n)$ zu beschleunigen. Wie sich jedoch die damit einhergehende Verringerung der Informationsmenge auf das Ergebnis auswirkt, ist offen. Doch auch das Laufzeitverhalten sollte noch Ziel einer Verbesserung werden.

8.1.3 Ergebnisse

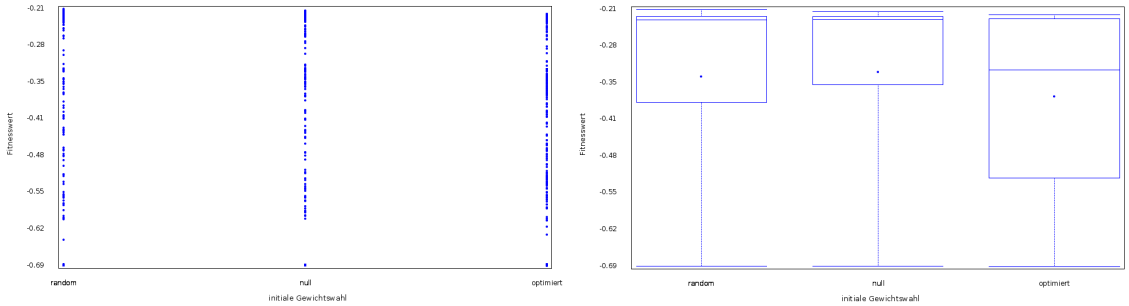
Das Ziel des Tests besteht in einer Verbesserung der Optimierung bzgl. der neuronalen Funktion durch die ausgewählten Parameter. Die dafür zur Analyse benötigten Daten werden aus dem 1219 großen Datensatz je nach Bedarf (z.B. als Querschnitt bzw. Panel Parameter $\times$ Fitnesswert) akkumuliert.

Startvektor

Es werden drei verschiedene Arten von Werten für den Startvektor getestet, normalverteilte $\mathrm{N}(0, 2)$, nullgesetzte und bekannte Gewichte einer vorherigen CMA-ES-Optimierung (ein Vektor gefüllt mit 0.5 wäre hier die Standardeinstellung des CMA-ES-Algorithmus). Abbildung 9 zeigt die resultierenden Daten. Es ist zu sehen, dass nullgesetzte Startwerte die besten (statistischen) Eigenschaften besitzen. Die Versuche mit randomisierten Startgewichten beinhalten zwar den Versuch mit dem besten Fitnesswert, sie sind den Versuchen mit nullgesetzten Startwerten jedoch insgesamt leicht unterlegen, wie im Boxplot sehr schön zu sehen ist. Da nullgesetzte und randomisierte Werte allerdings numerisch sehr nah beieinander liegen, sollten weitere Versuche letztendliche Klarheit erbringen. Es wird aber vermutet, dass Nullwerte den optimalen Ausgangspunkt markieren. Der Algorithmus ist damit auf jeden Fall nicht startwertunabhängig.

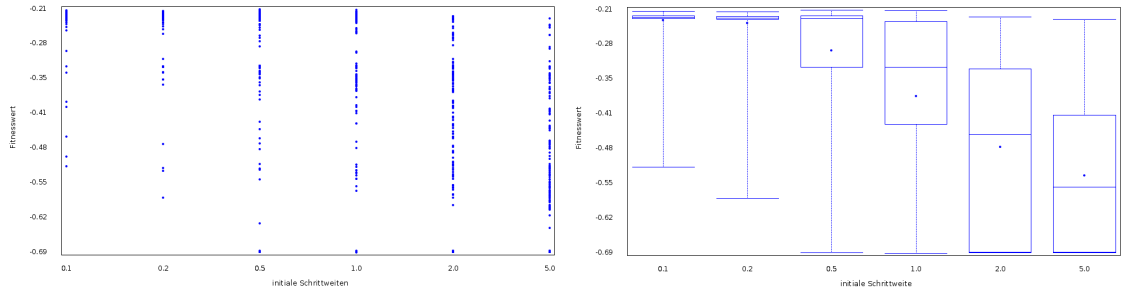
Schrittweite

Es werden sechs verschiedene Werte für die zu erwartende Schrittweite (Sigma) getestet, 0.1, 0.2, 0.5, 1.0, 2.0 und 5.0 (0.3 wäre hier die Standardsetzung von CMA-ES). Abbildung 10 zeigt die resultierenden Daten, und es ist erkennbar, dass kleine initiale Schrittweiten die beste Wahl darstellen. Je kleiner die Startschrittweiten gewählt werden, desto besser sind die Fitnesswerte. Umgekehrt ist für größere Schrittweiten ein starker Anstieg der Zahl schlechter Ergebnisse zu erkennen. Zu große initiale Schrittweiten lassen den Algorithmus demnach, siehe Boxplot, öfters in Regionen des Suchraums springen, aus denen er sich nicht wieder befreien kann. Die Wahl kleinerer initialer Schrittweiten jenseits von 0.1 wird die Ergeb-



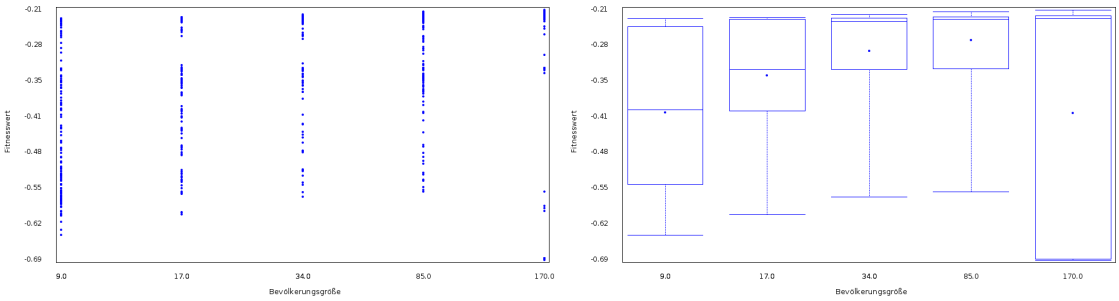
initiale Gewichtswahl	**random**	**null**	**optimiert**
Anzahl der Datenpunkte	363	394	462
Mittelwert	-0.334	-0.326	-0.372
Varianz	0.027	0.026	0.03
Interquartilabstand	0.162	0.128	0.3
Maximum	-0.208	-0.211	-0.217
75%-Quartil (Q3)	-0.221	-0.221	-0.225
50%-Quartil (Q2)	-0.227	-0.226	-0.321
25%-Quartil (Q1)	-0.383	-0.349	-0.525
Minimum	-0.691	-0.691	-0.691

Abbildung 9: Scatter- und Boxplot für das Panel Gewichtswahl × Fitnesswert



initialen Schrittweite	**0.1**	**0.2**	**0.5**	**1**	**2**	**5**
Anzahl der Datenpunkte	230	227	208	203	176	175
Mittelwert	-0.229	-0.234	-0.289	-0.38	-0.481	-0.538
Varianz	0.0010	0.0020	0.017	0.028	0.027	0.019
Interquartilabstand	0.0050	0.0060	0.102	0.205	0.365	0.273
Maximum	-0.211	-0.212	-0.208	-0.209	-0.222	-0.227
75%-Quartil (Q3)	-0.22	-0.22	-0.22	-0.231	-0.325	-0.417
50%-Quartil (Q2)	-0.223	-0.224	-0.225	-0.322	-0.456	-0.56
25%-Quartil (Q1)	-0.225	-0.226	-0.321	-0.436	-0.69	-0.69
Minimum	-0.52	-0.583	-0.691	-0.691	-0.691	-0.691

Abbildung 10: Scatter- und Boxplot für das Panel Schrittweite × Fitnesswert



Bevölkerungsgröße	**9**	**17**	**34**	**85**	**170**
Anzahl der Datenpunkte	162	162	166	307	422
Mittelwert	-0.407	-0.335	-0.287	-0.266	-0.408
Varianz	0.02	0.012	0.008	0.006	0.051
Interquartilabstand	0.305	0.177	0.1	0.1	0.471
Maximum	-0.225	-0.222	-0.216	-0.212	-0.208
75%-Quartil (Q3)	-0.24	-0.226	-0.223	-0.221	-0.22
50%-Quartil (Q2)	-0.401	-0.323	-0.23	-0.225	-0.224
25%-Quartil (Q1)	-0.545	-0.403	-0.323	-0.321	-0.69
Minimum	-0.643	-0.603	-0.57	-0.56	-0.691

Abbildung 12: Scatter- und Boxplot für das Panel Bevölkerungsgröße × Fitnesswert

nisse wahrscheinlich noch etwas verbessern können, denn ein Wert von 0 setzt das Stichprobenmittel der ersten Generation auf den Punkt x_0, der ansonsten nicht ausgewertet wird. In Bezug auf die chaotische Natur eines Neuronalen Netzwerkes ist der Punkt x_0 zudem einer, der bei nullgesetzten Startgewichten auf jeden Fall keine Probleme bereiten kann.

Bevölkerungsgröße

Es werden fünf verschiedene Werte für die Größe der Bevölkerung getestet, 9, 17, 34, 85 und 170 (für die gegebene Problemdimension $n = 85$ wäre hier $4 + \lfloor 3 \cdot \log(n) \rfloor = 17$ die Standardwahl des Algorithmus).

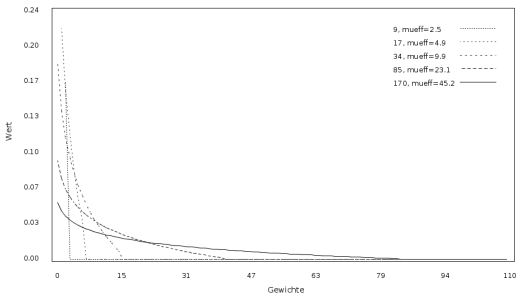


Abbildung 11: Standardgewichtungen nach Bevölkerungsgrößen

Abbildung 12 veranschaulicht die Resultate. Es zeigt sich, dass größere Bevölkerungen nicht die besten statistischen Eigenschaften haben. Dort findet sich zwar das beste Ergebnis, zugleich weist diese Parameterwahl aber eine unerwünscht große Varaianz auf, siehe auch Bias-Varianz-Dilemma in Bishop (2004) oder Duda u. a. (2004). Da die Datenerhebung relativ homogen erfolgt, wie Tabelle 18 offenbart, liegt vermutlich ein Problem im Optimierungsverfahren vor. Werden dazu allerdings die Abhängigkeiten der verschiede-

Startschrittweite	0.1	0.2	0.5	1	2	5
Panelgrößen	79	77	79	74	56	57
Startgewichte	random	null	optimiert			
Panelgrößen	140	141	141			

Tabelle 18: Aufteilung der Datenpunkte für Tests mit 170 Individuen

nen CMA-ES-Variablen zur Bevölkerungsgröße betrachtet, so ist bei Verwendung der Standardeinstellungen beinahe jede Variable von dieser abhängig. Ohne eine weitere detaillierte Analyse ist daher eine Aussage schwierig.
Es wird jedoch angenommen, dass die Wahl der Gewichtungskoeffizienten in Gleichung (9) mit wachsender Bevölkerung zu einer immer schlechteren Adaption des Stichprobenmittels führt. Abbildung (11) zeigt diese (exponentiell) abfallende und normalisierte Gewichtung der Individuen zur Berechnung von Mittelwert und Kovarianz für die getesteten Werte. Für eine ansteigende Bevölkerungszahl ist ein deutliches Schrumpfen der ersten Gewichte zu sehen, welche zur Gewichtung der besten Individuen benutzt werden. Das Absinken dieser Gewichte hat zur Folge, dass Stichprobenmittel und Kovarianz mehr Anteile an schlechteren Individuen erhalten und in ihrer Entwicklung stagnieren. Zur Verifizierung dieser Vermutung zu ist deshalb nach einer besseren Gewichtung zu suchen. Bis dahin stellen Bevölkerungsgrößen um 85 eine gute Wahl für den Benchmark dar, wobei eine größere Anzahl bessere Spitzenwerte erzielt.

Abhängigkeiten

Zur Auflösung höherdimensionaler (als eindimensionaler) Abhängigkeiten werden aus den 1219 Versuchen zweidimensionale Paneldatensätze erstellt und in den Abbildungen 13 und 14 dargestellt[88]. Dabei sind in Richtung der y-Achse in Abbildung 13 das arithmetische Mittel des Fitnesswertes und in Abbildung 14 die Standardabweichung des jeweiligen Panels angegeben. Die Auflösung der Grafiken ist wegen der wenigen Datenpaare relativ niedrig, konnte aber aufgrund begrenzter Rechnerkapazitäten nicht erhöht werden. Erwartungsgemäß sind jedoch für beide Ansichten die oben beschriebenen Effekte der einzelnen Parametervariationen zu sehen. Weitere zweidimensionale Abhängigkeiten unter den untersuchten Variablen, die sich positiv nutzen lassen, sind nicht auszumachen.

Abbruchkriterien

Neben der Analyse obiger Parameter zeigt die Betrachtung der genutzten Funktionsevaluationen eines Versuches (siehe Abbildung 15) die anzutreffende Korrelation zwischen diesen und dem resultierenden Fitnesswert. Mehr Funktionsevaluationen verbessern dabei erwartungsgemäß - sichtbar durch weiter rechts liegende Punkte - sehr häufig das Ergebnis.
Die horizontalen Linien sind auf verschiedene Bevölkerungsgrößen zurückzuführen, da jedes Individuum pro Generation genau einmal ausgewertet wird. Es ist jedoch

[88] Die Achsenbeschriftung der dreidimensionalen Grafiken ist numerisch. Für die Startgewichte steht 0 dabei für kleine normalverteilte Werte $N(0, 2)$, 1 für nullgesetzte und 2 für bereits optimierte Gewichte.

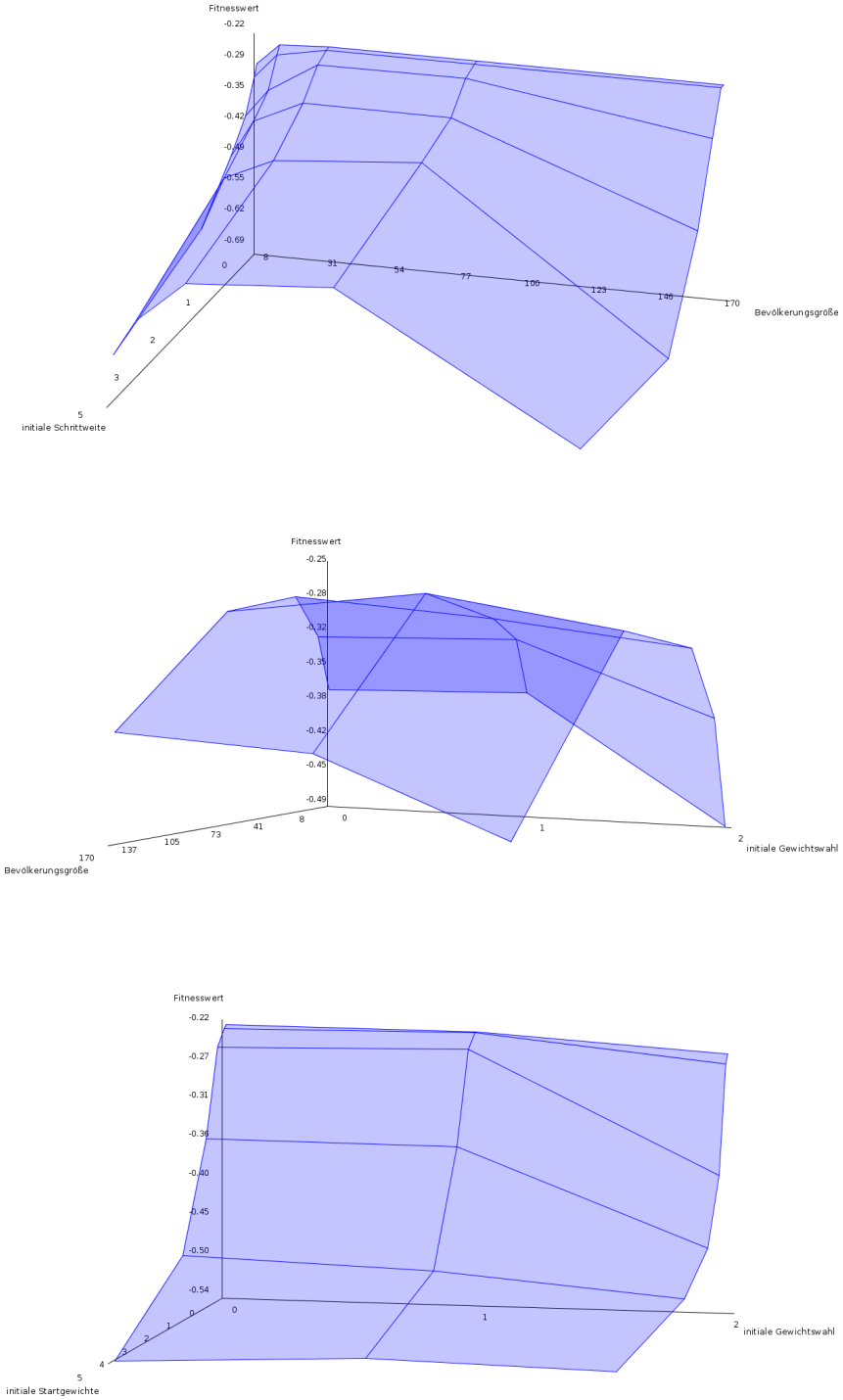


Abbildung 13: Dreidimensionale Darstellung der resultierenden gemittelten Güte zu Paaren der untersuchten Parameter

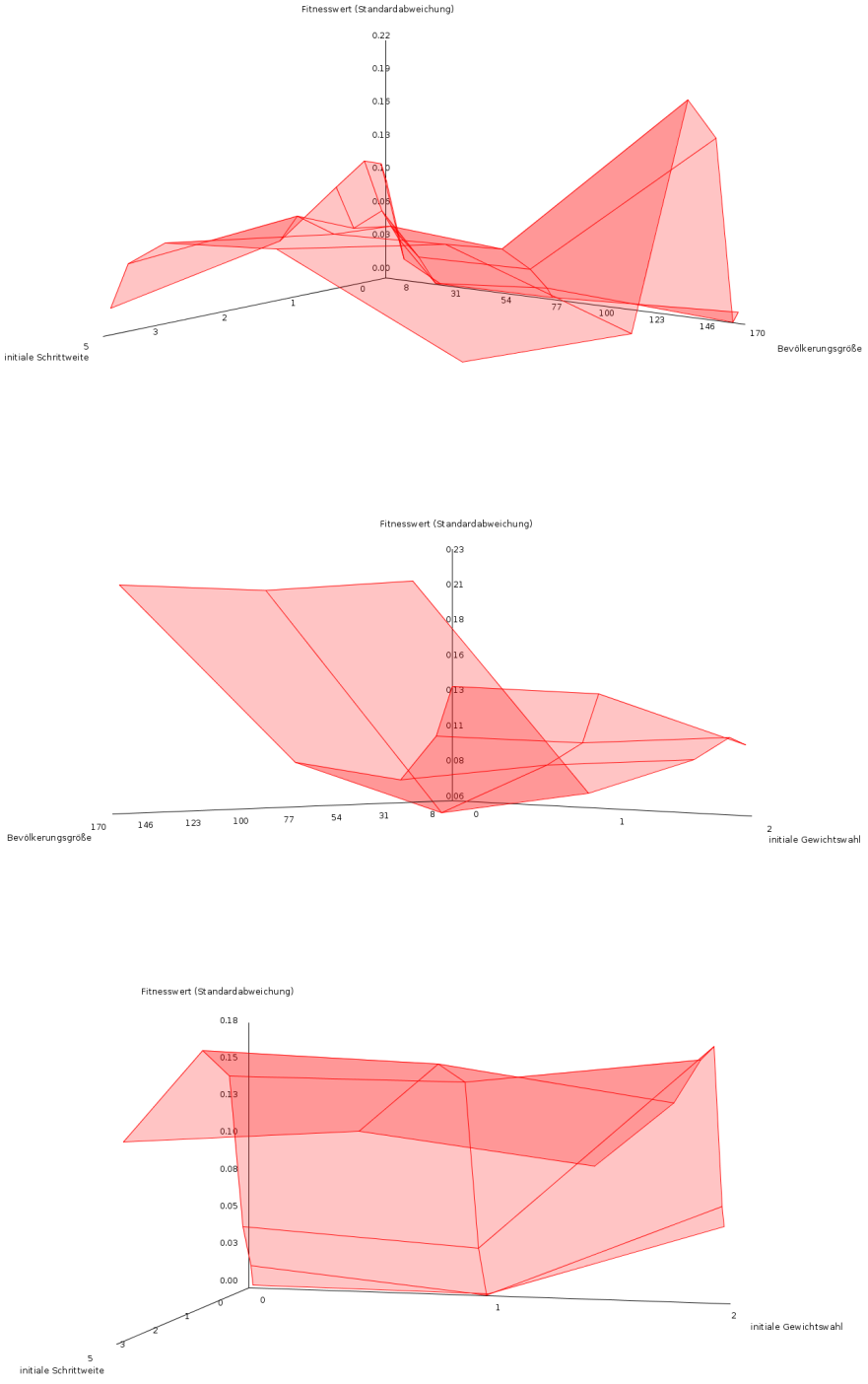


Abbildung 14: Dreidimensionale Darstellung der resultierenden Standardabweichungen zu Paaren der untersuchten Parameter

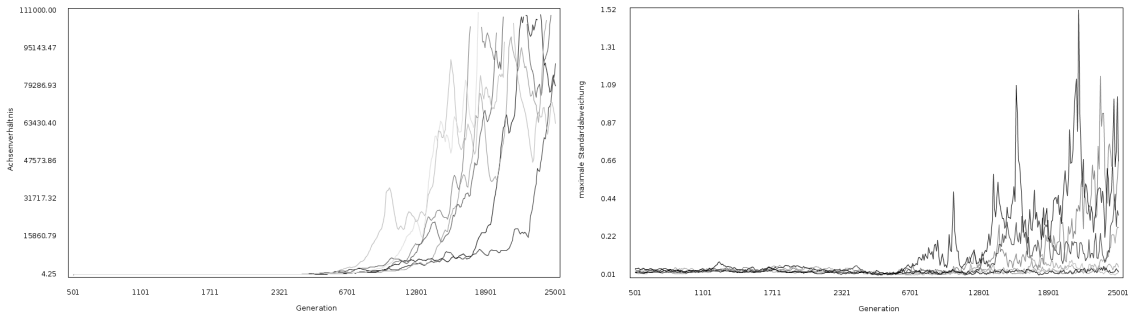


Abbildung 16: Maximale Achsenverhältnisse orthogonalisierter Kovarianzmatrizen und Standardabweichungen zufällig ausgewählter Versuche

zu sehen, dass mehrere Versuche mit 170 Individuen nicht die maximal möglichen[89] Funktionsevaluationen ausnutzen. Der Algorithmus terminiert in diesem Fällen, ohne das Optimum zu finden. In der Erwartung, dass diese zu frühen Abbrüche bei größeren Bevölkerungen nicht überhand nehmen, wird empfohlen, sowohl Bevölkerungsgröße als auch maximale Interationsanzahl zu erhöhen, sofern man die dadurch ebenfalls ansteigende Laufzeiten akzeptiert. Anstelle der Iterationzahl kann dabei auch die Zahl der Funktionsevaluationen fixiert werden.

Ferner zeigen die Testreihen[90], dass die Zahl der aufgrund von Stagnation abgebrochenen Optimierungen zu Beginn einer Evolution hoch ist und dann mit der Zeit sinkt, während komplementär die Anzahl der Abbrüche wegen des Erreichens der maximalen Iterationsanzahl ansteigt. Obwohl auch andere Abbruchbedingungen gesetzt waren, kommen nur diese beiden zum Einsatz.

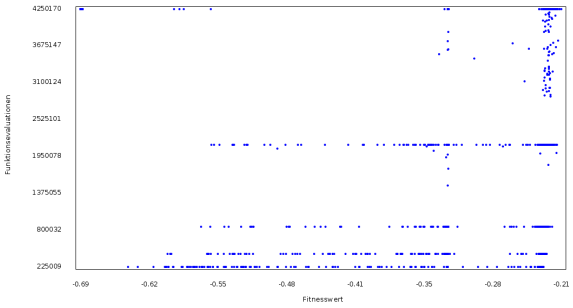


Abbildung 15: Effizienzpunkte aller CMA-ES-Versuche

Damit treten die in Abschnitt 5.2.5 beschriebenen numerischen Probleme, die z.B. zum Versagen des Algorithmus in Beispiel 2 der Abbildung 18 führen, tendenziell in den Hintergrund. Dennoch zeigen die maximalen Achsenverhältnisse $\frac{d_{max}}{d_{min}}$ und Standardabweichungen $\sigma \cdot \sqrt{C_{max,max}}$ von Versuchen, die bis zur maximalen Iterationszahl laufen und in Abbildung 16 abgebildet sind, fast immer einen signifikaten Anstieg gegen Ende einer Optimierung. Aufgrund der Doppelbedeutung von σ bleibt allerdings unklar, welche Einflüsse dafür kausal sind.

Wird angenommen, dass keine numerischen Probleme existieren, könnte dieses Verhalten durch eine Verlängerung des Evolutionspfades, der über Gleichung (8) die Adaption der Schrittweite vorgibt, aufgrund von Fitnessverbesserungen begründet werden. In diesem Fall sollte die Adaptionsregel überprüft werden.

Werden hingegen numerische Probleme unterstellt, könnten diese entweder durch die chaotische Natur der zu optimierenden neuronalen Funktion zustande kommen,

[89] Da die Funktionsevalutationen im Test kein Abbruchkriterium darstellen, ergibt sich ihr maximal erreichbarer Wert als Produkt aus maximaler Iterationsschrittsanzahl und Bevölkerungsgröße.

[90] Siehe Datensätze auf dem beiliegenden Datenmedium, Seite 150.

die fitnessverbessernde Veränderungen an Netzgewichten zumeist auf immer kleinere Werte begrenzt, oder auf das Skalierungsproblem des CMA-ES-Algorithms zurückzuführen sein. Je nach Kausalität müsste dann eine genaue Untersuchung folgen. Wahrscheinlich ist jedoch, dass beide Hypothesen teilweise zutreffen.

8.1.4 Zusammenfassung

Die beste Parameterkombination ergibt sich nach obigen Untersuchungen als initiale Schrittweite von 0.1, nullgesetzten Startgewichten und einer Bevölkerungsgröße von 85. Für einen Eindruck des Verhaltens dieser Parameterwahl und einen Vergleich gegenüber den nun nachfolgend vorgestellten Algorithmen, sind in Abbildung 19 (Seite 84) 90 Optimierungen mit maximal 1500 (Zeitersparnis) Iterationen bzw. Generationen zusammengefasst.
Anschaulich zeigt sich ein abfallender Verlauf, der Folge einer besser werdenden Fitness im Laufe des Evolutionsprozesses ist. Das enge Zusammenliegen der einzelnen Versuchsergebnisse veranschaulicht dabei die relativ kleine Standardabweichung seiner Ergebnisse, so dass der Algorithmus für diesen Test als konstistent angesehen wird. Für die untersuchte neuronale Funktion übertrifft er ferner beide Konkurrenten. Er ist jedoch nicht noch ausgereift, was die zahlreichen Probleme bei seiner Verwendung und die intern genutzten Heuristiken belegen.

8.2 Test: Stochastic Reflections

In einem Alternativtest wird die Gewichtsoptimierung des Neuronalen Netzwerkes aus Abbildung 7 mit dem Algorithmus Stochastic Reflections aus Abschnitt 5.3 getestet.
Sämtliche Parameter werden dabei auf gute Werte aus einer kleinen Vorversuchstestreihe fixiert vorgegeben, um den Test einfach zu halten. Für den initialen Vektor der Startgewichte wird der Nullvektor gewählt. Es werden ferner 10 Neustarts, eine Bevölkerungsgröße von 30 Individuen, eine minimale Toleranz für die Varianz der Fitness von $1e-16$ bei einem Zeitfenster von 5 Iterationsschritten und maximal 100000 Iterationen festgelegt.
Wie zu erwarten ist, zeigen Vorversuche bei kleinen Netzwerken bzw. einfachen (neuronalen) Funktionen ein Geschwindigkeitsvorteil von SR gegenüber CMA-ES. Mit steigender Komplexität beginnt sich dieses Verhältnis allerdings zunehmend umzukehren.
Abbildung 19 zeigt für das in Abbildung 7 genannte neuronale Netz 90 mit dem SR-Algorithmus durchgeführte Benchmarkoptimierungen und die zugehörigen Statistiken. Dem abfallenden Verlauf der Lernkurven ist zu entnehmen, dass der Algorithmus funktioniert, obgleich er oftmals viel zu früh stoppt. Zugleich ist an der großen Ergebnisvarianz anschaulich zu erkennen, dass der Algorithmus in einer konstanten Testumgebung nicht konsistent und damit auch nicht robust gegenüber anderen Aufgabenstellungen ist.

Trotzdem ist die Demonstration eines schnellen evolutionären Algorithmus zur Funktionsminimierung mit einer Laufzeit von $O(n)$ damit insgesamt gelungen, auch wenn seine Qualität noch stark verbesserungsbedürftig ist.

8.3 Test: Stochastic Downhill Simplex

In einem weiteren Alternativtest wird die Gewichtsoptimierung des Neuronalen Netzwerkes aus Abbildung 7 mit einer randomisierten Version des Algorithmus Stochastic Downhill Simplex aus Abschnitt 5.4 getestet.
Auch für diesen Test werden die Parameter auf bekannte, gute Werte fixiert, um ihn einfach zu halten. Die Problemdimension des verwendeten Netzwerkes beträgt $n = 85$. Als initialer Vektor der Startgewichte wird der Nullvektor gewählt. Ferner werden 10 Neustarts, eine Bevölkerungsgröße von $n + 1$ Individuen, ein minimales Volumen für den Simplex von $1e - 12$ und maximal 100000 Iterationen festgelegt. Die verwendeten und typischen Koeffizienten sind 1.0 für eine Reflektion, 0.5 für eine Kontraktion sowie 2.0 für eine Expansion.
Abbildung 19 zeigt für das in Abbildung 7 genannte Netzwerk 90 mit SDS durchgeführte Benchmarkoptimierungen und die zugehörigen Statistiken. Wie darin gezeigt, funktioniert der Algorithmus und die Lernkurven verlaufen entsprechend fallend.
Ferner ist die Standardabweichung der Ergebnisse im Gegensatz zu der von SR erfreulich gering, so dass der Algorithmus bzgl. des Benchmarks als konstistent betrachtet werden darf. Er unterliegt dennoch dem CMA-ES-Algorithmus, und aufgrund der Vorgehensweisen ist zu erwarten, dass diese Differenz mit steigender Komplexität der zu minimierenden Aufgabe weiter zunimmt.

8.4 Numerische Probleme

Die Ergebnisse der verschiedenen Algorithmen zeigen neben individuellen Besonderheiten ein wahrscheinlich allgemeines Problem bei der Optimierung der neuronalen Funktion. Kein Algorithmus erreicht den optimalen Fitnesswert 0 oder kommt in dessen Nähe. Da dieses Phänomen scheinbar nicht auf eine spezielle Vorgehensweise begrenzt ist, wird vermutet, dass ein numerisches Problem in Form der Konditionierung der gestellten Aufgabe vorliegt, wie dies in den Abschnitten 4.7 und 4.8 angesprochen wird.
Zur anschaulichen Begründung der schlechten Konditionierung einer neuronalen Funktion wird in Abbildung 17 die Oberfläche der Benchmarkfunktion aus Kapitel 7 nach der Optimierung des neuronalen Netzwerkes aus Abbildung 7 mittels CMA-ES-Algorithmus veranschaulicht. Die dargestellten und in der Nähe des gefundenen Optimums prozentual variierten Parameter zeigen jeweils c.p. ein sehr zerklüftetes Bild der Funktion, woran die schlechte Kondition des Problems zu erkennen ist.
Jedoch konnte im Rahmen der durchgeführten Versuche kein Verfahren gefunden werden, das mit diesem Problem effektiv umgehen kann. Die Analyse der Parameterabhängigkeiten im Rahmen der CMA-ES-Optimierung aus Abschnitt 8.1 be-

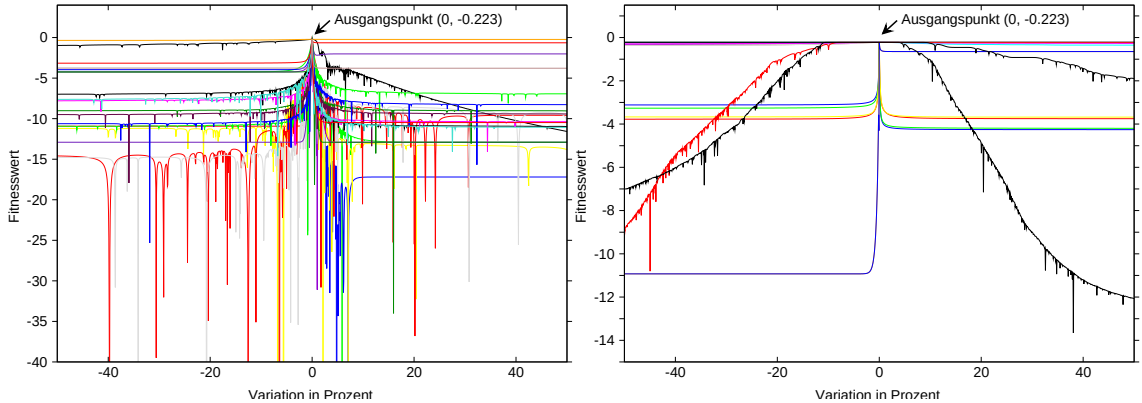


Abbildung 17: Variation um ein lokales Optimum

stätigt aber die Existenz[91] von verfahrenstechnisch noch ungenutzter Information in Form der Korrelation zwischen verschiedenen Gewichten im Netzwerk. Abbildung 18 zeigt dazu die (absolute) Korrelationsmatrix des Stichprobenmittels für drei veschiedene Optimierungen der Benchmarkfunktion mittels CMA-ES unter Benutzung des oben genannten Netzwerkes. Die senkrechten und waagerechten Linien stehen für positive (schwarz) bzw. negative (weiss) Korrelationen. Deutlich sind wiederholt auftretende Muster zu erkennen, die in zukünftigen Algorithmen vielleicht zu einer besseren Behandlung des neuronalen Konditionierungsproblems führen können. Beispiel 1 zeigt zudem gegenüber Beispiel 3, dass bei konstanter Iterationszahl scheinbar die Hinzunahme von Gewichten die Korrelation insgesamt reduziert. Der CMA-ES-Algorithmus konvergiert vermutlich auch deshalb mit zusätzlichen Gewichten nicht so schnell wie ohne diese.

8.5 Diskussion

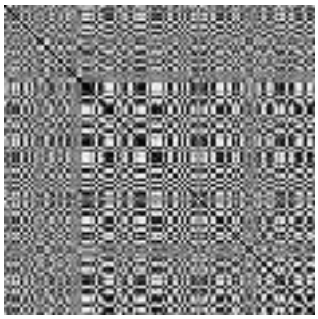
Abgesehen von Laufzeitschwächen bei kleiner Problemdimension, wie z.B. bei der in Vorversuchen benutzten Funktion

$$f : \mathbb{R}^3 \to \mathbb{R}^4, \quad (x, y, z) \mapsto \begin{pmatrix} 4 + x^2 \cdot z^3 \cdot y \\ -9 + y^2 + x^3 + z^4 \\ -0.9 + \frac{1}{z \cdot x^3} \\ \sqrt{2} + x^2 \cdot y^2 \end{pmatrix},$$

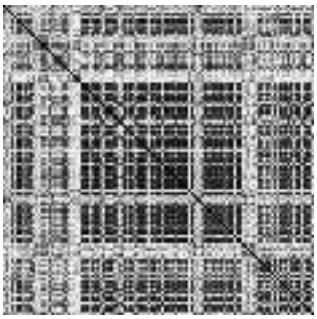
zu der die Ergebnisse nicht explizit angeführt werden[92], übertrifft der CMA-ES-Algorithmus fast immer die Konkurrenten SR und SDS. Dabei ist jedoch seine (hohe quadratische) Laufzeitzunahme mit steigender Problemdimension deutlich zu verzeichnen. Im Gegenzug arbeiten SDS und SR zwar pro Iterationschritt schneller,

[91] Die ermittelten Korrelationen basieren auf den Stichprobenmittelpunkten (arithmetisches Mittel) der CMA-ES-Generationen.

[92] Diese Ergebnisse sind mit der Software auf dem beiliegenden Datenmedium nachprüfbar.

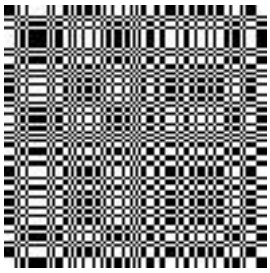


(Orginalwerte)
$[-1, 1] \mapsto$ [weiss, schwarz]

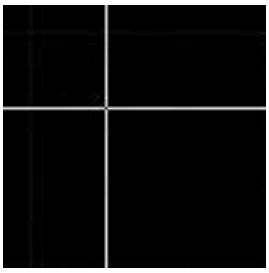


(Absolutwerte)
$[0, 1] \mapsto$ [weiss, schwarz]

Beispiel 1: eine erfolgreiche Optimierung
(mit Verbindungen zwischen jedem Eingabe- und Ausgabeneuron)

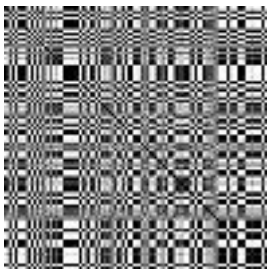


(Orginalwerte)
$[-1, 1] \mapsto$ [weiss, schwarz]

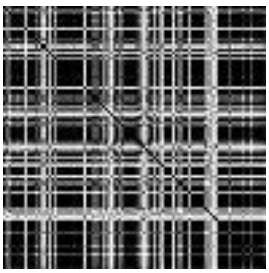


(Absolutwerte)
$[0, 1] \mapsto$ [weiss, schwarz]

Beispiel 2: eine misslungene Optimierung
(ohne Verbindungen zwischen jedem Eingabe- und Ausgabeneuron)



(Orginalwerte)
$[-1, 1] \mapsto$ [weiss, schwarz]



(Absolutwerte)
$[0, 1] \mapsto$ [weiss, schwarz]

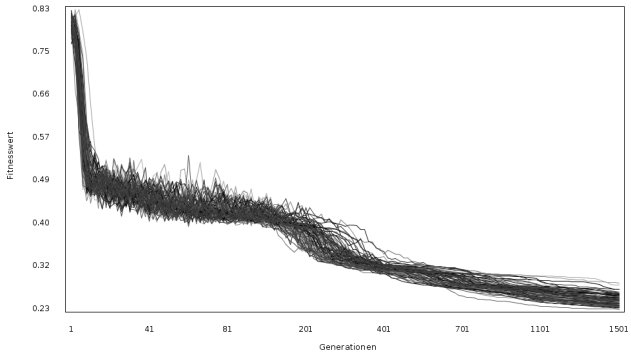
Beispiel 3: eine weitere erfolgreiche Optimierung
(ohne Verbindungen zwischen jedem Eingabe- und Ausgabeneuron)

Abbildung 18: Korrelationen der Netzgewichte für drei CMA-ES-Optimierungen

sind aber auch in ihrer Informationsmenge beschränkter und brauchen ab einer gewissen Komplexität der Zielfunktion letztendlich mehr Zeit als CMA-ES. Insbesondere zeigt Abbildung 19, dass eine (lokale) kostspielige Informationsgewinnung und -nutzung kann also durchaus lohnend sein. Diese Testaussage wird zudem durch den Erfolg von CMA-ES bei einem internationalen Wettbewerb[93] gestützt. Für schlecht konditionierte Aufgaben unterliegt er jedoch - genau wie andere Algorithmen auch - numerischen Problemen.
Die theoretische Laufzeitabschätzung eines Iterationsschrittes ist somit kein ausreichendes Vergleichsmaß für Minimierungsalgorithmen. Ebenso führt allerdings die (übliche) Erweiterung um eine Abschätzung des Konvergenzverhaltens zu keinem optimalen Maßstab, da diese Abschätzung abermals nur von der (iterativen) Vorgehensweise, nicht aber von der benutzten Informationsmenge abhängt. Verschiedene Verfahren mit denselben Konvergenzeigenschaften weisen schließlich nicht immer dieselben Ergebnisse auf, denn sie gehen entweder bei der Informationsgewinnung oder der -verarbeitung unterschiedlich vor.
Das Problem bei der Definition eines als geeignet angesehenen Effizienzmaßes (vergleiche Abschnitt 5.5) ist die bisherige Latenz eines allgemeinen (kontextfreien) Informationsbegriffes, der aus diesem Grund bislang abstrakt gehaltenen wird. Doch selbst eine kontextabhängige Definition[94] wäre eine Verbesserung der Situation.

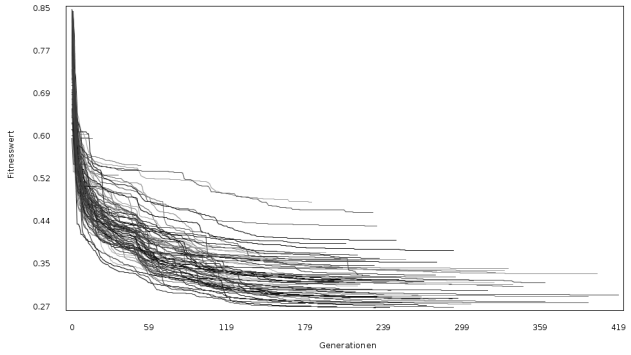
[93] Eine parameterfreie Variante des CMA-ES-Algorithmus übertrifft nahezu ausnahmslos die konkurrierenden Minimierungsverfahren des Wettbewerbs (IEEE Congress on Evolutionary Computation: Special Session on Real-Parameter Optimization at CEC-05, Edinburgh, UK, 2-5 Sept. 2005).

[94] Die Messung einer allerdings darstellungsabhängigen Informationsmenge könnte über den zur Laufzeit verwendeten Speicherplatz (z.B. in Bits) für Stichproben- und deren Verarbeitungsdaten durchgeführt werden. Ein praktikables Effizienzmaß wäre dann beispielsweise als Produkt aus Laufzeit (der Kontextabhängigkeit der Informationsdarstellung geschuldet) und Speicherplatz (TIME $\cdot$ SPACE) realisierbar. Kleinere Werte stünden dabei für Effizienz im Umgang mit Information, größere für Ineffizienz.



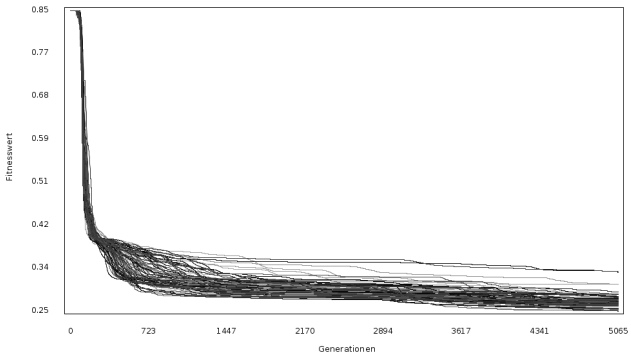
CMA-ES: 90 Optimierungen

arithmetisches Mittel	0.2468	Standardfehler	0.0011
Median	0.2453	Standardabweichung	0.0105
Interquartilabstand	0.0137	Minimum	0.2291



Stochastic Reflections: 90 Optimierungen

arithmetisches Mittel	0.3579	Standardfehler	0.0087
Median	0.3336	Standardabweichung	0.0821
Interquartilabstand	0.0673	Minimum	0.2714



Stochastic Downhill Simplex: 90 Optimierungen

arithmetisches Mittel	0.2698	Standardfehler	0.0014
Median	0.2687	Standardabweichung	0.0134
Interquartilabstand	0.0116	Minimum	0.2499

Abbildung 19: Testreihen zum Vergleich evolutionärer Optimierungsalgorithmen

9 Evolutionsergebnisse

Der Versuchsaufbau in Form der Offlinevariante des Visual Servoing-Benchmarks aus Kapitel 7 beschreibt eine Lernaufgabe im Batchmode: Über die gesamte Sammlung aus Paaren von Startposition und Zielmarkierung wird genau ein Fitnesswert berechnet, welcher dann (im Modus des Verstärkenden Lernens) die einzige Information für die Entwicklung eines Neuronalen Netzwerkes darstellt.
Nach Aufgabenstellung und wie in Kapitel 4 beschrieben, ist dabei auf Basis geeigneter Neuronen die Entwicklung eines Neuronalen Netzwerkes mit optimalem Fitnesswert unter der Nebenbedingung minimaler Netzstruktur das vorrangige Ziel, vor einem Test auf Generalisierungseigenschaften. Mit dieser Vorgabe werden anhand des Visual Servoing-Benchmarks die Tests von EANT und NEAT durchgeführt und Vergleiche gezogen.
Typischerweise werden dafür die Ergebnisse der besten Parameterisierungen der Algorithmen miteinander verglichen. Dementsprechend besteht die wesentliche (Vor-) Arbeit in der Bestimmung der optimalen Parameter der Algorithmen. Für den NEAT-Algorithmus kann dies (theoretisch) vollständig durchgeführt werden. Der EANT-Algorithmus hingegen benutzt in der vorliegenden (Entwickler-) Version (noch) viele heuristisch fixierte Einstellungen, so dass lediglich die Parameter seiner Gewichtsoptimierung, welche fast ausschließlich Bestandteil eines externen Optimierungsverfahrens sind, untersucht werden können. Diese sind allerdings Teil der funktionalen Optimierung und gesondert in Abschnitt 8.1 analysiert.
Nach dem Vergleich von NEAT und EANT erfolgt außerdem die kurze Zusammenfassung einiger Testreihen zur allgemeinen evolutionären Entwicklung neuronaler Topologien sowie eine Analyse des neuen Algorithmus TS.
Wie schon im vorangehenden Kapitel, wird darauf hingewiesen, dass vor allem wegen hoher Laufzeiten oft nicht ausreichend viele Daten gesammelt werden können, um umfassende statistische Untersuchungen oder Hypothesentests durchzuführen[95].

9.1 Anmerkungen

Für einen Vergleich der Algorithmen EANT und NEAT wird neben deren unterschiedlicher Vorgehensweise auch deren technische Umsetzung in Form der gewählten Programmiersprache bzw. des erzeugten Kompilats beachtet. Deshalb werden an dieser Stelle die für den folgenden Vergleich wesentlichen Unterschiede kurz hervorgehoben:

- Der EANT-Algorithmus trennt Struktur- und Gewichtsoptimierung. Er und NEAT verwenden zwar eine stochastische Suche bzw. Mutation für den diskreten Suchraum der Netzstruktur, doch während der NEAT-Algorithmus diese auch zur Adaption der Gewichte verwendet, benutzt EANT dafür in

[95] Sämtliche Datensätze und Ergebnisse dieser Arbeit befinden sich auf dem beiliegenden Datenmedium (Seite 150).

einem separaten Schritt den CMA-ES-Algorithmus. Dies ist die wesentlichste theoretische Differenz zwischen beiden Algorithmen.

- Der EANT-Algorithmus ist in C/C++ geschrieben, der NEAT-Algorithmus ursprünglich in C und Lisp, wobei die hier verwendete Version in Java implementiert ist. Dies ist nach den vorliegenden praktischen Erfahrungen die wesentlichste Differenz zwischen beiden Algorithmen.

- Der EANT-Algorithmus parallelisiert speziell die zeitaufwendige Gewichtsoptimierung per CMA-ES-Algorithmus mittels PVM, wohingegen der NEAT-Algorithmus keine Parallelisierung unterstützt. Im NEAT-Algorithmus könnte allerdings mit relativ wenig Aufwand eine Parallelisierung auf Individuenebene eingeführt werden.

- Der EANT-Algorithmus benutzt im Gegensatz zum NEAT-Algorithmus nur eine einzige Spezie, so dass die Möglichkeit einer evolutionären Speziation (Versuch eine große Vielfalt an betrachteten Lösungen im Suchraum zu gewährleisten) konzeptionell fehlt.

- Der EANT-Algorithmus besitzt in seiner derzeitigen Entwicklerversion in etwa[96] einen Parameter weniger als der NEAT-Algorithmus (22 zu 23). Aufgrund der Separierbarkeit der Parameter weist die Suche nach einer optimalen (kontextabhängigen) Parameterisierung beim EANT-Algorithmus dabei eine tendenziell kleinere Varianz als beim NEAT-Algorithmus auf.

Für eine bessere Vergleichbarkeit werden außerdem einige zusätzliche Voraussetzungen getroffen und der NEAT in einigen Punkten an EANT angepasst:

- Beide Algorithmen erhalten unskalierte Eingabedaten und benutzen anstelle der sonst in NEAT und ursprünglich auch in EANT verwendeten Sigmoidfunktion den Tangenshyperbolikus.

- Im NEAT4J-Algorithmus wird die Rückgabe eines Neurons ohne eingehende Signale von -1 auf $0 = \tanh(0)$ abgeändert und damit EANT angepasst.

- Zur Effizienzsteigerung von NEAT4J werden Verbindungen zum Biasneuron auf Quellcodeebene entfernt, da diese für den Benchmark irrelevant sind und eine Deaktivierung ansonsten nicht vorgesehen ist. In EANT werden sie jedoch weiterhin benutzt, weil dieser Algorithmus auf sie - im Gegensatz zu NEAT - unproblematisch reagiert.

- In NEAT4J wird eine Korrektur bei der Erzeugung initialer Netzwerke vorgenommen, damit voll verbundene Netze fehlerfrei generiert werden können.

[96] Während wegen vermuteter geringfügiger Relevanz für den Algorithmus bei Betrachtung von EANT die recht willkürliche Konstruktion eines minimalen Teilnetzwerkes (AddSubNetwork) vernachlässigt wird, geschieht dies analog bei NEAT für ein paar weniger bedeutende, aber ebenfalls fixierte Einstellungsmöglichkeiten der Mutation.

Auf einen Vergleich der erzeugten neuronalen Topologien (Struktur) wird ferner verzichtet, weil einerseits die Ergebnisse zu schlecht sind, um eine Bewertung sinnvoll erscheinen zu lassen, und andererseits (insbesondere für sehr große Netzwerke) keine theoretischen Kriterien bekannt sind[97], nach denen topologische Differenzen (z.B. statistisch) bewertet werden können. Einzig die Netzwerkgröße kann in der empirisch begründeten Annahme, dass kleinste Netzwerke am besten generalisieren, als Maßstab verwendet werden. Das Augenmerk liegt demnach primär auf dem Fitnesswert, wohingegen die meisten topologischen Kriterien Hypothesen darstellen und hier nicht betrachtet werden.
Des Weiteren wird für die Analyse des Algorithmus TS an Stelle der Netzgröße eine Kreuzvalidierung für $k = 10$ betrachtet, da hier die Betrachtung der Generalisierung aufgrund guter Fitnesswerte wieder Sinn macht.

9.2 Test: NEAT

Für die Testreihen des NEAT-Algorithmus in Form der unparallelisierten NEAT4J-Implementierung (Version 1.0, September 2006) wird auf verschiedenen Plattformen als Laufzeitumgebung (für die Übersetzung sowie zur Ausführung des Bytecodes) das JRE in Version 1.6 verwendet. Sofern nicht anders genannt, steht daher im Folgenden der Begriff NEAT für NEAT4J.
Die Analyse des Algorithmus gestaltet sich relativ aufwendig, da mehr als 20 Parameter zum Algorithmus gehören. Würden allein für jeden Parameter nur 2 Möglichkeiten getestet, so müssten für 20 Parameter bereits 20^2 Kombinationen getestet bzw. Versuche durchgeführt werden, um nur einen kleinen Eindruck des Verhaltens von NEAT zu erhalten. Aufgrund der stochastischen Natur des Algorithmus (stochastische Suche) müssten jedoch noch weit mehr Versuche (Monte-Carlo-Simulation) herangezogen werden, um seine statistischen Eigenschaften aufdecken zu können. Pro Parametersatz sollten es (Faustregeln folgend) $\gg 30$ Versuche sein. Zusammen mit der sehr langen Laufzeit[98] des Verfahrens ist dementsprechend keine umfassende Analyse möglich.

9.2.1 Startwerte und Parameter

Weder für das Original noch für dessen Variante NEAT4J sind Vorschläge für (problemspezifische) Startwerte bekannt. Aus diesem Grund werden bei der Suche nach geeigneten Parametern viele unterschiedliche Kombinationen durchprobiert. Es zeigt sich aber, dass die aus den Beispielen des NEAT-Algorithmus abgeleiteten Startwerte zusammen mit denen aus NEAT4J-Beispielen einen relativ guten Ausgangspunkt in Bezug auf den Benchmark bilden.

[97] Diesbezüglich suchen die Abschnitte 6.5 und 6.4 nach Eigenschaften und Verhalten von Struktur(en).

[98] Die Implementierung besitzt ein sehr schlechtes Laufzeitverhalten, was die Folge der gewählten Datenstruktur (ein einfacher Vektor von Genen) zur Repräsentation der Netzwerke ist. Die Analyse gründet deshalb insgesamt auf weniger als 60 Versuchen.

Parameter	Bedeutung	Werte
PopSize	Bevölkerungsgröße (Anzahl an Individuen)	$\mathbb{N}$
FeatureSelection	Keine voll verbundenen initialen Netzwerke? Deaktivierung von Verbindungen erlaubt?	$\{0, 1\}$
PAddLink	Wkt. für das Hinzufügen einer Verbindung	$[0, 1]$
PAddNode	Wkt. für das Hinzufügen eines Neurons	$[0, 1]$
PMutation	Wkt. für die Mutation einer einzelnen Sigmoidkonstante sowie Wkt. für die Veränderung eines einzelnen Gewichtes	$[0, 1]$
MaxPerturb	Obere Schranke für Veränderungen einer Sigmoidkonstante sowie obere Schranke für Veränderungen eines Verbindungsgewichtes	$\mathbb{R}$
PMutateBias	Wkt. für die Veränderung des Biasgewichtes eines Neurons	$[0, 1]$
MaxBiasPerturb	Obere Schranke für Veränderungen eines Biaswertes	$\mathbb{R}$
PToggleLink	Wkt. für die Deaktivierung einer einzelnen Verbindung	$[0, 1]$
PWeightReplaced	Wkt. für das Ersetzen eines einzelnen Verbindungsgewichtes $[-1, 1]$	$[0, 1]$
SurvivalThreshold	Prozentsatz der Besten einer Spezie, die eine Epoche überleben	$[0, 1]$
TerminationValue	Zielfitnesswert als Stoppkriterium	$\mathbb{R}$

Tabelle 19: NEAT-Parameter des evolutionären Prozesses

Die Tabellen 19 und 20 listen alle Parameter auf, von denen insgesamt 15 für den Vergleich mit dem EANT-Algorithmus berücksichtigt werden. In NEAT kann dabei zwischen voll verbundenen und nicht voll verbundenen initialen Netzwerken, jeweils ohne verdeckte Schicht, ausgewählt werden. Letztere enthalten genau eine zufällige Verbindung zu jedem Ausgabeneuron. Hinzu kommt in NEAT4J die Vorgabe, dass nur bei der Wahl voll verbundener Netzwerke Verbindungen nicht deaktiviert werden können.

Die verwendeten Parameter bzw. Startwerte werden samt ihren Ergebnissen im Anhang (Seite 123) aufgeführt und die maximale Laufzeit der jeweils parallel durchgeführten Versuche[99] einer Testreihe zumeist in der Tabellenbeschreibung[100] vermerkt. Tabelle 21 listet die im Folgenden untersuchten Parameter zusammen mit den für sie verwendeten Werten auf. Nicht untersuchte Parameter werden auf den am häufigsten anzutreffenden Wert aus den Beispielen von NEAT und NEAT4J gesetzt. Die Abbruchbedingungen des NEAT4J-Algorithmus werden nicht explizit untersucht, stattdessen wird ein Versuch spätestens nach $1e5$ Generationen (Epochen) beendet.

[99] Es werden Pentium 4 Prozessoren mit aktivem Hyperthreading und 3Ghz Takt benutzt.

[100] Die teilweise springende zeitliche Abfolge der Datumsangaben kommt durch die Wiederholung einzelner Tests zu Stande.

Parameter	**Bedeutung**	**Werte**
SpecieCount	Gewünschte Anzahl an Spezien	$\mathbb{N}$
MaxSpecieAge	Maximales Alter einer Spezie in Epochen	$\mathbb{N}$
SpecieYouthThreshold	Obere Altersgrenze für junge Spezien	$\mathbb{N}$
SpecieAgeThreshold	Untere Altersgrenze für alte Spezien	$\mathbb{N}$
YouthBoost	Fitnessmultiplikator für junge Spezien (1.0 ist neutral)	$\mathbb{R}^+$
AgePenalty	Fitnessmultiplikator für alte Spezien (1.0 ist neutral)	$\mathbb{R}^+$
DisjointCoeff	Kompatibilitätskoeffizient für die Anzahl unterschiedlicher Gene	$\mathbb{R}^+$
ExcessCoeff	Kompatibilitätskoeffizient für die Differenz in der Genanzahl	$\mathbb{R}^+$
WeightCoeff	Kompatibilitätskoeffizient für die Summe der Gewichtsdifferenzen	$\mathbb{R}^+$
CompatabilityChange	Veränderung des Kompatibilitätskoeffizienten	$\mathbb{R}^+$
Threshold	Kompatibilitätskoeffizient	$\mathbb{R}^+$

Tabelle 20: NEAT-Parameter der Speziation

Parameter	**Wert**	**Parameter**	**Wert**
PopSize	30, 150, 180, 1500	SpecieCount	1, 5, 15
FeatureSelection	ja, nein	MaxSpecieAge	500, 100000
PAddLink	0.0025, 0.025, 0.25	SpecieYouthThreshold	10
PAddNode	0.00125, 0.0125, 0.125	SpecieAgeThreshold	80
PMutation	0.25, 0.5	YouthBoost	1.0, 8.8
MaxPerturb	0.5	AgePenalty	1.0, 1.2
PMutateBias	0, 0.25	DisjointCoeff	1, 4
MaxBiasPerturb	-	ExcessCoeff	1, 4
PToggleLink	0, 0.01, 0.0001, 0.0025	WeightCoeff	2, 3
PWeightReplaced	0.85	CompatabilityChange	0.1, 3
SurvivalThreshold	0.3	Threshold	0, 1
TerminationValue	0		

Tabelle 21: Parameterwahl für den Test von NEAT4J

9.2.2 Bemerkungen

Entsprechend der Konstruktion des NEAT-Algorithmus ist zu erwarten, dass die stochastische Gewichtsoptimierung der numerischen des EANT-Algorithmus unterliegt. Um daher die Optimierung der Gewichte zu forcieren, werden die Wahrscheinlichkeiten der strukturellen Mutationen niedrig (z.B. PAddLink 0.0025 und PAddNode 0.00125) und die Mutationswahrscheinlichkeiten für Gewichte hoch (z.B. PMutation 0.25 und PWeightReplaced 0.85) angesetzt. Dadurch wird auch im Hinblick auf das Ziel kleiner Ergebnisnetzwerke (siehe Kapitel 4) die Gewichtsoptimierung gegenüber einer Erweiterung der Netzstruktur begünstigt.
In Bezug auf eine grafische Darstellung führt die stochastische Suche des Verfahren jedoch zu wenigen und über die Versuche zufälligen Verbesserungen im Evolutionsablauf, so dass die Angabe von Effizienzpunkten keine Tendenzen erkennen läßt und diese Abbildungen deshalb vernachlässigt werden.
In Vorversuchen werden ferner Verbindungen zum Biasneuron zugelassen, welches in den folgenden Tests allerdings aus zwei Gründen fehlt. Zum einen ist wie in Kapitel 7 konstatiert die gestellte Aufgabe ohne Biasneuron lösbar, zum anderen werden in den Vortests fast nur Biasverbindungen durch den Algorithmus verändert, ohne dass sich der Fitnesswert signifikant verbessert. Zudem führt die Existenz von Biasverbindungen schnell zu einer permanenten Stagnation, woraufhin ein Versuch dann nach ein paar tausend Generationen ohne Fortschritt manuel beendet wird.
Als abschließende Bemerkung wird noch einmal darauf hingewiesen, dass die gemachten Aussagen über NEAT zum Großteil auf einzelnen oder sehr wenigen Testläufen gründen und somit nicht dem Wesen eines stochastischem Algorithmus gerecht werden können! Da dies eine Zeitproblematik und diese dem Algorithmus inhärent ist, kann sie nicht gelöst oder umgangen werden.

9.2.3 Ergebnisse

Das Ziel des Tests besteht darin, den besten Parametersatz bezüglich des Fitnesswertes unter der Nebenbedingung eines Netzwerkes mit möglichst wenig Struktur zu finden. Überraschenderweise erreicht der beste Versuch aber lediglich einen Fitnesswert von 0.2984.
Damit rückt die Betrachtung der (Nebenbedingung minimaler) Netzstruktur tendenziell in den Hintergrund, weil bereits das primäre Ziel verfehlt wird. Die Netzstruktur wird aber dennoch beachtet.
Außerdem werden aufgrund des relativ kleinen Versuchsdatensatzes des Öfteren nicht einzelne Parameter, sondern die Bündel Startnetze (FeatureSelection), Speziation (SpecieCount, YouthBoost, AgePenalty, DisjointCoeff, ExcessCoeff, WeightCoeff, CompatabilityChange), Bevölkerungsgröße (PopSize), Struktur- (PAddLink, PAddNode, PToggleLink) und Gewichtsmutation (PMutation) betrachtet.

Initiale Netzwerke

In den (Vor-) Versuchen (Parameter und Ergebnisse siehe Tabellen 36 und 48 im Anhang) werden sowohl voll verbundene als auch nicht voll verbundenen initia-

le Netzwerke (FeatureSelection) und verschiedene andere Parameter variiert. Es ist zu erkennen, dass nicht voll verbundene initiale Netzwerke fast immer bessere Ergebnisse zur Folge haben als voll verbundene. Für dieses Verhalten mag die Eigenschaft, dass bei voll verbundenen Netzwerken Verbindungen nicht deaktiviert werden dürfen, mitverantwortlich sein. Diese ist jedoch auch im Fall nicht voll verbundener Netzwerke aufgrund einer sehr kleinen Wahrscheinlichkeit (PToggleLink 0.0001) beinahe abgeschaltet. Somit liegt die Vermutung nahe, dass die voll verbundenen Netzwerke zu viele Verbindungen enthalten, als dass der NEAT-Algorithmus diese korrekt optimieren könnte. Entsprechend ist zu erwarten, dass er tendenziell große Netzwerke erzeugt, um die ungenügende Optimierung der Gewichte durch mehr Struktur zu kompensieren. Dieses Verhalten wird aber nicht mit Sicherheit nachgewiesen. Der Test zeigt jedoch eindeutig, dass für den Benchmark die Wahl nicht voll verbundener initialer Netzwerke am besten ist.

Bevölkerungsgröße

Für die analysierten Bevölkerungsgrößen 30 und 150 sowie 1500 in einem weiteren Test, enthalten die Tabellen 38, 40 und 42 im Anhang Testreihen, in denen u.a. für verschiedene Parametersätze die Performanz unterschiedlicher Bevölkerungsgrößen zu sehen ist.

Auffällig ist, dass die besten und nahezu gleichen Fitnesswerte für die Wahl einer einzigen Spezie erreicht werden. Entsprechend schwierig ist es, die optimale Anzahl an Individuen zu bestimmen. Werden allerdings die Fitnesswerte pro Individuum ($\frac{\text{Fitness}}{\text{Bevölkerungsgröße}}$) betrachtet, so erzielen 150 Individuen die höchste Effizienz. Legt man hingegen die Anzahl der Funktionsauswertungen ($\frac{\text{Fitness}}{\text{Funktionsevaluationen}}$) zu Grunde, so sind Bevölkerungen mit 30 Individuen effizienter.

Der Annahme folgend, dass eine evolutionäre Vielfalt, also eine größere Bevölkerung, prinzipiell gut ist, kann demnach o.B.d.A. die Individuenanzahl bei zeitkritischen Versuchen gering, sonst eher groß, gehalten werden. Aus diesem Grund wird für nachfolgende Parametertests zumeist nur eine fixe Bevölkerungsgröße angesetzt.

Speziation

Die analysierte Anzahl von Spezien ist 1, 5 und 15. Die Tabellen 38, 40 und 42 im Anhang enthalten die Testreihen, in denen u.a. für verschiedene Parameter die Performanz unterschiedlicher Spezienanzahl und -parameter berechnet wird.

Für eine fixierte Bevölkerungsgröße zeigen die Querschnittsdaten und -statistiken in Tabelle 22 klar, dass eine Aufteilung der Bevölkerung in Spezien das Ergebnis verschlechtert. Erst wenn die einzelnen Spezien so viele Mitglieder haben wie im Test mit genau einer Spezie, erreicht ein Versuch ungefähr denselben Fitnesswert.

Ferner zeigen die Testreihen in Tabelle 40 (AgePenalty 1.0) und Tabelle 42 (AgePenalty 1.0 und YouthBoost 1.0) bezüglich des Ausgangstests in Tabelle 38, dass eine Bevor- oder Benachteiligung von jungen oder alten Individuen ceteris paribus ebenfalls eine negative Wirkung auf das Ergebnis hat.

Für den Test einer versuchten starken Annäherung von NEAT an EANT wird außerdem zusätzlich eine weitere Versuchsreihe mit gänzlich neuen Parameterwerten

Quelle \ Bev.-Größe	**30**		**150**		**1500**	
	Fitness	Netzgröße	Fitness	Netzgröße	Fitness	Netzgröße
1 Spezie						
Testreihe 1 (Tabelle 38)	0.3173	17	0.314	16	0.3143	12
Testreihe 2 (Tabelle 40)	0.3160	18	0.3135	12	0.313	9
Testreihe 3 (Tabelle 42)	0.366	18	0.314	11	0.3367	12
Mittelwert	0.3331	17.66	0.3138	13	0.3215	11
Standardabweichung	0.0164	0.5773	2.886e-4	2.6457	0.0131	1.732
5 Spezien						
Testreihe 1 (Tabelle 38)	0.3952	11	0.3441	14		
Testreihe 2 (Tabelle 40)	0.3801	23	0.3143	16		
Testreihe 3 (Tabelle 42)	0.3161	22	0.3275	20		
Mittelwert	0.3638	18.66	0.3286	16.66		
Standardabweichung	0.0419	6.6583	0.0149	3.055		
15 Spezien						
Testreihe 1 (Tabelle 38)			0.5402	14	0.3498	13
Testreihe 2 (Tabelle 40)			0.4044	14	0.3476	28
Testreihe 3 (Tabelle 42)			0.3549	16	0.3146	19
Mittelwert			0.4331	14.66	0.3373	20
Standardabweichung			0.0959	1.1547	0.0197	7.5498

Tabelle 22: Ergebnisse der Speziationsversuche

ausprobiert. Tabelle 46 zeigt deren Parameterwahl und Ergebnisse. Deutlich ist zu sehen, dass die verwendete Parameterisierung der Speziation zu signifikant kleineren Ergebnisnetzen, jedoch (vielleicht auch deswegen) zu keiner Verbesserung des Fitnesswertes führt.
Diesen Ergebnissen zur Speziation Rechnung tragend, ergibt ein nun Validierungstest mit 150 Individuen und 15 Spezien (siehe Tabelle 44), in dem abermals keine besseren Ergebnisse als mit einer einzelnen Spezie sichtbar werden, dass die Speziation abgeschaltet bzw. auf eine Spezie reduziert werden sollte. Auch der große Laufzeitzuwachs pro weiterer Spezie spricht praktisch gegen deren Einsatz, zumindest in der vorliegenden Implementierung.

Strukturmutationen

Zur Beschleunigung der Testdurchläufe wird o.B.d.A. die Bevölkerungsgröße auf 30 gesetzt. Die untersuchten Paare aus Mutationswahrscheinlichkeiten (PAddLink / PAddNode) sind 0.0025/0.00125, 0.025/0.0125 und 0.25/0.125. Wie bereits in den Bemerkungen ausgeführt, soll der NEAT-Algorithmus dazu gebracht werden, mehr Optimierungsaufwand für die Gewichte als für die Struktur zu betreiben. Diese Forderung entspricht den kleinsten untersuchten Wahrscheinlichkeiten, und sie wird durch die in Tabelle 48 zusammengefassten Resultate untergestützt. Anschau-

lich ist erkennbar, wie die Netzgröße ansteigt und der Fitnesswert stagniert, wenn strukturellen Mutationen mehr Priorität eingeräumt wird. Kleinere Wahrscheinlichkeiten für Strukturerweiterungen sind also gegenüber größeren zu bevorzugen.
Ein weiterer Test zur Deaktivierungswahrscheinlichkeit einer Verbindung (PToggleLink) mit den Werten 0.0001 und 0.01 zeigt darüberhinaus, dass ein mäßig kleiner Wert an Stelle eines Wertes nahe Null zwar kaum eine Verbesserung des Fitnesswertes ergibt, jedoch die Netzgröße merklich reduziert. Der Grund hierfür ist die starke Auswirkung dieser betragsmäßig relativ kleinen Änderung auf die Wahrscheinlichkeit p, dass keine Verbindung deaktiviert wird:

$$p = (1 - PToggleLink)^{\#Verbindungen} \tag{14}$$

Bei 10 Verbindungen und PToggleLink = 0.0001 folgt $p = 0.999$, bei PToggleLink = 0.01 hingegen $p = 0.904$. Für eine wachsende Anzahl an Verbindungen konvergiert Gleichung (14) jedoch immer gegen $\frac{1}{e} = 0.3678$. Dennoch ist anhand der Ergebnisse ein moderat von Null verschiedener Parameterwert positiv zu bewerten.

Gewichtsmutationen

O.B.d.A. wird die Bevölkerungsgröße auf 150 gesetzt, um der zur Gewichtsoptimierung verwendeteten stochastischen Suche etwas mehr Spielraum zu zugestehen. Untersucht werden die Wahrscheinlichkeiten 0.25 und 0.5 für die Veränderung eines Gewichtes (PMutation).
Weitere ebenfalls relevante Parameter werden vorerst nicht variiert, da diese nicht an der Entscheidung, ob eine Mutation stattfindet, beteiligt sind, sondern diese gestalten. Tritt ein Mutationsereignis ein, wird das Gewicht entweder (PWeightReplaced) durch eine gleichverteilte Zufallszahl im Intervall $[-1, 1]$ ersetzt oder ihm wird ein gleichverteilt gezogener Wert aus dem Intervall $[-\text{MaxPerturb}, \text{MaxPerturb}]$ hinzu addiert. Bei 10 Gewichten und einer Gewichtsveränderungswahrscheinlichkeit von 25% ergibt Gleichung (14) eine Wahrscheinlichkeit für keine Gewichtsveränderung von 5.6313%. Für 50% folgen entsprechend 0.0977%.
Die Tabellen 50 und 51 enthalten die Daten der beiden Testreihen, während Tabelle 23 deren Mittelwert und Standardabweichung veranschaulicht. Knapp und statistisch uneindeutig schneidet die kleinere Mutationswahrscheinlichkeit besser ab. Betrachtet man die schlechte Konditionierung der neuronalen Funktion, so sind kleinere Wahrscheinlichkeiten vorzuziehen, da sie weniger Störungen der Funktion darstellen und ein gutes Netzwerk weniger wahrscheinlich kaputt machen.

Quelle \ Bevölkerungsgröße	**Testreihe 1 (Tab. 50)**	**Testreihe 2 (Tab. 51)**
Stichprobenanzahl	7	14
empirischer Mittelwert	0.308	0.3139
empirische Standardabweichung	0.0057	0.0053

Tabelle 23: Ergebnisse der Versuche zur Gewichtsmutation

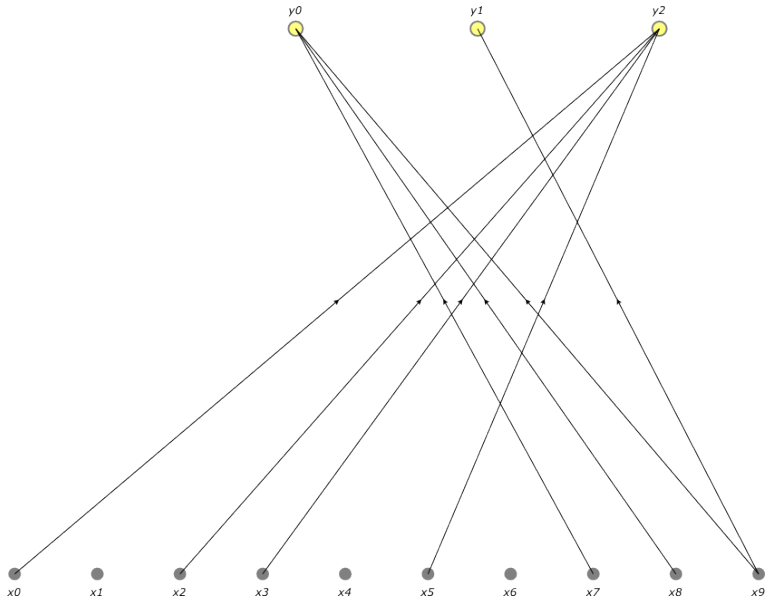


Abbildung 20: Netzwerk ohne verdeckte Schicht

9.2.4 Zusammenfassung

Die sehr einfache Datenstruktur zur Darstellung neuronaler Netzwerke wirkt sich außerordentlich negativ auf das Laufzeitverhalten zahlreicher Funktionen aus. Diese ist im Kontext zu Kapitel 3 unnötig schlecht.

Trotz der hohen Laufzeit wird die zentrale Aufgabe, mit möglichst kleinen Netzwerken den optimalen Fitnesswert zu erreichen, von NEAT aber nicht gelöst. Nur ein einziges der entwickelten Netzwerke erreicht einen Fitnesswert von knapp unter 0.3, welcher bereits mit einem Netzwerk ohne verdeckte Schicht realisierbar ist (siehe Abbildung 20 mit einem Fitnesswert von 0.313575).

Zudem zeigen die Versuche für eine jeweils verschiedene Anzahl von Spezien als einzigen Unterschied einen extremen Anstieg der Laufzeit, ohne dass bessere Fitnesswerte erzielt werden können. Es wird daher empfohlen, keine Speziation bzw. genau eine Spezie zu verwenden. Der evolutionäre Ansatz der Speziation, erweist sich in NEAT als nicht brauchbar. Mit Blick auf EANT, in welchem ebenfalls nur eine einzige Spezie existiert, erhöht dies praktischerweise die Vergleichbarkeit.

Außerdem wählt NEAT zum Einfügen eines neuen Neurons eine zufällig bestehende Verbindung aus, in deren Mitte das Neuron gefügt wird. Dadurch werden Neuronen mit vielen Verbindungen bevorzugt und im Evolutionsverlauf entstehen neue Strukturen wahrscheinlicher in Netzregionen mit viel als mit wenig Struktur (Verbindungen). Würden Neuronen stattdessen (wie in EANT) zufällig ausgewählt, so könnte diese Verzerrung der Wahrscheinlichkeit, eine neue Struktur (ein neues Neuron inkl. Verbindungen) an einer bestimmten Stelle hinzuzufügen, verhindert werden.

Schlussendlich stellt sich ferner die Frage, ob die stochastische Suche, auf der NEAT

und eine Vielzahl evolutionärer Algorithmen basieren, ein geeignetes Mittel darstellt, um generell genügend komplexe Lösungen bzw. neuronale Topologien entwickeln zu können. Die erzielten Ergebnisse zumindest der Variante NEAT4J sprechen für sich.

9.3 Test: EANT

Für die Testreihen des EANT-Algorithmus (Revision 326) wird eine SuSE-Linux-Umgebung auf einem Verbund von Pentium 4-Rechnern eingesetzt, der über die Software PVM eine parallele Verarbeitung des Algorithmus ermöglichen.
Analysiert wird das Zusammenspiel von Strukturerzeugung und CMA-ES-basierter Gewichtsoptimierung im Evolutionsprozess. Die Parameter der Strukturentwicklung bleiben dabei vorerst auf Werte fixiert, die sich während der bis dato erfolgten Entwicklung des Algorithmus als gut erwiesen. Andernfalls können - analog wie im Fall des stochastischen NEAT-Algorithmus - aufgrund der trotz Parallelisierung noch immer sehr hohen Laufzeit nicht genügend Versuche[101] durchgeführt werden, um auch für diese aussagekräftige Schlüsse zu ziehen.
Die folgenden Ausführungen beziehen sich ferner vornehmlich auf diejenigen Parameter der Gewichtsoptimierung, die normalerweise zwar simultan, aber getrennt von einem Optimierungsverfahren, betrachtet werden. Dies sind gängigerweise der Startvektor und die Abbruchbedingungen, die als (Meta-) Parameter nicht direkt zum iterativen Optimierungsverfahren, sondern zur zugrunde liegenden Iterationsprozess gehören.
Im Gegensatz zur statisch auf ein Neuronales Netzwerk fixierten Untersuchung von CMA-ES in Abschnitt 8.1 erfolgt die Analyse insbesondere nun auf Grundlage der im Evolutionsprozess dynamisch entstehenden Netzwerke.

9.3.1 Startwerte und Parameter

Als initiale Netzwerke werden etwas mehr als halbvoll verbundene verwendet und die Bevölkerungsgröße auf 30 festgelegt. Die prinzipiell möglichen Parameter des EANT-Algorithmus - ohne die relativ willkürliche Parameterisierung der Erzeugung eines zufälligen Teilnetzes (bestehend aus einem einzelnen Neuron mit einigen Verbindungen) - sind in Tabelle 24 aufgeführt. Bei den fixierten Parametern handelt es sich um die Mutationswahrscheinlichkeit ein Teilnetz hinzuzufügen (70%), die Mutationswkt. eine Biasverbindung einzufügen (5%), die Mutationswkt. des Löschens einer Verbindung (25%), die relative Mindestverbesserung (0.005; um von Modus B zu Modus A zu wechseln), die Anzahl der an verschiendenen Stellen zu verwendenden Netzkopien (bei Gewichtsoptimierung 2 Kopien in Modus A bzw. 3 Kopien in Modus B; bei Strukturerzeugung stets 3 Kopien) und die Reinitialisierungsgewichte (1.0 für Neuronen und 0.125 sonst; in Modus B).
Die ebenfalls fixierten CMA-ES-Parameter (nullgesetzte Startgewichte, $\sigma_0 = 0.1$, popSize = 85) sind Abschnitt 8.1 entnommen, während die nicht genannten weiter-

[101] Die Folgende Analyse zu EANT gründet auf dessen Evolutionsverhalten in 3 Versuchen.

hin den problemdimensionsabhängigen Standardvorgaben des CMA-ES-Algorithmus überlassen werden.

Parameter	**Bedeutung**	**Werte**
popSize	Bevölkerungsgröße	$\mathbb{N}$
numberOfWeightCopiesA	Anzahl der anzufertigenden Kopien eines Individuums bei der Gewichtsoptimierung in Modus A	$\mathbb{N}$
numberOfWeightCopiesB	Anzahl der anzufertigenden Kopien eines Individuums bei der Gewichtsoptimierung in Modus B	$\mathbb{N}$
numberOfStructureCopies	Anzahl der anzufertigenden Kopien eines Individuums bei der Strukturoptimierung	$\mathbb{N}$
pAddRandomSubNetwork	Hinzufügen eines zufälligen Teilnetzwerkes	$[0, 1]$
pAddRandomBias	Zufälliges Hinzufügen einer Biasverbindung	$[0, 1]$
pRemoveRamdonConnection	Zufälliges Entfernen einer Verbindung	$[0, 1]$
pAddRandomConnection	Zufälliges Hinzufügen einer Verbindung (geplant)	$[0, 1]$
delta	Relative Mindestverbesserung des Fitnesswertes für den Wechsel zu Modus A, sonst B	$[0, 1]$
wNeuronModeB	Reinitialisierungsgewicht für Neuronen in Modus B	$\mathbb{R}$
wBoundModeB	Intervallgrenze des Reinitialisierungsgewichtes für Nichtneuronen in Modus B	$\mathbb{R}$
initWeightOptimization	Startwertwahl für den CMA-ES-Algorithmus	{orig., null, random}
maxIterations	Maximale Anzahl an Iterationen für CMA-ES	$\mathbb{N}$
λ, σ_0, μ, w, c_c, c_s, d_s, c_{cov}, μ_{eff}, μ_{cov}	CMA-ES-Parameter, siehe Abschnitt 5.2	

Tabelle 24: Parameterwahl für den Test von EANT

9.3.2 Bemerkungen

Die im Voraus zu dieser Arbeit durchgeführten Versuche benutzen sowohl unverbundene als auch voll verbundene initiale Netzwerke. Die derzeitige Wahl von zufällig erzeugten halbvoll verbundenen Startnetzwerken ist einerseits heuristisch begründet, andererseits werden keine voll verbundenen Startnetzwerke eingesetzt, weil der Übergang von diesen zu andersartigen Netzen schwieriger zu sein scheint, als letztere ohne Strukturvorgabe zu entwickeln. Dies deutet allerdings auf ein Problem des Suchschemas (der Markovkette) im diskreten Raum der Netzstruktur hin. Ungeachtet dessen weisen voll verbundene Netzwerke bzgl. des Benchmarks durchweg schlechte Resultate auf.
Ferner wird die eher kleine Bevölkerungsgröße von 30 relativiert, wenn man einerseits die Verdoppelung der Bevölkerungsgröße in der Mutationsphase und andererseits die Verdoppelung oder Verdreifachung der Individuenanzahl für die Optimierungsphase berücksichtigt. Aus den anfänglich 30 Individuen werden dabei je nach

Modus temporär 60 oder 90 bei der Gewichtsoptimieung und permanent 60 bei der Strukturentwicklung. Dieses Vorgehen ist zwar unorthodox, scheint aber ansonsten keine negativen Auswirkungen zu haben.

9.3.3 Ergebnisse

Das Ziel des Tests besteht darin, anhand der Parameter initiale Gewichtswahl und maximale Iterationsanzahl des Gewichtsoptimierungsverfahrens effizienzsteigernde Möglichkeiten für den Evolutionsprozess zu entdecken[102]. Im Gegensatz zu CMA-ES besitzt EANT selbst bislang nämlich - außer für einen Fitnesswertes von 0 - keine Abbruchbedingungen.
Die Versuchsreihen, deren Ergebnisse im Folgenden interpretiert werden, basieren auf zwei Tests mit maximal 1500 bzw. 25000 CMA-ES-Iterationen (EANT-1500 und EANT-25000) mit 4570 bzw. 2624 Optimierungsergebnissen[103] sowie einem weiteren Versuch (EANT-SR) auf Grundlage des SR-Algorithmus mit 2760 Einzeloptimierungen[104] bei maximal $1e6$ Iterationen. Die Betrachtung des alternativen Verfahrens zur Gewichtsoptimierung geschieht dabei nach den Erkenntnissen von Kapitel 8.1 nur interessehalber.

Initiale Gewichte

Wie Abschnitt 8.1 aufdeckt, sind nullgesetzte Startwerte die beste Wahl für eine erneute Optimierung desselben Netzwerkes. Die Informationen über Netzgewichte früherer Evolutionsschritte sollten also vernachlässigt und für die Optimierung der aktuellen Generation nicht verwendet werden. Da für Mutationen dieser Netze die A-priori-Information über alte Gewichte zusätzlich ungültig wird, kann die Aussage auf Mutationen eines bereits optimierten Netzwerkes übertragen werden.
Die flache Umgebung des Nullvektors als Startpunkt ist also der schroffen Umgebung eines bekannten Minimums, wie sie in Abbildung 17 zu sehen ist, vorzuziehen. Dies zeigt zudem, dass der CMA-ES-Algorithmus nicht startwertunabhängig ist, was eine wünschenswerte Eigenschaft wäre.

Generationslängen

Die Abbildungen 21, 22 und 23 zeigen die Effizienzpunkte für Funktionsevaluationen und Fitnesswert dreier EANT-Evolutionsprozesse. Diese beginnen jeweils mit kleinen, schlechten Netzwerken und entwickeln diese zu größeren, besseren Netzen. In den Darstellungen ist daher der Evolutionsfortgang, der sich in einer nach rechts gerichteten Streuung der Punkte manifestiert, in Form unterschiedlicher Netzstruk-

[102] Insgesamt beläuft sich die Zahl der untersuchten EANT-Parameter damit auf 4, die der fixierten hingegen auf 18.
[103] Diese beiden EANT-Versuche EANT-1500 bzw. EANT-25000 dauern parallelisiert auf 7 bzw. 8 Rechner (Pentium 4, 3Ghz) 13 bzw. 23 Tage und ergeben knapp 26 bzw. etwas mehr als 15 EANT-Generationen.
[104] Der Test von EANT-SR dauert parallelisiert auf 3 Rechner (Pentium 4, 3Ghz) 20 Tage und erreichte fast 16 EANT-Generationen.

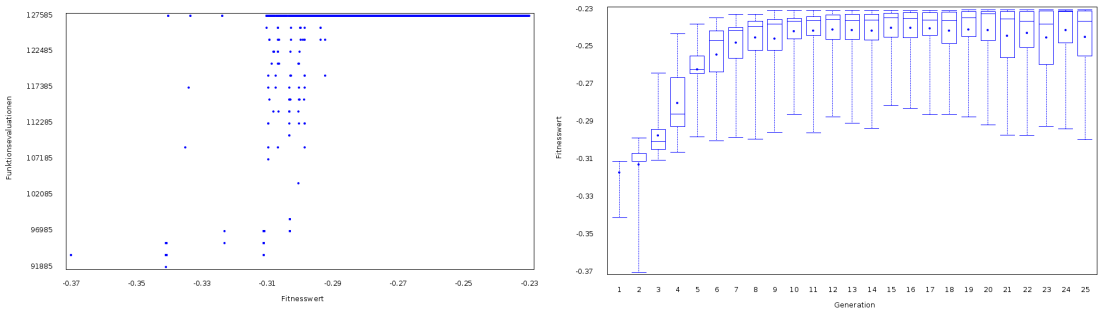


Abbildung 21: Funktionsevaluationen und Fitnesswerte der Testreihe EANT-1500

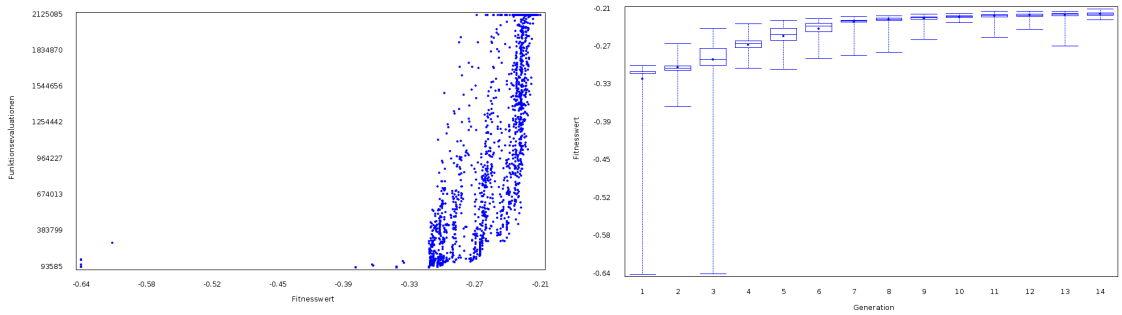


Abbildung 22: Funktionsevaluationen und Fitnesswerte der Testreihe EANT-25000

turen enthalten. Die Grafiken[105] der Effizienzpunkte geben deshalb einen Überblick über den Evolutionsverlauf, wohingegen die Boxplots diesen detailliert in Form der üblichen Statistiken zusammenfassen. Zusätzlich stellt Abbildung 24 die Testreihen anhand ihren wesentlichsten Kennzahlen vergleichend dar.
Besonders in Abbildung 24 ist dabei die erwartungsgemäß positive Korrelation zwischen Evolutionsdauer und Fitnesswert sowie die Dominanz von CMA-ES gegenüber der Alternative SR zu sehen. Zudem zeigt sich die Dominanz einer langen (maximal 25000 Zyklen) Generationen bei der evolutionären Gewichtsoptimierung gegenüber einer kürzeren (maximal 1500 Zyklen) und des alternativen Optimie-

[105] Die Abbildung enthält alle 180 Netzwerke einer Generation und nicht nur die 30 (vordefinierte Bevölkerungsgröße) besten.

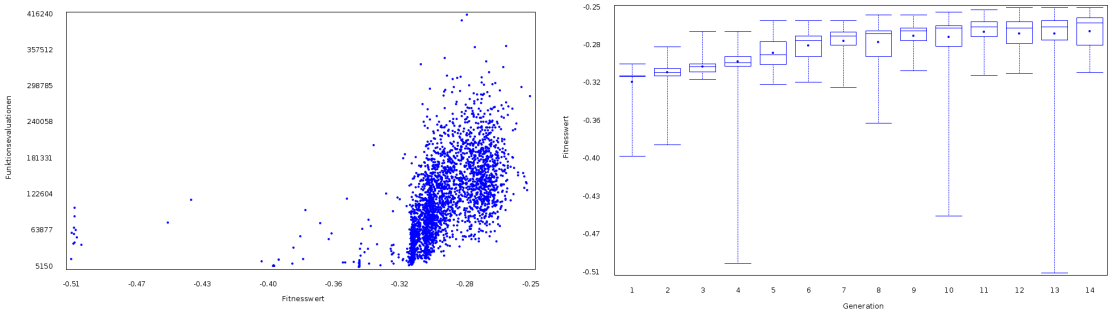


Abbildung 23: Funktionsevaluationen und Fitnesswerte der Testreihe EANT-SR

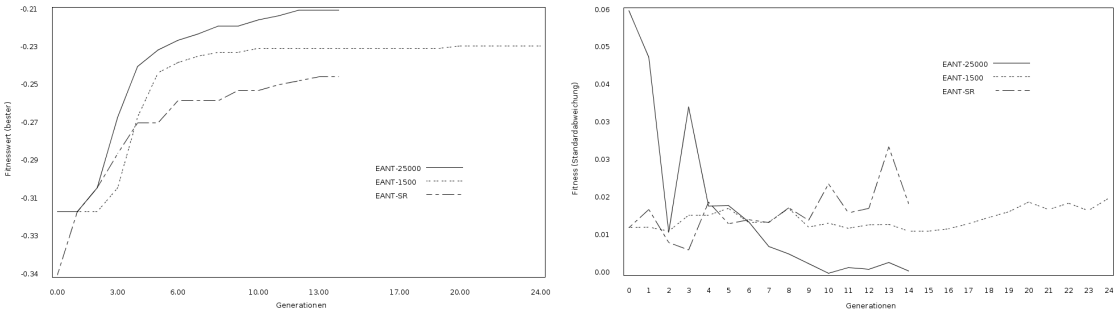


Abbildung 24: Bestes Individuum und Streuung einer EANT-Generation

rungsverfahrens (das maximal 416240 Zyklen benutzt). Während der CMA-ES-Algorithmus relativ häufig innerhalb von 25000 Generationen konvergiert, offenbart Abbildung 21 ferner die Begrenzung der maximalen Iterationsanzahl auf 1500 als Fortschrittshemmnis, und eine Konvergenz tritt nur noch selten ein. Ein hoher Aufwand bei der Optimierung der Gewichte ist also vorteilhaft. Das entsprechende Ergebnis der statischen Analyse aus Abschnitt 8.1 kann damit ebenfalls auf die dynamische Evolution übertragen werden.

Konvergenz

Für den Versuch mit maximal 25000 CMA-ES-Generationen fällt das Zurückgehen der Varianz mit fortschreitender Evolution in den Abbildungen 22 und 24 auf. Was einerseits als positiv zu bewertende Konvergenz des Verfahrens interpretierbar ist, sollte andererseits jedoch nicht weit entfernt von einer optimalen Lösung passieren. Die beiden anderen Versuchsreihen weisen diesen Effekt aufgrund einer zu geringen Qualität der dort verwendeten Gewichtsoptimierung nicht so deutlich auf. Werden allerdings nur die 30 besten Individuen einer Generation betrachtet, siehe Abbildung 37 im Anhang, so tendiert auch dort die Varianz gegen Null.
Für dieses Kollabieren der Varianz sind unterdessen mehrere Ursachen möglich. Als die wahrscheinlichsten zeichnen sich dafür jedoch die folgenden beiden ab. Aus Sicht der Gewichtsoptimierung könnte dieser Effekt auf die allgemeinen (und die numerischen) Probleme bei der Funktionsoptimierung von chaotischen Funktionen, die einer Verbesserung des Fitnesswertes irgendwann entgegen stehen und eine Art Barriere bilden, zurückführbar sein. Aus Sicht der Strukturentwicklung könnte dieser aber auch ausgelöst werden, wenn im Evolutionsverlauf zunehmend Netze mit gleichen Eigenschaften entstehen.
Da der erste Punkt nicht direkt anhand der vorliegenden Daten gezeigt werden kann, wird versucht, den zweiten auszuschließen. Dazu veranschaulichen die Abbildungen 25, 26 und 27 die Strukturgrößen (Anahl der Verbindungen) der besten 30 - in der jeweils letzten berechneten Generation - auftretenden Netzwerke. Deutlich ist für jede Testreihe eine große Bandbreite zu erkennen. Einige exemplarisch ausgewählte Netzwerke der letzten Generation der besten Testreihe werden in Abbildung 38 im Anhang (Seite 133) darstellt. Somit liegt die Annahme nahe, dass eine schlechte Strukturentwicklung nicht unmittelbar der Grund für die schrump-

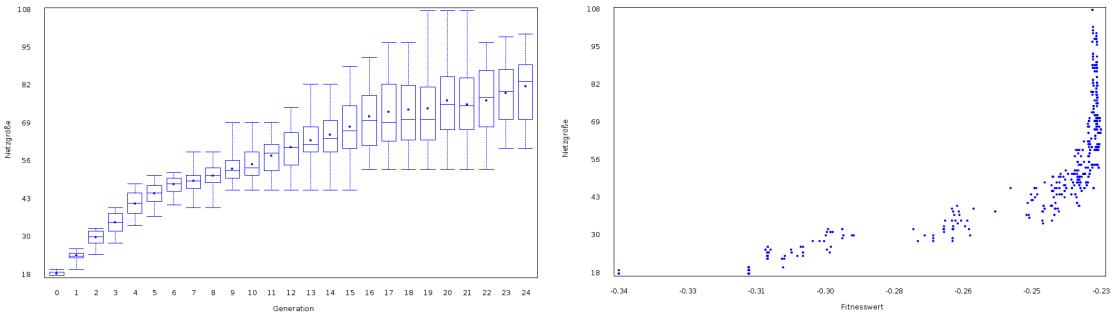


Abbildung 25: Netzgrößen und Fitnesswert der Testreihe EANT-1500

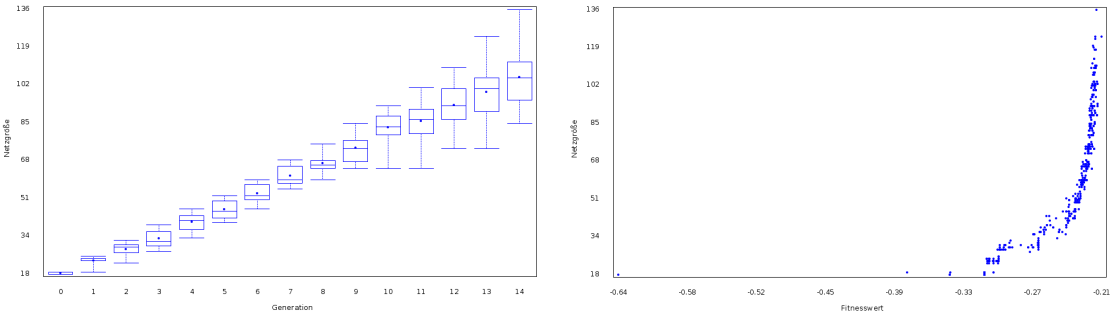


Abbildung 26: Netzgrößen und Fitnesswert der Testreihe EANT-25000

fende Varianz ist. In Anbetracht des unendlich abzählbar großen Suchraumes der Netzstruktur ist allerdings offen, wie Ähnlichkeit zwischen Netzstrukturen definiert werden sollte und wann Netzwerke gleiche Eigenschaften aufweisen.

Daneben zeigen die Scatterplots die Effizienzpunkte für Netzgröße und Fitnesswert für alle berechneten Generationen eines Versuchs. Die sichtbar werdenden Kurven erwecken den Eindruck, dass tatsächlich eine Barriere existiert. Der gegen unendlich tendierende Anstieg der Kurve entspricht dabei der tendenziellen Unabhängigkeit der beiden dargestellten Merkmale.

Abgesehen von den Ausführungen zur Gewichtsoptimierung in Kapitel 8, die bereits einige Schwierigkeiten bei der Entwicklung neuronaler Topologien aufzeigen, verhält sich die Strukturentwicklung - allem Anschein nach - ebenso komplex. Das aktuell verwendete Suchmuster für die Netzstruktur scheint eindeutig das Verfahren zu behindern. Aufgrund der Größe des Suchraumes wird jedoch erwartet, dass größere Netzwerkstrukturen die Grenze weiter in Richtung des optimalen Fitnesswertes verschieben. Der Strukturentwicklung - in Form der Dauer des Evolutionsprozesses - sollte dementsprechend mehr Zeit eingeräumt werden.

Außerdem muss bemerkt werden, dass die Kausalität zwischen Netzgröße und Fitnessstagnation nicht sichergestellt ist. Ebenso gut könnte die Netzgröße lediglich eine Begleiterscheinung (Scheinkorrelation) bzgl. einer anderen Eigenschaft eines Netzwerkes sein. Derartige Eigenschaften sind jedoch bislang noch nicht ausreichend erforscht.

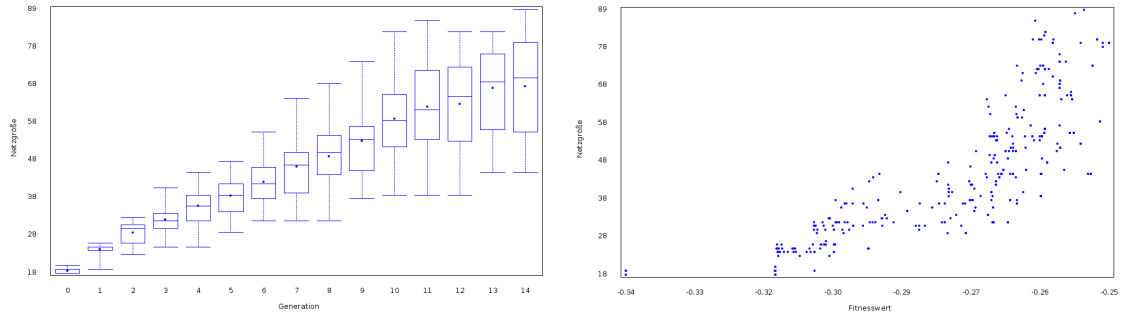


Abbildung 27: Netzgrößen und Fitnesswert der Testreihe EANT-SR

9.3.4 Zusammenfassung

Obwohl bis zur 15. Generation mehr als 3 Milliarden Funktionsevaluationen benutzt werden, erreicht das beste von EANT erzeugte Neuronale Netzwerk lediglich einen Fitnesswert von 0.2099. Der Algorithmus kann somit (in seiner bis dato entwickelten Form) die Testaufgabe nicht lösen. Die resultierenden, nicht trivialen Netze belegen aber, dass der Algorithmus funktioniert und zielgerichtet arbeitet. Er sollte jedoch in Bezug auf die Suche im Raum der Netzstruktur einige Erweiterungen und Verifikationen erfahren, die z.B. sicherstellen, dass in jedem Fall eine ausbalancierte Struktursuche erfolgt. Da ferner die unvermeidbare Vergrößerung der Netzstruktur eng mit dem Versagen eines Optimierungsverfahrens verknüpft zu sein scheint, wird die Begrenzung der Gewichtsoptimierung auf eine moderate (vielleicht dynamisierte) Anzahl an Funktionsevaluationen oder Iterationsschritten (Generationen) vorgeschlagen. Siehe dazu auch die Abbildungen 35, 36 und 37 (Seite 132), in denen Fitnesswert und Funktionsevaluationen nebeneinander im EANT-Evolutionsverlauf dargestellt sind. Parallel sollte außerdem die Suche nach einem geeigneteren Optimierungsverfahren für die Gewichte oder nach Verbesserungen an den bestehenden fortgesetzt werden.
EANT erscheint dennoch in Bezug auf die Gewichtsoptimierung relativ ausgereift zu sein, so dass Verbesserungsmöglichkeiten eigentlich nur bei der größtenteils unerforschten Strukturentwicklung zu sehen sind.

9.4 Baukastenexperimente

Aus Interesse und aufgrund der unbefriedigenden Leistungen, die NEAT und EANT in Bezug auf die gestellte Aufgabe erbringen, werden weitere Tests mit dem in Abschnitt 6.4 beschriebenen Baukasten zur Konstruktion evolutionärer Algorithmen für die Entwicklung Neuronaler Netzwerke durchgeführt.
Die konstruierten Algorithmen funktionieren größtenteils sehr ähnlich. Die Optimierung der Gewichte wird von einem numerischen Minimierungsalgorithmus auf Basis einer Blackboxoptimierung durchgeführt, und die Netzstruktur randomisiert verkleinert oder vergrößert. Für die Netzwerke werden ferner $(< \cdot, \cdot >, \tanh)$-Neuronen verwendet. Als Abbruchbedingung kommen das Erreichen von $1e8$ Funktionsaus-

wertungen und 0.0001 als Fitnesswert zur Anwendung. Zudem werden als Bevölkerungsgröße konstant 30 Individuen und als initiale Netzwerke unverbundene Netze ohne verdeckte Schicht eingesetzt.
Das Ziel der experimentellen Suche ist ein effizienter Algorithmus, der in weniger als zwei Tagen mit einem guten Ergebnis terminiert. Die verwendeten Algorithmen sind im Anhang (ab Seite 134) aufgeführt. Bei der Betrachtung der Fitness unterliegen die Versuchsalgorithmen durchgehend EANT. Manchmal schneiden sie dabei allerdings deutlich, meistens jedoch nur knapp besser besser als NEAT ab. Die Ergebnisse der experimentellen Versuche werden nicht eingehender vorgestellt, weil keiner der entworfenen Algorithmen das anvisierte Ziel erreicht.
Wegen der frappierenden Ähnlichkeit der Algorithmen zu EANT darf aber davon ausgegangen werden, dass eine längere Laufzeit, eine größere Bevölkerung und eine Parallelisierung zu besseren Ergebnissen führen würden. Vermutlich wäre dann sogar ein Großteil der Algorithmen erheblich besser als NEAT.
Insgesamt deckt sich die gewonnene Erkenntnis relativ gut mit der aus den EANT-Testreihen. Zum einen verläuft die Gewichtsoptimierung jeweils mit den typischen Problemen, zum anderen wirkt die (bzgl. Abschnitt 6.4 zumeist ausbalanciert) stochastische Strukturentwicklung in der Weite des Suchraumes verloren, da eine geeignetere Konstruktions- oder Adaptionsvorschrift für die Entwicklung einer Netzstruktur fehlt.

9.5 Test: TS

Für die Testreihen des TS-Algorithmus aus Abschnitt 6.5.3 wird auf unterschiedlichen Plattformen die Laufzeitumgebung JRE in Version 1.6 eingesetzt. Zusätzlich erfolgt die Abwandlung der Testaufgabe in Überwachtes Lernen. Damit und mit der Verwendung von zwei neuen Datensätzen in Vortests wird zunächst Rechenzeit eingespart. Als Gütekriterien kommen sowohl der RMSE der Residuen zwischen Ist- und (Referenz-) Sollausgabe des Netzwerkes als auch eine Kreuzvalidierung zum Einsatz. Bzgl. des Datensatzes des Benchmarks werden diese Größen auch für den Fitnesswert berechnet.
Die Tests beginnen mit Neuronalen Netzwerken ohne versteckte Schicht und werden sukzessiv vergrößert, wobei pro Test ein Datensatz durchlaufen, die Informationsmatrix des Algorithmus gefüllt und anschließend die optimale Linearkombination mittels CGNE-Algorithms berechnet wird. Aufgrund der Laufzeit des Algorithmus, die mit der Anzahl der Neuronen und der Größe des Datensatzes skaliert, werden dabei für die Analyse zunächst nicht die Daten des Benchmarks, sondern zwei neue Datensätze mit geringerer Komplexität, herangezogen.

9.5.1 Startwerte und Parameter

Der Algorithmus unterstützt die Parameter Propagierungsfunktion, Aktivierungsfunktion, Existenz eines Biasneurons, Existenz einer Konstante in der Linearkombination, Transformation der Eingabe und Tiefe des neuronalen Netzes bzw. Anzahl

Parameter	**Wahl**
Propagierungsfunktion	Skalarprodukt
Aktivierungsfunktion	AbsAZ, AbsBZ, SigmoidZ, SinCosZ, SinZZ, SinZ, TanhZZ
Skalierung der Aktivierungsfunktion	(variiert je nach Aktivierungsfunktion)
Biasneurons	ja, nein
Konstante in der Linearkombination	ja, nein
Transformation der Eingabe	ja, nein
Anzahl der verdeckten Neuronen	u.a. 0, 500, 1000, 2000, 5000, 10000
initialer CGNE-Vektor	$\overrightarrow{0}$
maximale CGNE-Iterationsanzahl	(vierfache Größe des verwendeten Datensatzes)
maximale CGNE-Fehlertoleranz	$1e-6$

Tabelle 25: Parameterwahl für den Test von TS

der verdeckten Neuronen. Ferner existieren die Metaparameter Startvektor, maximale Iterationsanzahl und maximale Fehlertoleranz des CGNE-Verfahrens sowie die Skalierung einer Aktivierungsfunktion.
Für die Testreihen wird die Wahl der Propagierungsfunktion auf das Skalarprodukt, die des CGNE-Startvektors auf den Nullvektor und die der maximalen CGNE-Iterationsanzahl auf die vierfache Problemdimension (Datensatzgröße) fixiert. Die letzten beiden Vorgaben entsprechen Standardwerten von CGNE. Die anderen Parameter werden dann für eine ausgewählte Aktivierungsfunktion aus Tabelle 2 (Seite 15) auf Basis verschiedener Netzwerkgrößen getestet. Tabelle 25 fasst die Parameterwahlen zusammen.

9.5.2 Datensätze

Der zuerst verwendete Datensatz wird anhand der zufällig ausgewählten Funktion $f : \mathbb{R}^4 \to \mathbb{R}^2$

$$(x_0, x_1, x_2, x_3) \mapsto \begin{pmatrix} x_0 \cdot x_1 \\ x_1 \cdot x_2 \cdot x_3 + 12 \end{pmatrix} \tag{15}$$

mit den ganzzahligen Argumentvektoren $(0:3, -1:2, 1:4, 3:4)$ erzeugt und enthält somit 128 Einträge aus Eingabe- und Referenzvektor. Der zweite Datensatz ist der persönlich in Tabelle 26 erstellte und enthält 21 willkürliche Datenpunkte. Der dritte Datensatz ist der des Benchmarks aus Abschnitt 7.2 und umfasst 1023 Einträge. Die Datensätze sind ferner auf dem Datenmedium (Seite 150) verfügbar.

9.5.3 Ergebnisse

Das Ziel des Tests ist ein Vergleich der Aktivierungsfunktionen, wobei nach einer Funktion gesucht wird, die primär eine möglichst große Informationsmenge im Netzwerk - und damit die beste Lösung der Lernaufgabe - sowie sekundär eine optimale Generalisierung des Netzes ergibt.

Eingabe		Referenz	Eingabe		Referenz
$(1, 0, -1, 10)$	$\mapsto$	$(0.1, 10)$	$(18, -8.81, 4.1, -7.78)$	$\mapsto$	$(3.14, 100)$
$(2, 1, -3, 20)$	$\mapsto$	$(0.2, -11)$	$(88, -0.2, 9.03, 4.803)$	$\mapsto$	$(-3, 991)$
$(3, 2, -6, 3)$	$\mapsto$	$(-20.3, 12)$	$(-48, 64.2, 3.003, 2340)$	$\mapsto$	$(-4.009, -124)$
$(4, 4, 9, 40)$	$\mapsto$	$(0.4, 23)$	$(238, 91, 5, -64.4)$	$\mapsto$	$(11.04, 54)$
$(-5e88, 8, -12, 50)$	$\mapsto$	$(0.5, 1e7)$	$(-998, 4, 5.78, -80)$	$\mapsto$	$(-100, -2.23)$
$(6, 16, -15, 60)$	$\mapsto$	$(-6, 54)$	$(28, 4, -5.9, 800)$	$\mapsto$	$(-8, -1)$
$(7, 3, 18, 70)$	$\mapsto$	$(40.7, 46.91)$	$(18, -64, 23, 42)$	$\mapsto$	$(-623, 2)$
$(8, 64, -21, 80)$	$\mapsto$	$(0.8, 74)$	$(-23, 42, 10, 1e9)$	$\mapsto$	$(-3000, 23)$
$(0.12, 24, -1, -12)$	$\mapsto$	$(-12391223, 749)$	$(3.491, -1.45, 3, 0)$	$\mapsto$	$(63, 13)$
$(-1, -100000, 0, -99)$	$\mapsto$	$(1, 4)$	$(277.924, -13.13, 78.74, 4)$	$\mapsto$	$(-0.23, 10.23)$
$(-10008, -1002, -7.7, 67.9)$	$\mapsto$	$(-9.9, -5)$			

Tabelle 26: Datenpaare des zweiten Datensatzes

Obwohl die zuletzt genannte Eigenschaft die erste bedingt und deren Verallgemeinerung darstellt, werden beide betrachtet, weil zumindest für Aktivierungsfunktionen mit unendlicher VC-Dimension[106] eine Lösung mit der ersten Eigenschaft vermutet, aber allgemein eine mit guter Generalisierung als unwahrscheinlich angesehen wird.

Ferner wird die Transformation der Eingabe separat betrachtet, weil ihre Analyse als Boolscher Parameter den hohen Aufwand ansonsten noch verdoppeln würde und eine Relevanz aufgrund von möglichen Skaleneffekten kaum erwartet wird.

XOR

Zu Beginn wird kurz der Vollständigkeit halber das klassische XOR-Problem, siehe Fußnote 8 (Seite 5), betrachtet. Dieses kann von allen Aktiverungsfunktionen fehlerfrei gelöst werden. Tabelle 104 zeigt dafür in Abhängigkeit der Boolschen TS-Parameter die minimalen Größen der verdeckten Schicht, die für das Finden einer Lösung notwendig sind. Gut zu erkennen ist dabei das uninformative Biasneuron, das lediglich die 1 propagiert, aber nicht zu einer größeren Informationsmenge beiträgt.

(un-) transformierte Eingabedaten	ohne Konstante	mit Konstante
ohne Bias	3 (3)	3 (3)
mit Bias	6 (6)	6 (6)

Tabelle 27: Minimale Zahl verdeckter Neuronen zum Erlernen von XOR für TS

[106] Die VC-Dimension einer Funktionsklasse ist definiert als die Kardinalität der größten Punktmenge, die von Funktionen dieser Klasse separiert (klassifiziert) werden kann. Die Klasse der Polynomfunktionen besitzt beispielsweise eine unendliche VCD. Siehe dazu ergänzend auch die Übersicht in Christey (2000, Abschnitt 2.6.1).

Funktions-	**Bias-**	**Datensatz**			**Trainingsmenge**			**Testmenge**		
name	**neuron**	RMSE	Neuronen	Z	RMSE	Neuronen	Z	RMSE	Neuronen	Z
AbsAZ	ohne	0.0574	2000	10	0.0604	1000	10	0.3141	500	4
	mit	0.0608	2000	10	0.064	2000	10	0.3636	2000	4
AbsBZ	ohne	0.0043	1000	15	0.0051	500	15	0.0294	10000	15
	mit	0.0051	2000	15	0.0061	5000	15	0.0296	2000	15
SigmoidZ	ohne	0.0293	10000	0.25	0.0288	10000	0.25	0.041	10000	0.25
	mit	0.0283	5000	0.25	0.0282	10000	0.25	0.0391	10000	0.25
SinCosZ	ohne	<1e-4	1000	7	<1e-4	1000	7	3.5539	500	15
	mit	<1e-4	1000	7	<1e-4	1000	7	2.1192	500	15
SinZZ	ohne	0.0030	5000	7	0.0032	5000	7	0.0111	1000	15
	mit	0.0002	2000	400	0.0004	2000	400	**0.0007**	2000	400
SinZ	ohne	**<1e-4**	500	15	**<1e-4**	500	15	1.9222	500	15
	mit	<1e-4	1000	15	<1e-4	1000	15	0.7016	500	15
TanhZZ	ohne	0.0051	5000	50	0.0066	5000	50	0.0176	1000	50
	mit	0.0006	500	400	0.0006	500	400	**0.0009**	500	400
TanhZ	ohne	0.9744	500	1	0.9857	500	1	4.0889	0	1
	mit	0.9896	1000	1	0.9883	1000	1	4.0972	0	1

Tabelle 28: Ergebnisse für Daten und Kreuzvalidierung des ersten Datensatzes bei transformierter Eingabe

Aktivierungsfunktion, Skalierung, Bias und Konstante

Die Aktivierungsfunktionen aus Tabelle 2 werden anhand des ersten Datensatzes mit transformierten Eingabedaten getestet. Dazu wird mit der jeweils ausgewählten Aktivierungsfunktion komponentenweise das Bild der Eingabe berechnet und an das Netzwerk als Eingabe weitergereicht.

Die besten Ergebnisse der Parametervariationen von Wahl der Aktivierungsfunktion, Skalierung der Aktivierungsfunktion, Existenz des Biasneurons und Anzahl der verdeckten Neuronen sind in Tabelle 28 dargestellt, wobei die selektierten Einträge nach größtem Fitnesswert, kleinster Netzgröße und kleinster Skalierung ausgewählt sind. Die vollständigen Testergebnisse befinden sich im Anhang (Seite 137).

Neben dem RMSE für den gesamten Datensatz wird eine Kreuzvalidierung mit einem Verhältnis von $1 : 9$ zwischen Test- und Trainingsmenge ($k = 10$) präsentiert. Für die verschiedenen Aktivierungsfunktionen und Skalierungen zeigen dabei fast immer Netzwerke mit Biasneuron bessere Resultate als solche ohne.

Dies ist jedoch nicht auf die Existenz eines Biasneurons im Netzwerk zurückzuführen, sondern auf die dadurch resultierende Konstante in der Linearkombination. Wird der Bias aus einem Netzwerk entfernt, aber die Konstante für die Berechnung der Linearkombination beibehalten, folgen bessere Ergebnisse. Der Grund hierfür ist die dann - bei fixierter Neuronenanzahl - steigende Verschachtelungstiefe des Netzes. Wie auch bei einer linearen Regression nimmt die Konstante dabei den Teil der Daten auf, der nicht durch die Regressoren (Neuronen) erklärt wird. Die Kon-

Funktions-	Bias-	Datensatz			Trainingsmenge			Testmenge		
name	**neuron**	RMSE	Neuronen	Z	RMSE	Neuronen	Z	RMSE	Neuronen	Z
transformiert										
SinZZ	mit	0.0008	500	400	0.0008	500	400	**0.001**	500	400
TanhZZ	mit	0.0006	500	400	0.0006	500	400	0.0009	500	400
untransformiert										
SinZZ	mit	**<1e-4**	500	0.1	**<1e-4**	500	0.1	0.0012	500	400
TanhZZ	mit	0.0006	500	400	0.0006	500	400	0.0009	500	400

Tabelle 29: Vergleich der Ergebnisse für Daten und Kreuzvalidierung des ersten Datensatzes bei transformierter und untransformierter Eingabe

stante wird aus diesen Gründen als wichtig erachtet und weiterhin benutzt. Für eine Vergleichbarkeit mit Tabelle 28 wird zudem, sofern nicht anders angegeben, das Biasneuron trotz Entbehrlichkeit beibehalten.

Ferner erzielen die Funktionen AbsBZ, SinCosZ, SinZZ, SinZ[107] und TanhZZ sehr kleine RMS-Fehler auf dem ersten Datensatz. Insbesondere erreichen SinZZ und TanhZZ sogar eine optimale Generalisierung. Das enge Zusammenliegen von SinZZ und TanhZZ beruht unterdessen auf dem fast identischen, nahezu linearen und sehr flachen Funktionsverlauf um den Nullpunkt herum, vergleiche auch Abbildung 2 (Seite 16). In Abbildung 28 genügt daher zur Illustration der mit zunehmender Neuronenzahl gegen Null konvergierenden Lernkurve die Darstellung der Funktion TanhZZ (mit einer Skalierung von 400 und Biasneuron im Netzwerk).

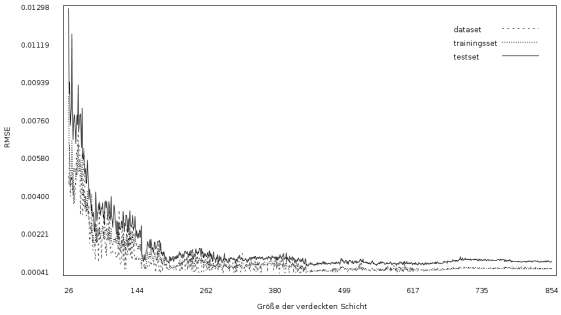


Abbildung 28: Lernkurven zur Funktion TanhZZ für den ersten Datensatz.

In die engere Wahl der besten Aktivierungsfunktionen für die nachfolgenden Testreihen wird zusätzlich zu den Funktionen SinZZ und TanhZZ, welche die beste Generalisierung einer Lösung bieten, auch die Funktion SinZ aufgenommen, die den ersten Test bzgl. des unverallgemeinerten Gütekriteriums gewinnt. Einen ausgezeichneten Gewinner gibt es vorerst nicht. In der Tendenz sind aber möglichst große Skalierungen einer Aktivierungsfunktion vorzuziehen.

Transformation der Eingabe

Tabelle 29 zeigt unter den Voraussetzungen des ersten Tests (c.p.) die Ergebnisse der besten Aktivierungsfunktionen für eine transformierte und eine untransformier-

[107] Für die Aktivierungsfunktion SinZ wird eine hohe Güte aufgrund der unendlichen VC-Dimension der sie umfassenden Funktionenklasse $f(\alpha, z;) = \sin(\alpha \cdot z)$ vermutet. Ferner wird erwartet, dass sich diese Eigenschaft auf neuronale Netze überträgt oder dort sogar bereits vorhanden ist.

Funktions-	**Bias-**	**Datensatz**			**Trainingsmenge**			**Testmenge**		
name	**neuron**	RMSE	Neuronen	Z	RMSE	Neuronen	Z	RMSE	Neuronen	Z
SinZ	ohne	**<1e-4**	500	1	**<1e-4**	500	1	**1146388**	10000	400
	mit	**<1e-4**	500	1	**<1e-4**	500	1	1192333	10000	50
TanhZZ	ohne	**<1e-4**	500	7	**<1e-4**	500	7	2001222	0	50
	mit	**<1e-4**	500	7	**<1e-4**	500	7	**1825005**	5000	1

Tabelle 30: Ergebnisse für Daten und Kreuzvalidierung des zweiten Datensatzes bei transformierter Eingabe

te Eingabe für einen kleinen Test mit 10, 100 und 500 verdeckten Neuronen sowie 11 unterschiedlichen Skalierungen. Vergleiche dazu auch Tabelle 10.4 im Anhang (Seite 145). Mit Blick auf die Generalisierung scheint die Transformation des Eingabevektors im Fall der Funktion SinZZ einen leichten Vorteil mit sich zu bringen, bei der TanhZZ-Funktion liegen die Unterschiede jedoch jenseits der 5. Nachkommastelle.

Während also für den ersten Datensatz kaum ein Unterschied zu sehen ist, führt die untransformierte Eingabe hingegen beim zweiten Datensatz aufgrund einer zu schlechten Kondition der Informationsmatrix zum numerischen Versagen des CGNE-Algorithmus, so dass die resultierenden Fehler oberhalb des sechsstelligen Bereiches liegen.

Die Transformation der Eingabe wird deshalb beibehalten. Ferner wird vermutet, dass das mäßige Abschneiden einiger (der auch für die Transformation der Eingabe verwendeten) Aktivierungsfunktionen im ersten Test auf ihre Unbeschränktheit zurückgeführt werden kann. Denn wie in Abschnitt 2.4 beschrieben, ist besonders ein stabiles bzw. robustes Verhalten der Aktivierungsfunktion gegenüber unbekannten Argumenten wünschenswert, um z.B. (numerisch) explodierende bzw. degenerierende Zustände in einem Netzwerk zu hindern.

Robustheit

Die Ergebnisse der beiden besten Aktivierungsfunktionen auf dem zweiten Datensatz sind in Tabelle 30 dargestellt. Obwohl beide Funktionen eine fehlerfreie Lösung der Aufgabe bzgl. des Gütekriteriums (RMSE) ermöglichen, misslingt die Generalisierung. Der TS-Algorithmus ist demnach nicht in der Lage, eine genügend reichhaltige Informationsmenge zu erzeugen, wie es der Test mit dem ersten Datensatz zunächst erwarten lässt. Damit ist der Algorithmus bzgl. der Generalisierung seiner Lösung nicht robust gegenüber einer gestellten Aufgabe.

Benchmark

Für den Test des TS-Algorithmus mit dem dritten Datensatz werden die ausgewählten Aktivierungsfunktionen für die Skalierungen 1, 7, 15, 50, 150, 400 in Netzwerken mit transformierter Eingabe und Bias bei 2500 und 5000 verdeckten Neuronen sowie Konstante in der Linearkombination getestet. Tabelle 31 zeigt zusammenfassend die besten und schlechtesten Resultate. Im Kontrast zu den bisherigen Testergebnissen fallen für die Funktionen SinZZ und TanhZZ dabei die mit steigender Skalie-

Funktions-	**Skal.**	**Datensatz**	**Trainingsm.**	**Testmenge**	**Datensatz**	**Trainingsm.**	**Testmenge**
name	z	RMSE (bei jeweils 5000 Neuronen)			Fitnesswert (bei jeweils 5000 Neuronen)		
SinZ	1	3.870	3.7867	**15.0048**	0.2527	0.2464	0.8186
SinZ	400	**<1e-4**	**<1e-4**	96.4665	**<1e-4**	**<1e-4**	4.9897
SinZZ	1	3.8701	3.7867	**15.0049**	0.2527	0.2464	0.8186
SinZZ	400	5.4418	5.4194	18.3075	0.3408	0.3404	**0.3778**
TanhZZ	1	3.5091	3.4006	16.0412	0.2231	0.2153	0.9972
TanhZZ	400	5.4242	5.4048	18.1766	0.3402	0.3401	**0.381**

Tabelle 31: Ergebnisse für Daten und Kreuzvalidierung des dritten Datensatzes bei transformierter Eingabe

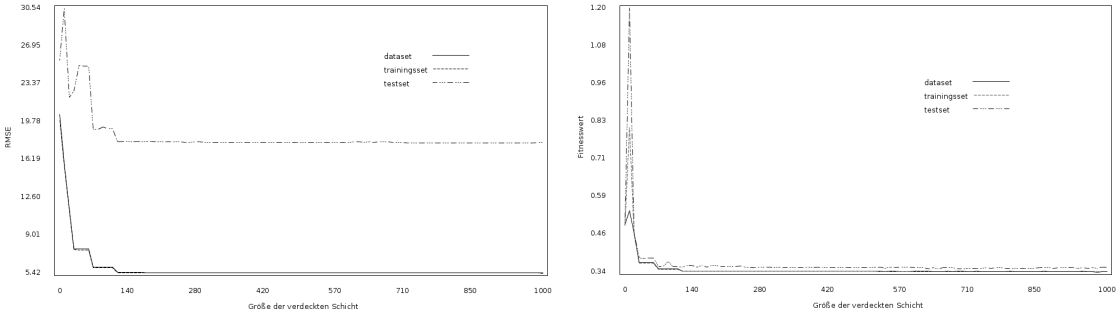


Abbildung 29: Lernkurven zur Funktion TanhZZ für den dritten Datensatz

rung schlechter werdenden Fehlerwerte (RMSE) auf, während die Fitnesswerte der Kreuzvalidierung gleichzeitig besser werden. Mangels einer alternativen Interpretation liegt daher die Annahme nahe, dass die erzeugten Netzwerke aufgrund zu geringer Größe an der Komplexität des Benchmarks scheitern oder gänzlich ungeeignet sind, um für eine Lösung eine brauchbare Informationsmenge zu enthalten. Letzteres wird zudem durch den Verlauf der Lernkurven für die Funktion TanhZZ mit einer Skalierung von 400 in Abbildung 29 gestützt, weil ab einer kleinen Anzahl verdeckter Neuronen eine fortwährende Stagnation eintritt. Vergleiche dazu auch die Werte für 2500 und 5000 Neuronen in Tabelle 31 oder Tabelle 66 im Anhang (Seite 148). Die Informationsmenge in den Netzen auf Grundlage von SinZZ und TanhZZ könnte allerdings gesteigert werden, indem Kombinationen von verschiedenen Neuronen eingesetzt werden. Derartige Versuche stehen aber noch aus.

Ungeachtet dessen ermöglicht die Aktivierungsfunktion SinZ auch in diesem Test wieder eine fehlerfreie, wenn auch abermals nicht generalisierende, Lösung. Die Lernkurven zu RMSE und Fitnesswert für diese Aktivierungsfunktion werden in Abbildung 30 dargestellt[108]. Die Grafiken zeigen vornehmlich die minimale Anzahl an verdeckten Neuronen, ab welcher der Datensatz fehlerfrei (auswendig) gelernt wird. Im Gegensatz zu den ersten beiden Datensätzen erfordert der dritte dazu

[108]Die Lernkurven der Kreuzvalidierung werden vernachlässigt, weil keine Generalisierung vorliegt. Die Daten dazu befinden sich trotzdem auf dem Datenmedium im Anhang (Seite 150).

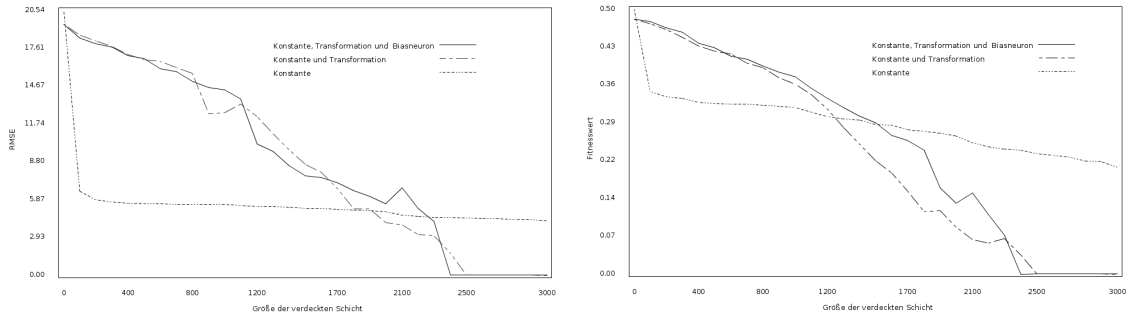


Abbildung 30: Lernkurven zur Funktion SinZ für den dritten Datensatz

merklich mehr Neuronen.

9.5.4 Zusammenfassung

In Bezug auf die numerischen Probleme des Verfahrens verweisen insbesondere schon die Abschnitte 2.2.1, 2.4 und 5.2.5 verschieden auf das Problem mit dem Umgang unbeschränkter Eingaben (entweder global für ein Netzwerk oder lokal für ein Neuron). Dabei fällt die häufig anzutreffende hohe Skalierung der Aktivierungsfunktionen bei den besten Ergebnissen interessanterweise mit einer ausgeprägten Linearität der Funktion im Nullpunkt zusammen. Darin kann der Versuch gesehen werden, den unendlichen Wertebereich (die Unendlichkeit) einer unbeschränkten Eingabe optimal auf ein beschränktes Intervall zu transformieren - wobei die (lineare) Identität id keiner Transformation entspricht, was in Bezug auf die Existienz von Skaleneffekten optimal ist. Für den Fall einer beschränkten Aktivierungsfunktion könnte diesbezüglich mit Hilfe einer Abschätzung der netzinternen (minimalen und maximalen) Zustände dynamisch zur Eingabe versucht werde, diese Transformation optimal zu gestalten.
Bezüglich der Laufzeiten von NEAT und EANT für die wiederholte Auswertung eines Netzwerkes zeigt sich zudem die Wichtigkeit einer effizienten Implementierung der zugrunde liegenden Datenstruktur. EANT setzt diese für die Auswertung nicht aber für die Repräsentation eines NN um, während NEAT in beiden Punkten ineffizienz ist. Eine Auswirkung der Programmiersprachenwahl ist dabei natürlich enthalten. Ihr wird bei den verwendeten Operatoren jedoch keine signifikante Bedeutung zu Teil.
Ferner zeigen die sehr verschiedenen Ergebnisse kein klares Bild des TS-Algorithmus. Während für den ersten Datensatz eine optimale und sogar generalisierende Lösung gefunden wird, versagt der Algorithmus bei der Suche nach einer verallgemeinernden Lösung für die anderen Datzensätze und nahezu jede Wahl einer Aktivierungsfunktion. Dennoch ist er fast immer in der Lage, eine spezielle Lösung auf Basis der SinZ-Funktion zu finden. Eine theoretische Betrachtung der Wahl einer Aktivierungsfunktion sollte den Versuchen daher genauso folgen, wie die Analyse des Verhaltens einzelner Neuronen und die Suche nach Mustern bei der

Informationsgewinnung.
Nicht vollständig geklärt ist auch die Wirkung des CGNE-Verfahrens auf die Resultate von TS. Ein kurzer Vergleich mit anderen Lösungsverfahren für lineare Gleichungssysteme (SVD sowie Minimierung eines quadratischen Fehlerfunktionals) erbrachte jedoch gleiche Ergebnisse.
Insgesamt zeigt sich damit, dass der TS-Algorithmus auf Basis struktureller Informationsgewinnung und -verarbeitung grundsätzlich funktioniert. Da sein Test zudem vorerst mehr Fragen als Unmöglichkeiten aufzeigt, ist seine Vorgehensweise vermutlich ausbaufähig.

9.6 Diskussion

Die untersuchten Algorithmen NEAT, EANT und TS (sowie am Rande der evolutionäre Baukasten) stellen die wesentlichen Techniken zur Entwicklung einer neuronalen Topologie dar.
Den Experimenten mit EANT und dem Baukasten nach zu urteilen, liegt der Erfolg dabei im Hinzufügen von Struktur (Teilnetzwerken), die sonst scheinbar nicht oder nur äußerst unwahrscheinlich bzw. selten entsteht. Dies ist unbefriedigend, da keine neuartige Technik, sondern nur eine leichte Abwandelung eines alten Vorgehens zur Struktursuche den Unterschied zu älteren Verfahren auszumachen scheint, und die wichtige Frage, wie sich aus einer Informationsmenge eine neuronale Struktur (-mutation) ableiten lässt, weiterhin offen bleibt.
Genauer betrachtet, besteht das Problem in der Latenz eines direkt formulierbaren Zieles bei der Strukturentwicklung. Zwar ist eine Rückmeldung der Umwelt an das Netzwerk in Form einer Referenzausgabe (Überwachtes Lernen) oder eines Skalars (Verstärkendes Lernen) vorhanden, doch ist kein Weg (außer der stochastischen Suche) bekannt, um daraus eine (zielgerichtete) strukturelle Abpassung abzuleiten. Ebenso wenig gelingt es, ein Gesellschaftsspiel mit dieser Problematik als Gegenstand zu formulieren, um spielerisch kreative Lösungen oder Ansätze zu suchen bzw. suchen zu lassen[109].
Immerhin zeigt sich beim Vergleich von EANT und NEAT der zu erwartende Vorteil einer Trennung von Struktur- und Gewichtsoptimierung gegenüber einer nicht getrennten Optimierung und bestätigt die bekanntermaßen effizienzsteigernde Separierung eines Problems in weniger komplexe Teilprobleme. Sichere Schlussfolgerungen aus den Experimenten mit den stochastischen Algorithmen werden jedoch aufgrund der mitunter sehr wenigen Versuche in ihrer Aussagekraft beschränkt.
In der Tendenz zeigt sich allerdings sowohl bei NEAT und EANT als auch den Baukastenexperimenten ab einer bestimmten Netzwerkgröße eine beginnende Stagnation der Ergebnisse mit simultaner Abnahme der Streuung. Aufgrund der Vielzahl verschieden konstruierter Netze innerhalb einer Generation dürfte dies aber eigentlich nicht passieren; insbesondere nicht, wenn der diskrete Raum der Netzstruktur

[109] Auf der Suche nach einem möglichen Ansatz für eine Problemlösung ist dies ein ungewöhnlicher aber mitunter nicht untypischer Weg.

ausbalanciert verwendet wird.
Als Grund für diese Degeneration wird neben möglichen Eigenheiten eines Optimierungsverfahren maßgeblich die chaotische Natur eines Neuronalen Netzwerkes angeführt. Die Eigenschaft einer chaotischen Funktion, dass kleine Variationen in der Eingabe große und sprunghafte Variationen in der Ausgabe ergeben, bedingt dabei im Zuge einer Funktionsoptimierung immer kleinere Schrittweiten im Suchraum (Urbild der Funktion), die letztendlich zu Stagnation, numerischen Problemen und den beobachteten Folgen führen.
Ein praktikabler Ausweg aus dieser Problematik scheint in einer extensiven Strukturentwicklung zu liegen. Größere Netzstrukturen sind zwar schwieriger zu optimieren, weisen aber mehr Plastizität auf, von der ein ausreichender Teil trotz numerischer Probleme durch eine Gewichtsoptimierung zugänglich zu sein scheint. Dies überrascht etwas, weil die evolutionäre Suche auch hinsichtlich einer Vermeidung von großen Netzstrukturen, vergleiche Einleitung zu Kapitel 6, eingesetzt wird.
Zur Umsetzung einer verstärkten Strukturentwicklung könnte das Aufwandsverhältnis zwischen Gewichtsoptimierung und Strukturentwicklung dynamisch in Abhängigkeit vom Zustand des Evolutionsprozesses angepasst werden. Für eine solche Adaption wäre jedoch zunächst ein geeignetes (Ähnlichkeits-) Maß für Netzwerktopolgien zu entwerfen, das eine Entscheidung über die Art der Anpassung ermöglicht.
Neben der gewichtsbasierten evolutionären Konstruktion eines Neuronalen Netzwerkes (z.B. in Form einer evolutionären Entwicklung) steht außerdem die alleinige Strukturentwicklung zur Verfügung, wie sie der TS-Algorithmus versucht. Dabei kann TS zwar oft keine akzeptable Generalisierung einer Lösung gewährleisten, die Konstruktion eines angelernten funktionalen Informationsspeichers (bzw. die mehrdimensionale Funktionsapproximation) scheint dagegen gut zu gelingen. Indes sind die Möglichkeiten einer rein strukturellen Entwicklung von neuronalen Topologien (ohne Gewichtung) bislang eher unbekannt.

10 Schlussbetrachtung

Im Folgenden werden Gedankengänge formuliert, die immer wieder während der Arbeit mit (künstlichen) Neuronalen Netzwerken aufkommen.
Die erste Betrachtung befasst sich mit dem Lernvorgang in neuronalen Netzen, und betrifft sowohl die unscharfe Definition des Auswendiglernens, die vielfach eine klare Sicht auf eine Vorgehensweise erschwert sowie mit passenden und unpassenden Interpretationen überladen ist, als auch die Gestaltung des dahinterliegenden Lernprozesses.
Der zweite Gedanke fragt nach Sinn und Zweck der (neuronalen) Modellsuche und betrachtet Simulation als konstruktives Mittel, um Dinge besser verstehen zu können. Dabei verfolgt dieser in groben Zügen den Weg möglicher Erkenntnis und versucht den Ansatz der Modellsuche soweit wie möglich zu rechtfertigen.
Der dritte Gedankengang fasst die erkannten Probleme und Grenzen der vorgestellten Arbeit mit künstlichen Neuronalen Netzwerke zu einer Kritik zusammen und stellt die Erfolgsfrage für diese Konzepte und die untersuchten Algorithmen.
Abschließend erfolgt der Ausblick auf weiterführende und noch nicht abgeschlossene Fragestellungen.

10.1 Neuronales Lernen

Das Phänomen des Auswendiglernens von Daten in einem Neuronalen Netzwerk, dargelegt in Abschnitt 4.6.1, widerstrebt dem gewünschten Interpolationsvermögen eines neuronalen Netzes, denn statt einer allgemeinen Aufgabenlösung wird eine spezialisierte erzeugt. Jedoch sind beide Richtungen legitim. Die allgemeine beispielsweise um eine unbekannte Funktion zu interpolieren, die spezielle zum Beispiel um eine vollständig bekannte Relation (z.B. XOR) zu erlernen. Es ist also sinnvoll vorab eine Entscheidung über das Lernziel zu treffen. Tabelle 32 zeigt das favorisierte Vorgehen bei der Selektion eines Lernalgorithmus.
Dieses Lernziel sollte zudem Wahl und Zielsetzung des Lernalgorithmus beeinflussen, denn ein Blick auf das Beispiel der Gewichtsoptimierung offenbart, dass diese aufgrund der verbreiteten Zielsetzung nur ein approximatives Verfahren zur Suche nach generalisierenden Netzwerken ist. Bei der Wahl eines Gütekriteriums wird nämlich gängigerweise nur das Kriterium, nicht aber dessen Generalisierung, als Lernziel verwendet[110]. Diese wird dann als schwache Nebenbedingung parallel zur Güte mitberechnet und ihr fortschreitender Plot nach jedem Lernschritt betrachtet. Bekanntermaßen weist dieser sehr bald ein Minimum auf, an dem der Lernvorgang üblicherweise abgebrochen und das entsprechende Netzwerk als generalisierende Lösung verwendet wird. Die gefundene Lösung ist somit ein Kompromiss zwischen Gütekriterium und dessen Verallgemeinerung. Weiter offenbart der externe Abbruch des Lernvorgangs die Unvollkommenheit dieses Vorgehens. Würde

[110] Dies gilt insbesondere auch für die Verfahren Backprop, Blackbox, NEAT und EANT. Hingegen ist TS bis auf die Selektion eines Neuronalen Netzwerkes aus der Menge aller erzeugten Netze von der Wahl eines Gütekriteriums unabhängig.

1. Schritt:	Wahl der Lernaufgabe
2. Schritt:	*Wahl des Lernziels*
3. Schritt:	Wahl des Lernparadigmas
4. Schritt:	Wahl des Gütekriteriums
5. Schritt:	Wahl des Algorithmus
6. Schritt:	Algorithmische Informationsgewinnung
7. Schritt:	Algorithmische Informationsverarbeitung
8. Schritt:	Validierung des Erlernten

Tabelle 32: Lernvorgang mit Lernziel

hingegen die Generalisierung direkt als Lernziel eingesetzt, gäbe es diesen Kompromiss nicht. Demgegenüber zielt die komplette Vernachlässigung der Generalisierung auf die Konstruktion eines funktionalen Informationsspeichers ab, wie er auch in Abschnitt 4.6.1 genannt wird. Die präzise Ausgestaltung des Lernzieles ist daher, genau wie bei jeder Optimierung, wesentlich.
Ferner ist im Rahmen der reinen Strukturentwicklung bemerkenswert, dass einerseits mit Hilfe der Sinusfunktion ebenfalls ein funktionaler Informationsspeicher scheinbar problemunabhängig konstruiert und andererseits bei Verwendung des einheitlich skalierten Tangenshyperbolikus oder Sinus zumindest manchmal eine gute Generalisierung erreicht werden kann. Die einzigen verfahrensspezifischen Probleme stellen dabei die Wahl von Aktivierungsfunktion und Skalierung sowie die Netzwerkgröße dar. Allgemein ist außerdem auch ungeklärt, ob eine konstante Gewichtung die Plastizität von Neuronalen Netzwerken einschränkt.

10.2 Abstraktion

Der in Abschnitt 1.1 vorgestellte abstrakte Informationsbegriff ist ein Annäherungsversuch an einen allgemein praktischen Umgang mit Information, weil dieser (in möglichst kontextfreier Umgebung) ein generelles Problem der Erkenntnistheorie ist. Eine alte Frage, die auch mit Platos Höhlengleichnis in Verbindungen gebracht werden kann, lautet: Ist der Mensch in der Lage, das Universum zu erklären, in dem er sich befindet? Wird der Mensch als Element und das Universum als System abstrahiert, so beinhaltet die nun verallgemeinerte Frage für ein gegebenes System folgende Teilfragen.

1. Existiert ein Modell des Systems?
2. Kann das Modell aus Elementen des Systems bestehen?
3. Kann das Modell innerhalb des Systems entwickelt werden?
4. Kann das Modell innerhalb des Systems benutzt werden?

Die erste Teilfrage ist eine Existenzfrage und läßt sich bejahen, weil eine mögliche Identifikation von Modell und System die Existenz eines Modells des Systems ergibt.
Die anderen Teilfragen sind hingegen Entscheidungsprobleme, womit die Unvollständigkeitssätze des Logikers Kurt Gödel konsultiert werden können. Diese zeigen, dass das Modell eines Systems mit den Mitteln des Systems nicht beweisbar ist. Ein solches Modell ist damit jedoch nicht widerlegt, denn es bedarf prinzipiell keines Korrektheitsbeweises, um es zu entwickeln und zu benutzen. Doch erst das Verlassen des Systems ermöglicht die Beweisbarkeit dieses Modells. Werden insbesondere Modell und System miteinander identifiziert, so illustrieren beispielweise die Fragen „Schlafe ich?“ oder „Lebe ich?“ ein Problem, welches z.B. für das autarke System eines einzelnen Mensch nicht immer entscheidbar ist. Siehe dazu auch das Entscheidbarkeits- und Halteproblem in Turing (1936) und Church (1936). Erst ein äußeres (beobachtendes) System erbringt die Entscheidbarkeit. Wird diese Argumentationskette für korrekt gehalten, können die Antworten auf die folgenden Teilfragen nur Hypothesen sein.
Teilfrage 2 betrifft die Darstellbarkeit eines Modells innerhalb des Systems. Eine Antwort kann mit Hilfe weiterer Abstraktionen versucht werden, denn ein Modell braucht eine beschreibende Sprache, und eine Sprache basiert auf einer Grammatik (siehe Chomskys Universalgrammatik in Chomsky (1956) und Chomsky-Hierarchie in Chomsky u. Schützenberger (1963); Chomsky (1965) sowie Chomsky (1965)) und diese wiederum auf einem Alphabet. Letzteres kann durch Kombinationen von Symbolen kodiert werden, wobei vermutlich zwei Symbole (z.B. die Bits $\{0, 1\}$) die minimale Menge notwendiger Information darstellen. Dies trifft z.B. auf binäre Rechnersysteme zu, die Modelle ihrer selbst und darauf aufbauende komplexere Systeme enthalten und simulieren können. Damit wird die Modellierung eines Systems ermöglicht, sobald in ihm zwei verschiedene Zustände darstellbar sind.
Die 3. Teilfrage bezieht sich auf die Fähigkeit des Umgangs mit Information und daran anknüpfend mit Abstraktion eines potenziellen Konstrukteurs im System. Im Universum scheint dies derzeit dem Menschen vorbehalten zu sein. Allerdings können Algorithmen zur Entwicklung Neuronaler Netzwerke als ein erster künstlicher Versuch einer Suche nach einem Modell auf Basis von Information interpretiert werden. Da Darstellung und Konstruktion eines Modells jedoch nicht eindeutig sind, dürfte fast sicher auch der berühmte Affe an der Schreibmaschine nach Borel (1913) und Christey (2000) irgendwann das korrekte Modell des Universums ertippen.
Die letzte Teilfrage fragt nicht nach (Turing-) Berechenbarkeit[111] und Church-These[112], sondern nach der Simulierbarkeit eines Modells. Diese ist immer möglich, denn eine Simulation entspricht der Kapselung eines Modells in der Instanz eines beobachtenden Systems, womit das Halteproblem umgangen und die Berechenbarkeit garantiert wird. Ihr steht lediglich ein zeitliches Problem entgegen, da einfache Systeme aus Kausalitätsgründen nicht in der Lage sein können, komplexere Modelle

[111] Für jede beliebige Eingabe kann das Modell (die Funktion) eine Ausgabe berechnen.
[112] „Die Klasse der Turing-berechenbaren Funktionen ist genau die Klasse der intuitiv berechenbaren Funktionen.“

schneller als sich selbst zu berechnen.
Insgesamt fehlt neben einem Beweis obiger Entscheidungsprobleme auch die äußere Sicht auf unser Universum, um alle Fragen beantworten zu können. Mit Sicherheit verbleibt damit nur die begründete Hoffnung, dass (sogar rechnergestützt z.B. mit der autonomen Entwicklung neuronaler Netzwerke) ein Modell des uns umgebenden Systems entdeckbar ist.

10.3 Zusammenfassung

Wird ein Neuronales Netzwerk als Funktion (mit veränderlichem abstraktem Syntaxmehrfachbaum) betrachtet, lassen nicht nur die numerischen Konditionierungsprobleme bei der Gewichtsoptimierung, sondern auch die offene Suche nach allgemeinen Zielen der Strukturentwicklung oder der Wahl von Aktivierungsfunktionen, den Eindruck aufkommen, dass zu wenig Grundlagen (Axiome) für die erfolgreiche Entwicklung einer neuronalen Topologie existieren. Dabei scheint die wesentliche Frage die nach der besten (biologischen oder künstlichen) Informationskodierung und deren Eigenschaften zu sein und sich die Suche nach Lernalgorithmen auf Grundlage dieser Informationsmenge einfacher zu gestalten, sobald diese Informationsmenge bekannt wird.
Der biologische Rahmen des neuronalen Konzeptes umfasst jedoch weit mehr Komponenten als bisher modelliert werden. Daher dürfen die Fragen nach einer zielgerichteten Verwendung von hemmenden und nicht-hemmenden Leitungen oder nach Möglichkeiten für selbstständig arbeitende BOT-Neuronen nicht (weiterhin) außer Acht gelassen werden. Gerade in Bezug auf das Zusammenwirken mehrerer Neuronentypen und BOT-Komponenten in einem neuronalen Netz, die eine hochgradig parallele (asynchrone) Informationsverarbeitung ermöglichen, werden wichtige Impulse für einen besseren Umgang mit neuronalen Funktionen erwartet. In diesem Zusammenhang wird besonders in SOM- und RBF-Netzwerken aufgrund des BOT-nahen Vorgehens ihrer Lernalgorithmen ein großes Potenzial vermutet.
Mit dieser Einschätzung und zusammen mit der großen Plastizität bzw. Approximationsfähigkeit der derzeit modellierten Netzwerke (der zweiten Generation) wird der neuronale Ansatz insgesamt als erfolgreich eingestuft. Dennoch basiert das zugrunde liegende Konzept bislang nur auf einer biologischen Beobachtung mit zahlreichen relativ unpräzisen Interpretationen ohne erkennbare Zielsetzung. Bei einer Betrachtung von außen gleicht ein Neuronales Netzwerk deshalb mehr der Datenstruktur einfacher (mehrdimensionaler funktionaler) Informationsspeicher, als einem Medium, das zu einer Generalisierung von Erlerntem fähig ist.
Bezogen auf CMA-ES zeigt sich das Bild eines Optimierungsalgorithmus, der sehr robust gegenüber den betrachteten (neuronalen) Funktionen ist, aber vermutlich an den Auswirkungen der chaotischen Eigenschaften dieser Netze scheitert. Bei Einbeziehung der neuronalen Netzstruktur in eine Vorgehensweise hingegen wird erwartet, dass ein Algorithmus konstruierbar ist, der die aufgetretenen Probleme bei der Optimierung nicht aufweist.

Bezogen auf NEAT ergibt sich das Bild eines evolutionären Algorithmus, der theoretisch eine umfassende neuronale Modellsuche ohne die Probleme einer numerischen Gewichtsoptimierung realisiert. Dies wird jedoch realisiert, indem bis auf die Verwaltung der Evolution (inkl. Speziation) und die einfache stochastische Suche im neuronalen Parameterraum weder Informationsgewinnung noch -verarbeitung stattfinden. Dem somit praktisch blinden Algorithmus widerfahren deshalb während der Suche weder große Probleme noch gute Lösungen, weil er sie schlicht nicht findet.
Für EANT ergibt sich hingegen das Bild eines Algorithmus, der aufgrund einer guten Gewichtsoptimierung häufig an die Grenzen der funktionalen Optimierung einer chaotischen Funktion stößt. Seine nicht ausbalancierte, (mangels eines bekannten Ziels) ziellose und simple stochastische Suche nach Struktur entspricht allerdings stark der von NEAT. Sie ist dieser aber leicht überlegen, weil anstelle von kleinsten Komponenten (Verbindungen, Neuronen) komplette Teilnetzwerke eingefügt oder entfernt werden. Dennoch wird sie als der Grund für das Nichterreichen einer Lösung ausgemacht.
Schließlich ergibt TS das Bild eines Algorithmus, der eine verfügbare Informationsmenge mit Hilfe der Komponenten eines Neuronalen Netzwerkes vergrößert und dann zu einer Lösung linear kombiniert. Zwar kann die Güte der Lösungen nicht immer überzeugen, trotzdem weist TS gegenüber NEAT und EANT erstmals eine Strategie bei der Strukturentwicklung auf, die zumindest teilweise funktioniert.

10.4 Ausblick

In genereller Betrachtung erscheint - den Versuchen mit NEAT, dem evolutionären Baukasten und EANT nach zu urteilen - bei der Konstruktion eines neuronalen Netzwerkes die Gestaltung (Strukturentwicklung) bedeutsamer zu sein, als die Ausgestaltung (Gewichtsoptimierung), weil diese Möglichkeiten erschafft und nicht nur modifiziert.
Auf Ebene der Netzbestandteile wird dabei erwartet, dass die Verbindungsgewichte in ihrer derzeit gebräuchlichen Verwendung zur Ausgestaltung nicht notwendig sind. Vielmehr könnten sie als Zustandsinformation über die topologisch bedingte Korrelation zwischen Verbindungen[113], welche die Analyse des CMA-ES-Algorithmus aufzeigt, genutzt werden, um z.B. als Maß der Wichtigkeit einer Verbindung bei der Gestaltung eingesetzt zu werden.
Außerdem bestätigt die Untersuchung von TS tendenziell die Vermutung, dass in einem (chaotischen) Netzwerk bei einer Netzeingabe allein durch die (binäre) Impulsweitergabe ein reichhaltiges und charakteristisches Muster von Aktivierungswerten (Informationsmengen) entstehen[114] kann, das zur Erzeugung guter Lösungsvor-

[113] Gemeint sind die Aktivierungswerte des Urspungsneurons der Verbindung.
[114] Dieser Idee weit folgend, könnte das Bewußtsein im Chaos der zahlreichen Verbindungen begründet liegen, von denen beständig einige durch innere oder äußere Reize aktiv gehalten werden, da ohne eine Eingabe gar nichts passieren dürfte.

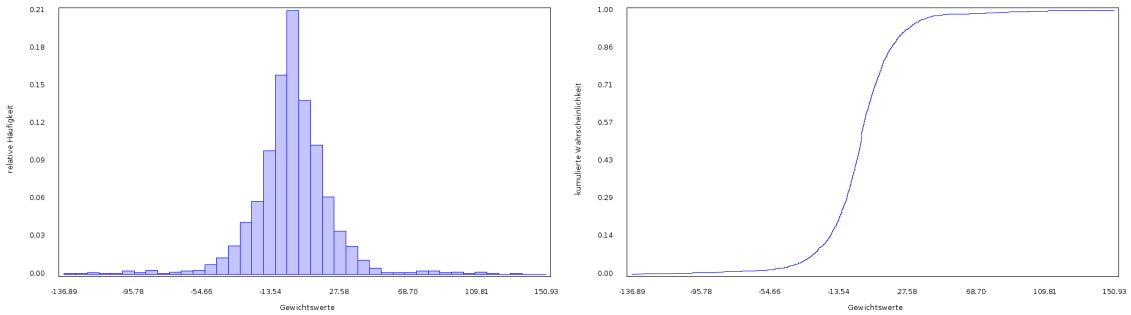


Abbildung 31: Histogramm und CDF der Gewichte

schläge ausreicht. Die sich hier abzeichnende Möglichkeit zur konstruktiven Strukturerzeugung sollte ebenfalls weiter verfolgt werden.
Neben diesen Überlegungen zeigt sich ferner eine bislang unbeschriebene Eigenschaft Neuronaler Netzwerke. Sowohl die optimierten Gewichte in EANT (CMA-ES, SDS oder SR) als auch die Koeffizienten aus der Linearkombination von TS weisen eine der Glockenform ähnliche Verteilung auf, wie Abbildung 31 für den Fall einer Gewichtsoptimierung veranschaulicht. Die neuronale Funktion zeigt demnach, vermutlich aufgrund ihrer chaotischen Eigenschaft, ein Verhalten, das dem eines datengenerierenden bzw. stochastischen Prozesses ähnelt. Die Hypothese, dass Chaos und Zufall kausal zusammenhängen, ist aber (vorerst) nur sehr vage, obgleich chaotische Funktionen auch zur Erzeugung von Pseudozufallszahlen verwendet werden und vergleichbare Phänomene für chaotische Funktionen vielfältig zu beobachten sind.
Diese Fragen legen einen umfangreichen Forschungsbedarf offen. Zudem ist die Trennung zwischen Gewichten und Struktur bei der Betrachtung Neuronaler Netzwerke lediglich eine von vielen möglichen Sichtweisen. Die Suche nach geeigneten Beschreibungen und Modellen zum Erfassen des Wesen neuronaler Netze (Informationsmengen) geht also weiter. 42.

Anhang

Strukturaufzählungen

1 Eingabeneuron, maximal 5 eingehende Verbindungen										
Level k	1	2	3	4	5	6	7	8	9	10
Anzahl der Neuronen	3	6	37	68	99	130	161	192	223	254
Anzahl der Verbindungen	3	10	121	232	343	454	565	676	787	898

1 Eingabeneuron, maximal 10 eingehende Verbindungen										
Level k	1	2	3	4	5	6	7	8	9	10
Anzahl der Neuronen	3	6	37	1.060	2.083	3.106	4.129	5.152	6.175	7.198
Anzahl der Verbindungen	3	10	121	6.264	12.407	18.550	24.693	30.836	36.979	43.122

1 Eingabeneuron, maximal 20 eingehende Verbindungen										
Level k	1	2	3	4	5	6	7	8	9	10
Anzahl der Neuronen	3	6	37	1.048.612	2.097.187	3.145.762	4.194.337	5.242.912	6.291.487	7.340.062
Anzahl der Verbindungen	3	10	121	11.534.456	23.068.791	34.603.126	46.137.461	57.671.796	69.206.131	80.740.466

1 Eingabeneuron, maximal 30 eingehende Verbindungen										
Level k	1	2	3	4	5	6	7	8	9	10
Anzahl der Neuronen	3	6	37	1.073.741.860	$>$2.2e9	-	-	-	-	-
Anzahl der Verbindungen	3	10	121	17.179.869.304	$>$ 3.4e10	-	-	-	-	-

1 Eingabeneuron, maximal 40 eingehende Verbindungen										
Level k	1	2	3	4	5	6	7	8	9	10
Anzahl der Neuronen	3	6	37	$>$4e9	-	-	-	-	-	-
Anzahl der Verbindungen	3	10	121	$>$4e10	-	-	-	-	-	-

Tabelle 33: Strukturgrößen aus Gleichung (12) für 1 Eingabeneuron

10 Eingabeneuronen, maximal 5 eingehende Verbindungen										
Level k	1	2	3	4	5	6	7	8	9	10
Anzahl der Neuronen	1.034	73	104	135	166	197	228	259	290	321
Anzahl der Verbindungen	6.153	232	343	454	565	676	787	898	1009	1120

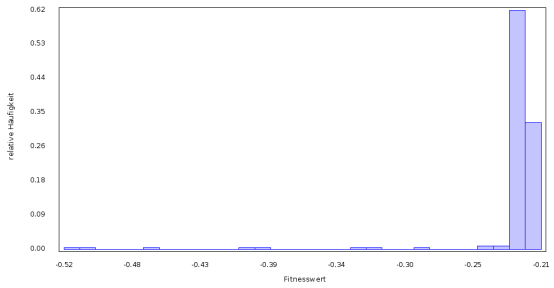
10 Eingabeneuronen, maximal 10 eingehende Verbindungen										
Level k	1	2	3	4	5	6	7	8	9	10
Anzahl der Neuronen	1.034	2.057	3.080	4.103	5.126	6.149	7.172	8.195	9.218	1.0241
Anzahl der Verbindungen	6.153	12.296	18.439	24.582	30.725	36.868	43.011	49.154	55.297	61.440

10 Eingabeneuronen, maximal 20 eingehende Verbindungen										
Level k	1	2	3	4	5	6	7	8	9	10
Anzahl der Neuronen	1.034	1.049.609	2.098.184	3.146.759	4.195.334	5.243.909	6.292.484	7.341.059	8.389.634	9.438.209
Anzahl der Verbindungen	6.153	11.540.488	23.074.823	34.609.158	46.143.493	57.677.828	69.212.163	80.746.498	92.280.833	103.815.168

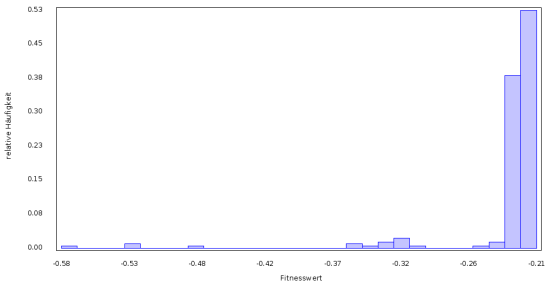
10 Eingabeneuronen, maximal 30 eingehende Verbindungen										
Level k	1	2	3	4	5	6	7	8	9	10
Anzahl der Neuronen	1.034	1.073.742.857	>2.2e9	-	-	-	-	-	-	-
Anzahl der Verbindungen	6.153	17.179.875.336	> 3.4e10	-	-	-	-	-	-	-

10 Eingabeneuronen, maximal 40 eingehende Verbindungen										
Level k	1	2	3	4	5	6	7	8	9	10
Anzahl der Neuronen	1.034	>4e9	-	-	-	-	-	-	-	-
Anzahl der Verbindungen	6.153	>4e910	-	-	-	-	-	-	-	-

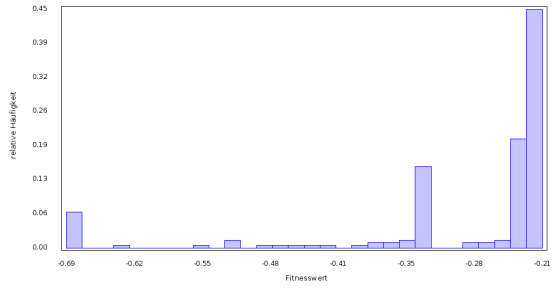
Tabelle 34: Strukturgrößen aus Gleichung (12) für 10 Eingabeneuronen



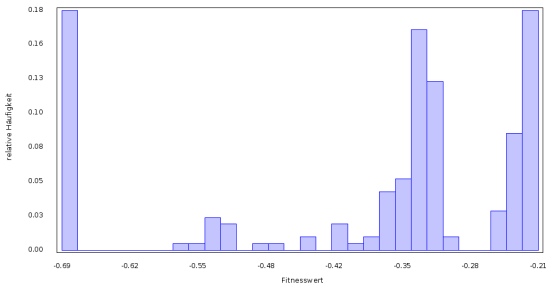
Schrittweite 0.1



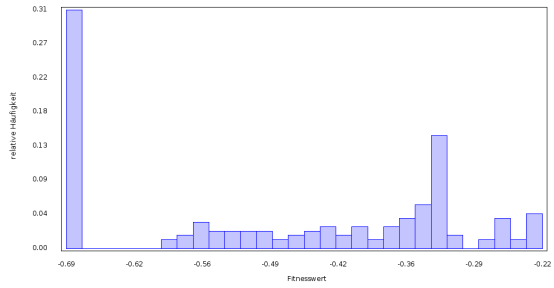
Schrittweite 0.2



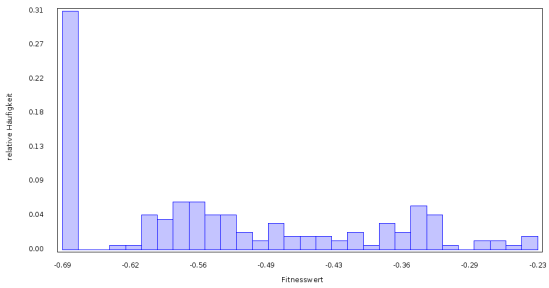
Schrittweite 0.5



Schrittweite 1.0



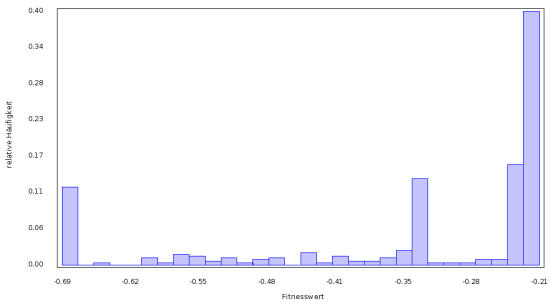
Schrittweite 2.0



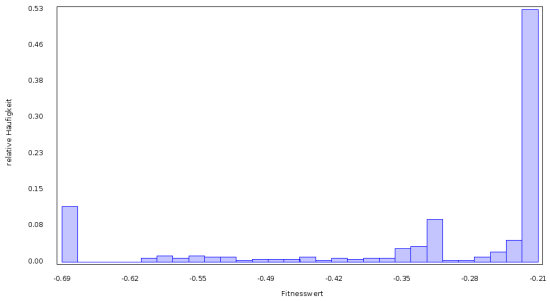
Schrittweite 5.0

(Histogramme auf Basis von 30 Urnen)

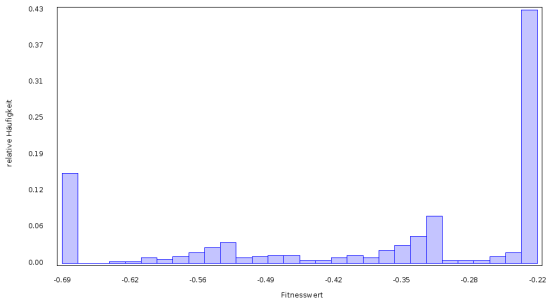
Abbildung 32: Paneldatenhistogramme der initialen Schrittweiten



zufällige initiale Gewichte



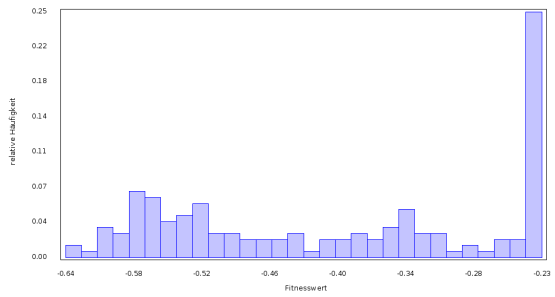
nullgesetzte initiale Gewichte



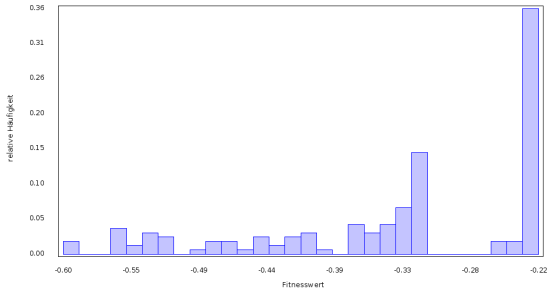
Gewichte aus einer früheren Optimierung

(Histogramme auf Basis von 30 Urnen)

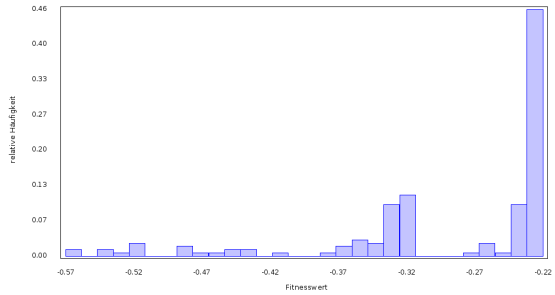
Abbildung 33: Paneldatenhistogramme der initialen Gewichtswahlen



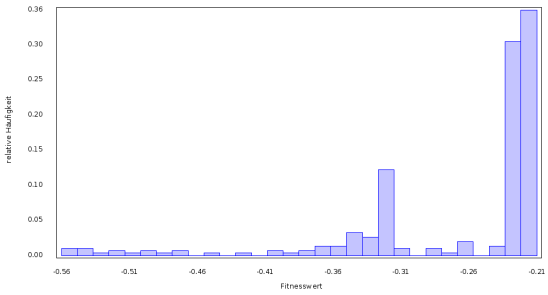
9 Individuen



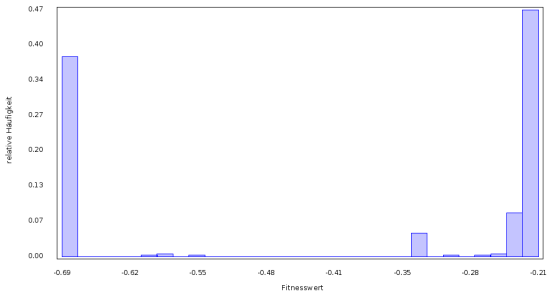
17 Individuen



34 Individuen



85 Individuen



170 Individuen

(Histogramme auf Basis von 30 Urnen)

Abbildung 34: Paneldatenhistogramme der Bevölkerungsgrößen

Parameter \ Versuch	**1**	**2**	3	**4**	**5**	**6**	**7**
PopSize	*30*	*30*	*30*	*30*	*150*	*150*	*1500*
FeatureSelection	ja	nein	ja	ja	ja	ja	ja
PAddLink	0.0025	0.0025	0.0025	0.0025	0.0025	0.0025	0.0025
PAddNode	0.00125	0.00125	0.00125	0.00125	0.00125	0.00125	0.00125
PMutation	0.25	0.25	0.25	0.25	0.25	0.25	0.25
MaxPerturb	0.5	0.5	0.5	0.5	0.5	0.5	0.5
PMutateBias	0.25	0.25	0	0	0	0	0
MaxBiasPerturb	0.1	0.1	0.1	0.1	0.1	0.1	0.1
PToggleLink	0.0025	0	0.0001	0.0001	0.0001	0.0001	0.0001
PWeightReplaced	0.25	0.25	0.85	0.85	0.85	0.85	0.85
SurvivalThreshold	0.5	0.5	0.3	0.3	0.3	0.3	0.3
TerminationValue	0	0	0	0	0	0	0
SpecieCount	*5*	*5*	*5*	*5*	*15*	*15*	*15*
MaxSpecieAge	100000	100000	100000	100000	100000	100000	100000
SpecieYouthThreshold	10	10	10	10	10	10	10
SpecieAgeThreshold	80	80	80	80	80	80	80
YouthBoost	8.8	8.8	8.8	8.8	8.8	8.8	8.8
AgePenalty	1.2	1.2	1.2	1.2	1.2	1.2	1.2
DisjointCoeff	1	1	1	1	1	1	1
ExcessCoeff	1	1	1	1	1	1	1
WeightCoeff	3	3	1	1	1	1	1
CompatabilityChange	0.1	0.1	3	3	3	3	0.1
(Initial) Threshold	0	0	0	0	0	0	0
Laufzeit in Epochen	20795	21523	100000	100000	100000	100000	4222
letzte Verbesserung	1221	3156	99018	99915	97365	98989	4171
Fevals	37851	97836	3.1e6	3.1e6	1.51e7	1.51e7	6.3e6
Netzgröße	15	98	17	22	15	17	15
Fitness	0.649	0.6448	0.3777	0.3772	0.3319	0.3296	0.4196
Bemerkung	mit Bias		nur die Ausgabeneuronen besitzen einen Bias				

Tabelle 36: Ergebnisse zu NEAT, Februar und März 2007 (Vorversuche)

Parameter \ Versuch	**1**	**2**	**3**	**4**	**5**	**6**	**7**
PopSize	*30*	*30*	*150*	*150*	*150*	*1500*	*1500*
FeatureSelection	ja	ja	ja	ja	ja	ja	ja
PAddLink	0.0025	0.0025	0.0025	0.0025	0.0025	0.0025	0.0025
PAddNode	0.00125	0.00125	0.00125	0.00125	0.00125	0.00125	0.00125
PMutation	0.25	0.25	0.25	0.25	0.25	0.25	0.25
MaxPerturb	0.5	0.5	0.5	0.5	0.5	0.5	0.5
PMutateBias	0	0	0	0	0	0	0
MaxBiasPerturb	-	-	-	-	-	-	-
PToggleLink	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001
PWeightReplaced	0.85	0.85	0.85	0.85	0.85	0.85	0.85
SurvivalThreshold	0.3	0.3	0.3	0.3	0.3	0.3	0.3
TerminationValue	0	0	0	0	0	0	0
SpecieCount	*1*	*5*	*1*	*5*	*15*	*1*	*15*
MaxSpecieAge	100000	100000	100000	100000	100000	100000	100000
SpecieYouthThreshold	10	10	10	10	10	10	10
SpecieAgeThreshold	80	80	80	80	80	80	80
YouthBoost	8.8	8.8	8.8	8.8	8.8	8.8	8.8
AgePenalty	1.2	1.2	1.2	1.2	1.2	1.2	1.2
DisjointCoeff	1	1	1	1	1	1	1
ExcessCoeff	1	1	1	1	1	1	1
WeightCoeff	3	3	3	3	3	3	3
CompatabilityChange	0.1	0.1	0.1	0.1	0.1	0.1	0.1
(Initial) Threshold	0	0	0	0	0	0	0
Laufzeit in Epochen	100000	100000	100000	100000	100000	100000	100000
letzte Verbesserung	99975	497	71840	914	881	20312	558
Fevals	3.1e6	3.1e6	1.51e7	5e6	3e6	1.501e8	3e7
Netzgröße	17	11	16	14	14	12	13
Fitness	0.3173	0.3952	0.314	0.3441	0.5402	0.3143	0.3498
Bemerkung	Variationen der Bevölkerungsgröße - ohne Bias						

Tabelle 38: Ergebnisse zu NEAT, 06.04. bis 16.04.2007

Parameter \ Versuch	**1**	**2**	**3**	**4**	**5**	**6**	**7**
PopSize	*30*	*30*	*150*	*150*	*150*	*1500*	*1500*
FeatureSelection	ja	ja	ja	ja	ja	ja	ja
PAddLink	0.0025	0.0025	0.0025	0.0025	0.0025	0.0025	0.0025
PAddNode	0.00125	0.00125	0.00125	0.00125	0.00125	0.00125	0.00125
PMutation	0.25	0.25	0.25	0.25	0.25	0.25	0.25
MaxPerturb	0.5	0.5	0.5	0.5	0.5	0.5	0.5
PMutateBias	0	0	0	0	0	0	0
MaxBiasPerturb	-	-	-	-	-	-	-
PToggleLink	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001
PWeightReplaced	0.85	0.85	0.85	0.85	0.85	0.85	0.85
SurvivalThreshold	0.3	0.3	0.3	0.3	0.3	0.3	0.3
TerminationValue	0	0	0	0	0	0	0
SpecieCount	*1*	*5*	*1*	*5*	*15*	*1*	*15*
MaxSpecieAge	100000	100000	100000	100000	100000	100000	100000
SpecieYouthThreshold	10	10	10	10	10	10	10
SpecieAgeThreshold	80	80	80	80	80	80	80
YouthBoost	8.8	8.8	8.8	8.8	8.8	8.8	8.8
AgePenalty	1.0	1.0	1.0	1.0	1.0	1.0	1.0
DisjointCoeff	1	1	1	1	1	1	1
ExcessCoeff	1	1	1	1	1	1	1
WeightCoeff	3	3	3	3	3	3	3
CompatabilityChange	0.1	0.1	0.1	0.1	0.1	0.1	0.1
(Initial) Threshold	0	0	0	0	0	0	0
Laufzeit in Epochen	100000	100000	100000	100000	100000	100000	100000
letzte Verbesserung	99573	96818	94316	82928	8141	6599	15433
Fevals	3.1e6	3.1e6	1.51e7	1.51e7	1.51e7	5e7	5e7
Netzgröße	18	23	12	16	14	9	28
Fitness	0.316	0.3801	0.3135	0.3143	0.4044	0.3136	0.3476
Bemerkung	Variationen der Bevölkerungsgröße - ohne Bias						

Tabelle 40: Ergebnisse zu NEAT, 16.04. bis 30.04.2007

Parameter \ Versuch	**1**	**2**	**3**	**4**	**5**	**6**	**7**
PopSize	*30*	*30*	*150*	*150*	*150*	*1500*	*1500*
FeatureSelection	ja	ja	ja	ja	ja	ja	ja
PAddLink	0.0025	0.0025	0.0025	0.0025	0.0025	0.0025	0.0025
PAddNode	0.00125	0.00125	0.00125	0.00125	0.00125	0.00125	0.00125
PMutation	0.25	0.25	0.25	0.25	0.25	0.25	0.25
MaxPerturb	0.5	0.5	0.5	0.5	0.5	0.5	0.5
PMutateBias	0	0	0	0	0	0	0
MaxBiasPerturb	-	-	-	-	-	-	-
PToggleLink	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001
PWeightReplaced	0.85	0.85	0.85	0.85	0.85	0.85	0.85
SurvivalThreshold	0.3	0.3	0.3	0.3	0.3	0.3	0.3
TerminationValue	0	0	0	0	0	0	0
SpecieCount	*1*	*5*	*1*	*5*	*15*	*1*	*15*
MaxSpecieAge	100000	100000	100000	100000	100000	100000	100000
SpecieYouthThreshold	10	10	10	10	10	10	10
SpecieAgeThreshold	80	80	80	80	80	80	80
YouthBoost	1.0	1.0	1.0	1.0	1.0	1.0	1.0
AgePenalty	1.0	1.0	1.0	1.0	1.0	1.0	1.0
DisjointCoeff	1	1	1	1	1	1	1
ExcessCoeff	1	1	1	1	1	1	1
WeightCoeff	3	3	3	3	3	3	3
CompatabilityChange	0.1	0.1	0.1	0.1	0.1	0.1	0.1
(Initial) Threshold	0	0	0	0	0	0	0
Laufzeit in Epochen	100000	100000	100000	100000	100000	100000	100000
letzte Verbesserung	98923	97000	99467	99319	72544	33099	17670
Fevals	3.1e6	3.1e6	1.51e7	1.51e7	1.51e7	1.501e8	1.501e8
Netzgröße	18	22	11	20	16	12	19
Fitness	0.366	0.3161	0.314	0.3275	0.3549	0.3367	0.3146
Bemerkung	Variationen der Bevölkerungsgröße - ohne Bias						

Tabelle 42: Ergebnisse zu NEAT, 03.05. bis 01.06.2007

Parameter \ Versuch	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
PopSize	150	150	150	150	150	150	150	150	150
FeatureSelection	ja	ja	ja	ja	ja	ja	ja	ja	ja
PAddLink	0.0025	0.0025	0.0025	0.0025	0.0025	0.0025	0.0025	0.0025	0.0025
PAddNode	0.00125	0.00125	0.00125	0.00125	0.00125	0.00125	0.00125	0.00125	0.00125
PMutation	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25
MaxPerturb	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
PMutateBias	0	0	0	0	0	0	0	0	0
MaxBiasPerturb	-	-	-	-	-	-	-	-	-
PToggleLink	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001
PWeightReplaced	0.85	0.85	0.85	0.85	0.85	0.85	0.85	0.85	0.85
SurvivalThreshold	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3
TerminationValue	0	0	0	0	0	0	0	0	0
SpecieCount	15	15	15	15	15	15	15	15	15
MaxSpecieAge	100000	100000	100000	100000	100000	100000	100000	100000	100000
SpecieYouthThreshold	10	10	10	10	10	10	10	10	10
SpecieAgeThreshold	80	80	80	80	80	80	80	80	80
YouthBoost	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
AgePenalty	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
DisjointCoeff	1	1	1	1	1	1	1	1	1
ExcessCoeff	1	1	1	1	1	1	1	1	1
WeightCoeff	3	3	3	3	3	3	3	3	3
CompatabilityChange	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
(Initial) Threshold	0	0	0	0	0	0	0	0	0
Laufzeit in Epochen	100000	100000	100000	100000	100000	100000	35359	98700	96749
letzte Verbesserung	86489	48340	99611	58279	99641	79169	27201	73769	92627
Fevals	1.51e7	1.51e7	1.51e7	1.51e7	1.51e7	1.51e7	5.3e6	1.49e7	1.46e7
Netzgröße	13	21	12	14	19	16	12	15	13
Fitness	0.3891	0.3547	0.3803	0.3896	0.3343	0.3564	0.3492	0.3555	0.3894
Bemerkung	Speziation - ohne Bias								

Tabelle 44: Ergebnisse zu NEAT, 24.06. bis 06.07.2007

Parameter \ Versuch	**1**	**2**	**3**	**4**
PopSize	180	180	180	180
FeatureSelection	ja	ja	ja	ja
PAddLink	0.0025	0.0025	0.0025	0.0025
PAddNode	0.00125	0.00125	0.00125	0.00125
PMutation	0.5	0.5	0.5	0.5
MaxPerturb	0.5	0.5	0.5	0.5
PMutateBias	0	0	0	0
MaxBiasPerturb	-	-	-	-
PToggleLink	0.0001	0.0001	0.0001	0.0001
PWeightReplaced	0.85	0.85	0.85	0.85
SurvivalThreshold	0.3	0.3	0.3	0.3
TerminationValue	0	0	0	0
SpecieCount	15	15	15	15
MaxSpecieAge	500	500	500	500
SpecieYouthThreshold	10	10	10	10
SpecieAgeThreshold	80	80	80	80
YouthBoost	10	10	10	10
AgePenalty	1.0	1.0	1.0	1.0
DisjointCoeff	8	8	8	8
ExcessCoeff	8	8	8	8
WeightCoeff	2	2	2	2
CompatabilityChange	0.1	0.1	0.1	0.1
(Initial) **Threshold**	1	1	1	1
Laufzeit in Epochen	3643	4911	1491	3319
letzte Verbesserung	17	3650	1291	3041
Fevals	659383	888891	269871	600739
Netzgröße	8	7	6	6
Fitness	0.5028	0.4714	0.4937	0.3941
Bemerkung	Extraversuche ohne Bias			

Tabelle 46: Ergebnisse zu NEAT, 11.07. bis 28.07.2007

Parameter \ Versuch	**1**	**2**	**3**	**4**	**5**	**6**	**1b**	**5b**
PopSize	*30*	*30*	*30*	*30*	*30*	*30*	*30*	*30*
FeatureSelection	ja	ja	ja	nein	nein	nein	ja	nein
PAddLink	0.0025	0.025	0.25	0.0025	0.025	0.25	0.0025	0.025
PAddNode	0.00125	0.0125	0.125	0.00125	0.0125	0.125	0.00125	0.0125
PMutation	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25
MaxPerturb	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
PMutateBias	0	0	0	0	0	0	0	0
MaxBiasPerturb	-	-	-	-	-	-	-	-
PToggleLink	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.01	0.01
PWeightReplaced	0.85	0.85	0.85	0.85	0.85	0.85	0.85	0.85
SurvivalThreshold	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3
TerminationValue	0	0	0	0	0	0	0	0
SpecieCount	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*
MaxSpecieAge	100000	100000	100000	100000	100000	100000	100000	100000
SpecieYouthThreshold	10	10	10	10	10	10	10	10
SpecieAgeThreshold	80	80	80	80	80	80	80	80
YouthBoost	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
AgePenalty	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
DisjointCoeff	1	1	1	1	1	1	1	1
ExcessCoeff	1	1	1	1	1	1	1	1
WeightCoeff	3	3	3	3	3	3	3	3
CompatabilityChange	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
(Initial) Threshold	0	0	0	0	0	0	0	0
Laufzeit in Epochen	100000	100000	100000	100000	100000	100000	100000	100000
letzte Verbesserung	96961	92798	91546	98455	93151	48342	97291	99708
Fevals	3.1e6	3.1e6	3.1e6	3.1e6	3.1e6	3.1e6	3.1e6	3.1e6
Netzgröße	11	20	61	34	34	47	9	33
Fitness	0.3135	0.33	0.3148	0.3933	0.3885	0.4123	0.3135	0.388
Bemerkung	Variationen der Strukturmutation - ohne Bias							

Tabelle 48: Ergebnisse zu NEAT, März und August 2007

Parameter \ Versuch	**1**	**2**	**3**	**4**	**5**	**6**	**7**
PopSize	150	150	150	150	150	150	150
FeatureSelection	ja	ja	ja	ja	ja	ja	ja
PAddLink	0.05	0.05	0.05	0.05	0.05	0.05	0.05
PAddNode	0.01	0.01	0.01	0.01	0.01	0.01	0.01
PMutation	0.25	0.25	0.25	0.25	0.25	0.25	0.25
MaxPerturb	0.5	0.5	0.5	0.5	0.5	0.5	0.5
PMutateBias	0	0	0	0	0	0	0
MaxBiasPerturb	-	-	-	-	-	-	-
PToggleLink	0.1	0.1	0.1	0.1	0.1	0.1	0.1
PWeightReplaced	0.85	0.85	0.85	0.85	0.85	0.85	0.85
SurvivalThreshold	0.3	0.3	0.3	0.3	0.3	0.3	0.3
TerminationValue	0	0	0	0	0	0	0
SpecieCount	1	1	1	1	1	1	1
MaxSpecieAge	100000	100000	100000	100000	100000	100000	100000
SpecieYouthThreshold	10	10	10	10	10	10	10
SpecieAgeThreshold	80	80	80	80	80	80	80
YouthBoost	1.0	1.0	1.0	1.0	1.0	1.0	1.0
AgePenalty	1.0	1.0	1.0	1.0	1.0	1.0	1.0
DisjointCoeff	1	1	1	1	1	1	1
ExcessCoeff	1	1	1	1	1	1	1
WeightCoeff	3	3	3	3	3	3	3
CompatabilityChange	0.1	0.1	0.1	0.1	0.1	0.1	0.1
(Initial) Threshold	0	0	0	0	0	0	0
Laufzeit in Epochen	100000	100000	100000	100000	100000	100000	100000
letzte Verbesserung	89809	97160	96800	94185	85173	99904	99985
Fevals	1.51e7	1.51e7	1.51e7	1.51e7	1.51e7	1.51e7	1.51e7
Netzgröße	13	14	18	17	20	15	12
Fitness	0.3135	0.3144	0.3067	0.3052	0.2984	0.3057	0.3118
Bemerkung	Variationen der Gewichtsmutation - ohne Bias						

Tabelle 50: Ergebnisse zu NEAT, 02.08. bis 10.08.2007

Parameter \ Versuch	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
PopSize	150	150	150	150	150	150	150	150	150	150	150	150	150	150
FeatureSelection	ja	ja	ja	ja	ja	ja	ja	ja	ja	ja	ja	ja	ja	ja
PAddLink	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05
PAddNode	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
PMutation	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
MaxPerturb	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
PMutateBias	0	0	0	0	0	0	0	0	0	0	0	0	0	0
MaxBiasPerturb	-	-	-	-	-	-	-	-	-	-	-	-	-	-
PToggleLink	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
PWeightReplaced	0.85	0.85	0.85	0.85	0.85	0.85	0.85	0.85	0.85	0.85	0.85	0.85	0.85	0.85
SurvivalThreshold	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3
TerminationValue	0	0	0	0	0	0	0	0	0	0	0	0	0	0
SpecieCount	1	1	1	1	1	1	1	1	1	1	1	1	1	1
MaxSpecieAge	100000	100000	100000	100000	100000	100000	100000	100000	100000	100000	100000	100000	100000	100000
SpecieYouthThreshold	10	10	10	10	10	10	10	10	10	10	10	10	10	10
SpecieAgeThreshold	80	80	80	80	80	80	80	80	80	80	80	80	80	80
YouthBoost	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
AgePenalty	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
DisjointCoeff	1	1	1	1	1	1	1	1	1	1	1	1	1	1
ExcessCoeff	1	1	1	1	1	1	1	1	1	1	1	1	1	1
WeightCoeff	3	3	3	3	3	3	3	3	3	3	3	3	3	3
CompatabilityChange	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
(Initial) Threshold	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Laufzeit in Epochen	100000	100000	100000	100000	100000	100000	100000	100000	100000	100000	100000	100000	100000	100000
letzte Verbesserung	97952	99520	97806	99724	96077	96768	87049	98721	91222	99891	99726	97989	95154	94440
Fevals	1.51e7	1.51e7	1.51e7	1.51e7	1.51e7	1.51e7	1.51e7	1.51e7	1.51e7	1.51e7	1.51e7	1.51e7	1.51e7	1.51e7
Netzgröße	9	9	10	13	9	9	8	11	8	11	8	14	8	9
Fitness	0.3136	0.3148	0.3136	0.313	0.3139	0.314	0.3138	0.3136	0.3138	0.3133	0.3139	0.3127	0.3136	0.3139
Bemerkung	Variationen der Gewichtsmutation - ohne Bias													

Tabelle 51: Ergebnisse zu NEAT, 29.08. bis 08.09.2007

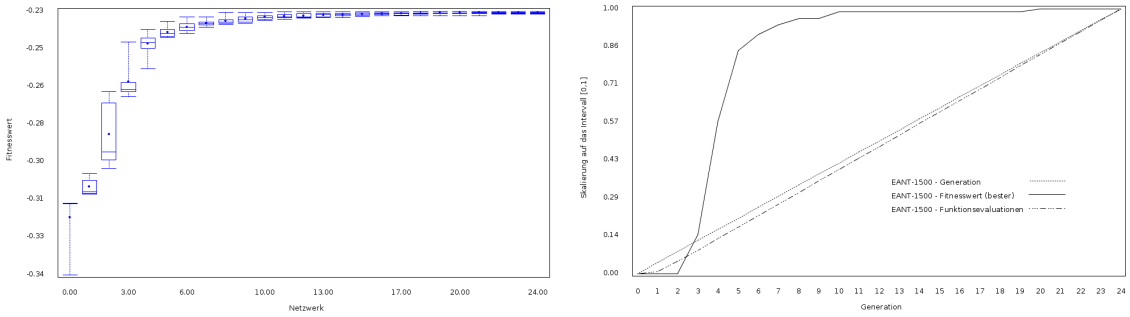


Abbildung 35: Fitness- und einheitlich skalierte Evolutionsverläufe zur Testreihe EANT-1500

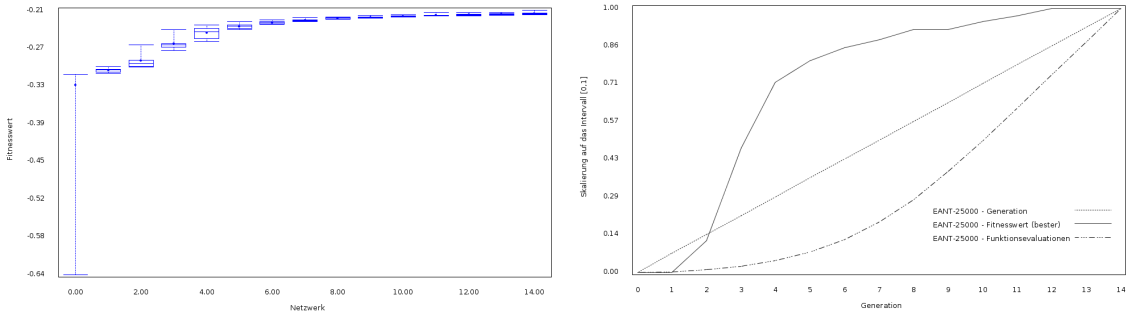


Abbildung 36: Fitness- und einheitlich skalierte Evolutionsverläufe zur Testreihe EANT-25000

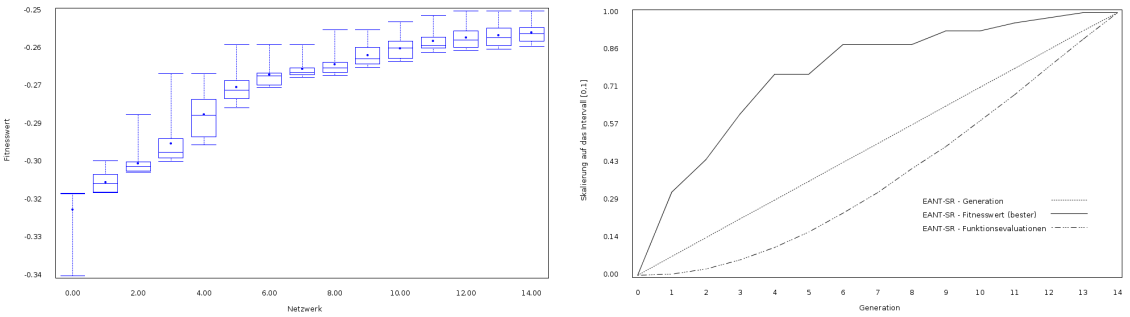


Abbildung 37: Fitness- und einheitlich skalierte Evolutionsverläufe zur Testreihe EANT-SR

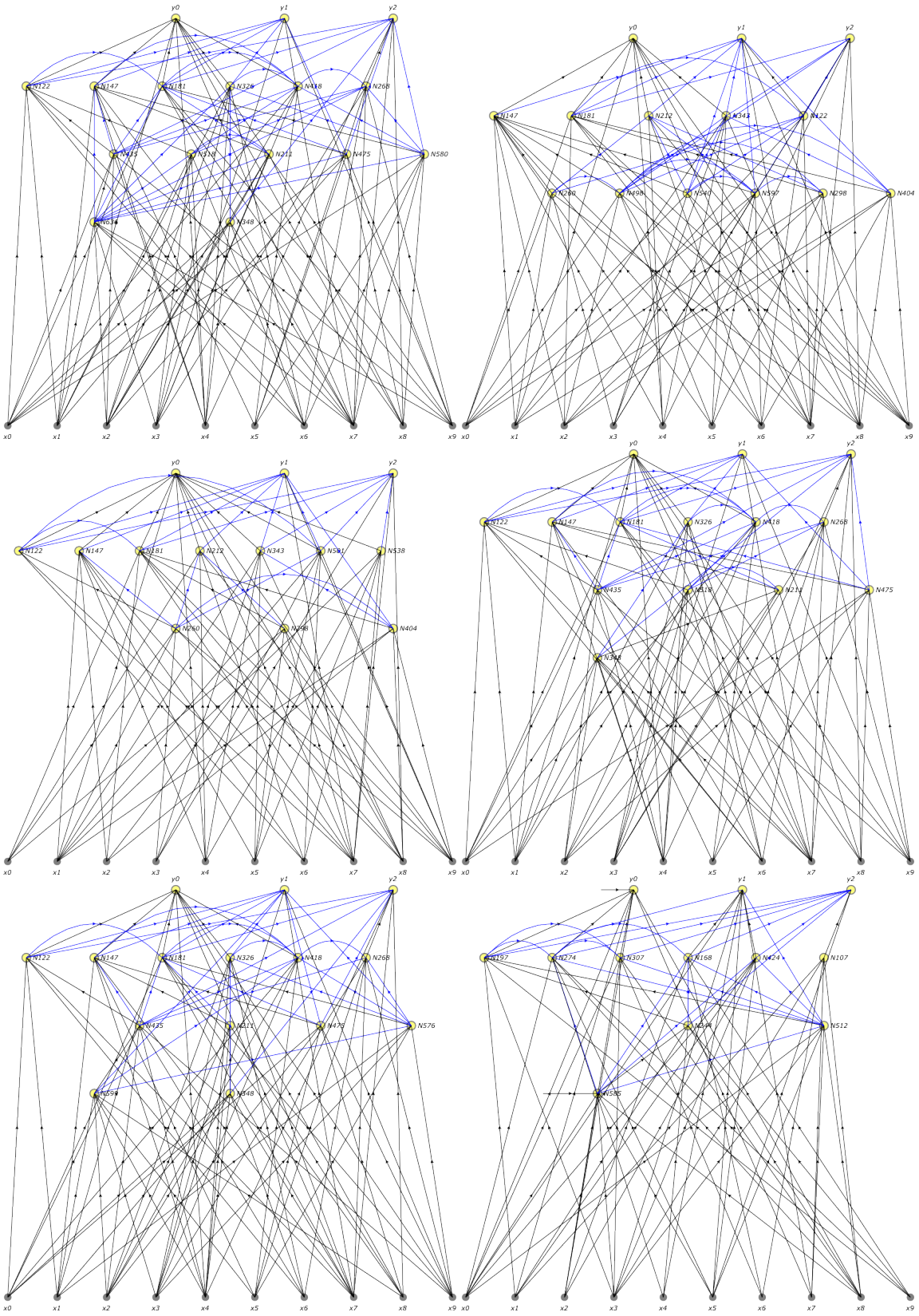


Abbildung 38: Zufällig ausgewählte Netzwerke der besten 14. EANT-Generation

Legende

K / C	:	„Kanonische“ / „Chaotische“ Datenstruktur
#inputs	:	Anzahl der Eingabeneuronen.
#hidden	:	Anzahl der verdeckten Neuronen.
#outputs	:	Anzahl der Ausgabeneuronen.
#nin / #nout	:	Anzahl der Neuronen mit eingehenden / ausgehenden Verbindungen.
$[a, b]$	:	Ziehung einer gleichverteilten (diskreten) Zufallsvariablen im Intervall $[a, b]$.
name	:	Steht eine Variablenbezeichnung in der Tabelle, so werden die Aktionen gemeinsam ausgeführt.

Datenstruktur	C	C	C
Überleben des Besten	ja	ja	ja
Selektionskriterium	alle	alle	alle
Wkt. für Rekombination	-	-	-
Entferne alle Verbindungen des Schlechteren	-	-	-
Überlebenswkt. pro Verbindungen des Besseren	-	-	-
Überlebenswkt. aller Verbindungen des Besseren	-	-	-
Wkt. für neue V. $pl1$	-	50%	$1 - ps1$
Max. hinzuzufügende V. $l1$	-	#inputs	$\left[1, \frac{\#\text{inputs}}{2}\right]$
Wkt. für das Entfernen einer V. $pl0$ nach Art des Hinzufügens	$1 - ps1$	50%	-
Max. zu entfernende V. $l0$ nach Art des Hinzufügens	$[1, \#\text{nout}^2]$	#inputs	-
Wkt. für das Entfernen einer V. $pl0$ aus der Menge existierender V.	-	25%	$1 - pm0$
Max. zu entfernende V. $l0$ aus der Menge existierender V.	-	#inputs	$\left[1, \frac{\#\text{inputs}}{2}\right]$
Intervall für 1 neues Neuron	-	-	-
Wkt. für neue Neuronen $pn1$	-	0.1	-
Max. hinzuzufügende N. $n1$	-	1	-
Wkt. für das Entfernen eines N. $pn0$	-	0.1	50%
Max. zu entfernende N. $n0$	-	1	1
Wkt. für neue Teilnetze $ps1$	50%	50%	50%
Max. hinzuzufügende T. $s1$	1	1	1
Anzahl der Teilnetzeingänge	$[1, \#\text{nout}]$	$[1, \#\text{nout}]$	$[1, \#\text{nout}]$
Anzahl der Teilnetzausgänge	1	$[1, \#\text{nin}]$	$[1, \#\text{nin}]$
Minimierungsverfahren	SR	SR	SR

Tabelle 52: Spaltenweise Charakterisierung von EA, Teil 1

Datenstruktur	K	K	K	K	K	K	K	K
Überleben des Besten	nein	nein	nein	nein	nein	ja	ja	ja
Selektionskriterium	alle (1)	alle (1)	alle (1)	alle (1)	alle (1)	alle	alle	alle
Wkt. für Rekombination	-	-	-	-	-	-	100%	100%
Entferne alle Verbindungen des Schlechteren	-	-	-	-	-	-	ja	ja
Überlebenswkt. pro Verbindungen des Besseren	-	-	-	-	-	-	85%	85%
Überlebenswkt. aller Verbindungen des Besseren	-	-	-	-	-	-	-	-
Wkt. für neue V. $pl1$	100%	66%	50%	66%	50%	75%	15%	15%
Max. hinzuzufügende V. $l1$	1	1	1	1	1	1	alle	alle
Wkt. für das Entfernen einer V. $pl0$ nach Art des Hinzufügens	-	-	-	33%	50%	25%	-	-
Max. zu entfernende V. $l0$ nach Art des Hinzufügens	-	-	-	1	1	1	-	-
Wkt. für das Entfernen einer V. $pl0$ aus der Menge existierender V.	-	33%	50%	-	-	-	-	-
Max. zu entfernende V. $l0$ aus der Menge existierender V.	-	1	1	-	-	-	-	-
Intervall für 1 neues Neuron	-	-	-	-	-	-	-	-
Wkt. für neue Neuronen $pn1$	3.3%	3.3%	3.3%	3.3%	3.3%	3%	3%	-
Max. hinzuzufügende N. $n1$	1	1	1	1	1	1	1	-
Wkt. für das Entfernen eines N. $pn0$	-	3.3%	3.3%	3.3%	3.3%	3%	3%	-
Max. zu entfernende N. $n0$	-	1	1	1	1	1	1	-
Wkt. für neue Teilnetze $ps1$	-	-	-	-	-	-	-	-
Max. hinzuzufügende T. $s1$	-	-	-	-	-	-	-	-
Anzahl der Teilnetzeingänge	-	-	-	-	-	-	-	-
Anzahl der Teilnetzausgänge	-	-	-	-	-	-	-	-
Minimierungsverfahren	SR	SR	SR	SR	SR	SR	SR	SR

Tabelle 53: Spaltenweise Charakterisierung von EA, Teil 2

Datenstruktur	K	K	K	K	K	K	K
Überleben des Besten	ja	ja	ja	ja	ja	ja	ja
Selektionskriterium	alle	alle	alle	alle	alle	alle	alle
Wkt. für Rekombination	-	100%	100%	100%	100%	100%	-
Entferne alle Verbindungen des Schlechteren	-	ja	ja	ja	ja	ja	-
Überlebenswkt. pro Verbindungen des Besseren	-	-	95%	85%	-	-	-
Überlebenswkt. aller Verbindungen des Besseren	-	$\sim 90\%$	-	-	90%	90%	-
Wkt. für neue V. $pl1$	50%	50%	100%	15%	100%	100%	-
Max. hinzuzufügende V. $l1$	#inputs	#nwi	$[1, \#\text{hidden}]$	$[1, \#\text{nout}^2]$	$[1, \#\text{nout}^2]$	$[1, \#\text{hidden}]$	-
Wkt. für das Entfernen einer V. $pl0$ nach Art des Hinzufügens	50%	-	-	-	-	-	$1 - ps1$
Max. zu entfernende V. $l0$ nach Art des Hinzufügens	#inputs	-	-	-	-	-	$[1, \#\text{nout}^2]$
Wkt. für das Entfernen einer V. $pl0$ aus der Menge existierender V.	25%	-	-	-	-	-	-
Max. zu entfernende V. $l0$ aus der Menge existierender V.	#inputs	-	-	-	-	-	-
Intervall für 1 neues Neuron	-	-	-	30	-	-	-
Wkt. für neue Neuronen $pn1$	0.1	-	-	-	-	-	-
Max. hinzuzufügende N. $n1$	1	-	-	-	-	-	-
Wkt. für das Entfernen eines N. $pn0$	0.1	-	-	-	-	-	-
Max. zu entfernende N. $n0$	1	-	-	-	-	-	-
Wkt. für neue Teilnetze $ps1$	50%	-	-	-	-	100%	50%
Max. hinzuzufügende T. $s1$	1	-	-	-	-	1	1
Anzahl der Teilnetzeingänge	$[1, \#\text{nout}]$	-	-	-	-	$[1, \#\text{nout}]$	$[1, \#\text{nout}]$
Anzahl der Teilnetzausgänge	$[1, \#\text{nin}]$	-	-	-	-	1	1
Minimierungsverfahren	SR	SR	SR	SR	SR	SR	SR

Tabelle 54: Spaltenweise Charakterisierung von EA, Teil 3

Skalierung	Neuronenanz.	mit Biasneuron			ohne Biasneuron		
z		dataset	trainingsset	testset	dataset	trainingsset	testset
0.5	0	3.982	3.9562	4.356	4.1432	4.1232	4.4506
	500	0.1351	0.136	0.7563	0.3494	0.3594	1.0019
	1000	0.1447	0.1558	0.7977	0.3447	0.3536	1.1136
	2000	0.1478	0.1564	0.7955	0.3523	0.3767	1.3333
	5000	0.1638	0.1753	0.8189	0.3649	0.3893	1.3574
	10000	0.1702	0.1861	0.8433	0.3644	0.3864	1.3036
1	0	3.982	3.9562	4.356	4.1432	4.1232	4.4506
	500	0.0998	0.1004	0.739	0.1681	0.1831	0.7527
	1000	0.1096	0.1121	0.7414	0.1751	0.1813	0.7474
	2000	0.11	0.1113	0.7844	0.2079	0.2201	0.8407
	5000	0.1214	0.1247	0.8443	0.2112	0.2261	0.8805
	10000	0.126	0.1313	0.8306	0.2077	0.2208	0.8303
2	0	3.982	3.9562	4.356	4.1432	4.1232	4.4506
	500	0.0786	0.0846	0.5527	0.0812	0.089	0.5395
	1000	0.0852	0.0898	0.5613	0.0848	0.0949	0.5378
	2000	0.0794	0.0865	0.5247	0.0815	0.0954	0.5274
	5000	0.0914	0.0928	0.6062	0.1019	0.1088	0.6347
	10000	0.0922	0.098	0.6433	0.0891	0.1075	0.6273
4	0	3.982	3.9562	4.356	4.1432	4.1232	4.4506
	500	0.0721	0.0725	0.3687	0.0718	0.0723	**0.3141**
	1000	0.0748	0.0767	0.4037	0.0728	0.0724	0.3422
	2000	0.0746	0.0754	**0.3636**	0.0722	0.0742	0.3424
	5000	0.0748	0.076	0.4041	0.076	0.0765	0.3865
	10000	0.0756	0.0796	0.4599	0.0767	0.0783	0.3903
10	0	3.982	3.9562	4.356	4.1432	4.1232	4.4506
	500	0.0672	0.0655	0.6095	0.0576	0.0612	0.4585
	1000	0.0691	0.0694	0.5731	**0.0574**	**0.0604**	0.4786
	2000	**0.0608**	**0.064**	0.548	0.0617	0.0624	0.4949
	5000	0.0671	0.0656	0.5051	0.0646	0.0659	0.4916
	10000	0.0743	0.071	0.5208	0.0725	0.0705	0.4853
15	0	3.982	3.9562	4.356	4.1432	4.1232	4.4506
	500	0.1076	0.1063	0.5461	0.1106	0.1062	0.5403
	1000	0.0975	0.0987	0.5529	0.0887	0.0891	0.5956
	2000	0.0953	0.0933	0.5914	0.0874	0.0881	0.5597
	5000	0.0883	0.0896	0.5279	0.0813	0.0818	0.5325
	10000	0.0959	0.0933	0.4728	0.0864	0.09	0.495
(Transformation der Eingabe, Netzwerke mit Biasneuron, Linearkombination mit Konstante)							

Tabelle 55: Ergebnisse zum ersten Datensatz und der Funktion AbsAZ

Skalierung	Neuronenanz.	mit Biasneuron			ohne Biasneuron		
z		dataset	trainingsset	testset	dataset	trainingsset	testset
0.5	0	3.7635	3.7401	4.0568	3.9049	3.887	4.1385
	500	0.104	0.1162	0.7086	0.2899	0.2901	1.2899
	1000	0.1249	0.1335	0.6922	0.2731	0.2845	1.3147
	2000	0.1247	0.1331	0.7631	0.2911	0.3065	1.7889
	5000	0.1358	0.1687	0.7318	0.299	0.3266	1.8858
	10000	0.1506	0.1662	0.7337	0.3039	0.3339	1.625
1	0	3.647	3.6247	3.8928	3.772	3.755	3.9609
	500	0.0682	0.0719	0.5799	0.1653	0.1728	0.8731
	1000	0.0754	0.08	0.5773	0.146	0.1744	0.6808
	2000	0.0757	0.0821	0.5896	0.1909	0.1967	0.8723
	5000	0.0839	0.0956	0.6386	0.1904	0.1989	0.9577
	10000	0.0813	0.0964	0.693	0.1905	0.1991	0.8628
2	0	3.5346	3.5133	3.7312	3.6351	3.6188	3.7765
	500	0.0375	0.0439	0.3701	0.0637	0.0695	0.5034
	1000	0.0432	0.0482	0.3774	0.0581	0.0666	0.4092
	2000	0.0425	0.0476	0.3305	0.0684	0.0741	0.5077
	5000	0.0453	0.0509	0.3842	0.0711	0.0793	0.6
	10000	0.0459	0.053	0.4475	0.0712	0.0805	0.5745
4	0	3.458	3.4371	3.6202	3.53	3.5137	3.6371
	500	0.0212	0.0257	0.1736	0.0214	0.0268	0.2759
	1000	0.024	0.0272	0.1807	0.026	0.0293	0.1746
	2000	0.0213	0.0296	0.1217	0.0278	0.0322	0.2253
	5000	0.0239	0.0277	0.1616	0.0301	0.0344	0.2632
	10000	0.0248	0.0282	0.2047	0.0287	0.0335	0.2427
10	0	3.4143	3.3935	3.5598	3.4559	3.4393	3.5457
	500	0.0070	0.0078	0.0794	0.0060	0.0067	0.0474
	1000	0.0072	0.0082	0.0685	0.0065	0.0079	0.0594
	2000	0.0066	0.0082	0.0435	0.0080	0.0092	0.0679
	5000	0.0081	0.0093	0.0574	0.0090	0.0109	0.0872
	10000	0.0088	0.0114	0.059	0.0096	0.0103	0.0682
15	0	3.4073	3.3864	3.5513	3.4401	3.4234	3.5284
	500	0.0059	0.0065	0.0567	0.0046	**0.0051**	0.06
	1000	0.0057	0.0066	0.0514	**0.0043**	0.0053	0.0518
	2000	**0.0051**	0.0065	**0.0296**	0.0054	0.0061	0.0446
	5000	0.0052	**0.0061**	0.0421	0.0056	0.0066	0.0502
	10000	0.0056	0.0069	0.0427	0.0056	0.0061	**0.0294**
(Transformation der Eingabe, Netzwerke mit Biasneuron, Linearkombination mit Konstante)							

Tabelle 56: Ergebnisse zum ersten Datensatz und der Funktion AbsBZ

Skalierung	Neuronenanz.	mit Biasneuron			ohne Biasneuron		
z		dataset	trainingsset	testset	dataset	trainingsset	testset
0.125	0	3.4004	3.3793	3.5462	4.3612	4.3418	4.4823
	500	0.4304	0.4278	0.4561	0.4333	0.4307	0.4591
	1000	0.4321	0.4295	0.4587	0.4351	0.4325	0.4617
	2000	0.4368	0.4341	0.4631	0.1941	0.334	0.4143
	5000	0.4365	0.3836	0.4124	0.0945	0.1074	0.1478
	10000	0.4384	0.3483	0.4171	0.0335	0.0999	0.1545
0.25	0	3.4011	3.3799	3.5472	4.0512	4.0329	4.1539
	500	0.0743	0.0652	0.0995	0.0608	0.0582	0.0748
	1000	0.0624	0.0575	0.0823	0.033	0.0417	0.0661
	2000	0.0304	0.0363	0.0534	0.0317	0.0318	0.0529
	5000	**0.0283**	0.0292	0.0427	0.0297	0.0289	0.0437
	10000	0.0289	**0.0282**	**0.0391**	**0.0293**	**0.0288**	**0.041**
0.5	0	3.4103	3.389	3.56	3.6914	3.6735	3.7822
	500	0.0998	0.1037	0.1737	0.0957	0.1004	0.1679
	1000	0.1072	0.106	0.1633	0.091	0.0986	0.1575
	2000	0.0926	0.0987	0.1677	0.1001	0.1014	0.1617
	5000	0.0844	0.0899	0.1587	0.0842	0.0843	0.1626
	10000	0.0823	0.092	0.1587	0.083	0.097	0.1607
1	0	3.4944	3.4727	3.6751	3.566	3.5463	3.6995
	500	0.377	0.3779	0.5712	0.3798	0.3777	0.5865
	1000	0.3821	0.38	0.5482	0.3804	0.3797	0.5898
	2000	0.3842	0.3803	0.6084	0.3854	0.3855	0.6554
	5000	0.3822	0.385	0.6191	0.3795	0.3823	0.6337
	10000	0.3858	0.3867	0.612	0.3855	0.3847	0.6602
4	0	4.1528	4.1241	4.4825	4.1763	4.1486	4.4712
	500	2.2513	2.2215	184.0111	1.707	2.2029	208.9262
	1000	2.2514	2.2222	184.8067	1.9376	2.218	209.1044
	2000	1.9626	2.2149	215.182	2.2517	2.2224	209.255
	5000	2.2517	2.2181	215.8243	2.2517	2.2198	209.4324
	10000	2.2517	2.2222	215.6781	2.2518	2.2222	209.4385
10	0	4.2377	4.2094	4.5399	4.2377	4.2094	4.5399
	500	3.0988	3.0144	31.4404	2.9632	3.0459	31.4983
	1000	3.099	3.0479	31.702	2.9632	3.0349	31.6141
	2000	2.9632	3.0161	31.608	3.0994	3.0262	31.6406
	5000	2.9632	3.0315	31.726	2.9632	3.0473	31.7025
	10000	2.9632	3.0211	31.7498	2.9632	2.9968	31.6745
(Transformation der Eingabe, Netzwerke mit Biasneuron, Linearkombination mit Konstante)							

Tabelle 57: Ergebnisse zum ersten Datensatz und der Funktion SigmoidZ

Skalierung	Neuronenanz.	mit Biasneuron			ohne Biasneuron		
z		dataset	trainingsset	testset	dataset	trainingsset	testset
1	0	7.7863	7.7111	8.2747	12.401	12.3143	13.2281
	500	1.0956	1.227	33.9078	0.5048	0.5072	36.9503
	1000	0.4509	0.7123	39.7114	0.4268	0.4183	37.9763
	2000	0.3544	0.6248	40.9013	0.3891	0.4677	43.8691
	5000	0.294	0.4114	32.3662	0.1985	0.2151	30.4276
	10000	0.2293	0.3535	32.7656	0.1033	0.1481	34.7814
7	0	7.7863	7.7111	8.2747	12.401	12.3143	13.2281
	500	0.0013	$6.0E-4$	2.1989	$9.0E-4$	$4.0E-4$	3.6923
	1000	**0.0**	**0.0**	2.4803	**0.0**	**0.0**	3.6756
	2000	0.0	0.0	3.0453	0.0	0.0	5.0193
	5000	0.0	0.0	3.1652	0.0	0.0	5.4969
	10000	0.0	0.0	3.5934	0.0	0.0	6.2699
15	0	7.7863	7.7111	8.2747	12.401	12.3143	13.2281
	500	0.0	0.0	**2.1192**	0.0	0.0	**3.5539**
	1000	0.0	0.0	2.7516	0.0	0.0	4.2067
	2000	0.0	0.0	3.33	0.0	0.0	5.4496
	5000	0.0	0.0	3.6059	0.0	0.0	6.4935
	10000	0.0	0.0	4.1237	0.0	0.0	7.1405
50	0	7.7863	7.7111	8.2747	12.401	12.3143	13.2281
	500	0.0	0.0	3.4014	0.0	0.0	6.1895
	1000	0.0	0.0	3.4616	0.0	0.0	5.9189
	2000	0.0	0.0	4.8101	0.0	0.0	7.1474
	5000	0.0	0.0	4.6465	0.0	0.0	7.9602
	10000	0.0	0.0	5.3201	0.0	0.0	8.7375
150	0	7.7863	7.7111	8.2747	12.401	12.3143	13.2281
	500	0.0	0.0	2.9728	0.0	0.0	6.256
	1000	0.0	0.0	3.6029	0.0	0.0	7.1365
	2000	0.0	0.0	4.6999	0.0	0.0	8.8199
	5000	0.0	0.0	5.0931	0.0	0.0	9.4974
	10000	0.0	0.0	5.8213	0.0	0.0	10.2372
400	0	7.7863	7.7111	8.2747	12.401	12.3143	13.2281
	500	0.0	0.0	3.291	0.0	0.0	6.2832
	1000	0.0	0.0	4.0167	0.0	0.0	7.6811
	2000	0.0	0.0	5.2362	0.0	0.0	9.8644
	5000	0.0	0.0	5.9139	0.0	0.0	10.4051
	10000	0.0	0.0	6.5367	0.0	0.0	11.4288
(Transformation der Eingabe, Netzwerke mit Biasneuron, Linearkombination mit Konstante)							

Tabelle 58: Ergebnisse zum ersten Datensatz und der Funktion SinCosZ

Skalierung	Neuronenanz.	mit Biasneuron			ohne Biasneuron		
z		dataset	trainingsset	testset	dataset	trainingsset	testset
1	0	4.0414	4.0145	4.3835	6.8343	6.7963	7.1843
	500	0.2624	0.2554	3.8447	0.2531	0.2443	2.8815
	1000	0.2216	0.2186	3.6601	0.2063	0.2029	2.7056
	2000	0.2017	0.1829	4.0511	0.2104	0.2081	3.8852
	5000	0.1707	0.1527	3.7505	0.177	0.1677	3.4382
	10000	0.1634	0.1503	3.5482	0.1499	0.1361	3.929
7	0	3.4007	3.3796	3.5466	3.4094	3.3919	3.5021
	500	0.0038	0.0040	0.0192	0.0035	0.0039	0.0193
	1000	0.0040	0.0041	0.0199	0.0032	0.0035	0.0165
	2000	0.0042	0.0044	0.0192	0.0037	0.0040	0.0232
	5000	0.0031	0.0036	0.0165	**0.0030**	**0.0032**	0.0225
	10000	0.0029	0.0035	0.0177	0.0032	0.0036	0.0293
15	0	3.4004	3.3793	3.5461	3.4134	3.3961	3.5046
	500	0.0039	0.0043	0.0111	0.0041	0.0046	0.0146
	1000	0.0038	0.0042	0.0134	0.0030	0.0033	**0.0111**
	2000	0.0047	0.0055	0.017	0.0036	0.0041	0.0143
	5000	0.0031	0.0033	0.0103	0.0030	0.0035	0.013
	10000	0.0031	0.0035	0.0113	0.0034	0.0040	0.016
50	0	3.4004	3.3793	3.5461	3.4146	3.3973	3.5055
	500	0.0015	0.0019	0.0042	0.0219	0.0218	0.0467
	1000	0.0016	0.0021	0.0048	0.0197	0.0189	0.041
	2000	0.0020	0.0023	0.0045	0.0174	0.0194	0.0407
	5000	0.0019	0.0019	0.0036	0.0159	0.017	0.0353
	10000	0.0019	0.0021	0.0041	0.0176	0.0173	0.0361
150	0	3.4004	3.3793	3.5461	3.4147	3.3974	3.5055
	500	0.0010	0.0010	0.0015	0.0364	0.0382	0.0832
	1000	0.0012	0.0012	0.0018	0.0316	0.0363	0.0806
	2000	$9.0E-4$	0.0011	0.0019	0.0316	0.0336	0.0754
	5000	$8.0E-4$	$9.0E-4$	0.0016	0.0316	0.0302	0.0713
	10000	0.0010	0.0010	0.0016	0.0318	0.0305	0.0725
400	0	3.4004	3.3793	3.5461	3.4147	3.3974	3.5056
	500	$8.0E-4$	$8.0E-4$	0.0010	0.0457	0.0738	0.1302
	1000	$6.0E-4$	$7.0E-4$	0.0010	0.0465	0.051	0.0993
	2000	$\mathbf{2.0E-4}$	$\mathbf{4.0E-4}$	$\mathbf{7.0E-4}$	0.052	0.0498	0.0988
	5000	$3.0E-4$	$4.0E-4$	$7.0E-4$	0.05	0.0481	0.0936
	10000	$2.0E-4$	$6.0E-4$	$9.0E-4$	0.0503	0.0482	0.096
(Transformation der Eingabe, Netzwerke mit Biasneuron, Linearkombination mit Konstante)							

Tabelle 59: Ergebnisse zum ersten Datensatz und der Funktion SinZZ

Skalierung	Neuronenanz.	mit Biasneuron			ohne Biasneuron		
z		dataset	trainingsset	testset	dataset	trainingsset	testset
1	0	4.0414	4.0145	4.3835	6.8343	6.7963	7.1843
	500	0.2624	0.2554	3.8447	0.2531	0.2443	2.8815
	1000	0.2216	0.2186	3.6601	0.2063	0.2029	2.7056
	2000	0.2017	0.1829	4.0511	0.2104	0.2081	3.8852
	5000	0.1707	0.1527	3.7505	0.177	0.1677	3.4382
	10000	0.1634	0.1503	3.5482	0.1499	0.1361	3.929
7	0	4.0414	4.0145	4.3835	6.8343	6.7963	7.1843
	500	0.189	0.1822	98.0572	0.2975	0.2502	276.9743
	1000	0.1661	0.162	76.7673	0.1608	0.1536	140.3818
	2000	0.1655	0.1607	88.5751	0.1574	0.152	118.2477
	5000	0.1614	0.156	66.946	0.1562	0.1509	94.1072
	10000	0.1606	0.1542	58.3124	0.1566	0.1509	87.1799
15	0	4.0414	4.0145	4.3835	6.8343	6.7963	7.1843
	500	$2.0E-4$	$1.0E-4$	**0.7016**	**0.0**	**0.0**	**1.9222**
	1000	**0.0**	**0.0**	0.9982	0.0	0.0	2.4657
	2000	0.0	0.0	1.5669	0.0	0.0	3.3569
	5000	0.0	0.0	1.7569	0.0	0.0	4.3378
	10000	0.0	0.0	2.1851	0.0	0.0	4.7816
50	0	4.0414	4.0145	4.3835	6.8343	6.7963	7.1843
	500	0.0	0.0	5.0502	0.0	0.0	7.1452
	1000	0.0	0.0	5.4605	0.0	0.0	7.2341
	2000	0.0	0.0	6.8662	0.0	0.0	9.1441
	5000	0.0	0.0	6.6997	0.0	0.0	9.8712
	10000	0.0	0.0	7.2859	0.0	0.0	10.4238
150	0	4.0414	4.0145	4.3835	6.8343	6.7963	7.1843
	500	0.0	0.0	4.132	0.0	0.0	7.0746
	1000	0.0	0.0	5.4034	0.0	0.0	7.1079
	2000	0.0	0.0	6.7912	0.0	0.0	9.3625
	5000	0.0	0.0	6.9888	0.0	0.0	9.5515
	10000	0.0	0.0	7.4962	0.0	0.0	10.1667
400	0	4.0414	4.0145	4.3835	6.8343	6.7963	7.1843
	500	0.0	0.0	3.9382	0.0	0.0	4.558
	1000	0.0	0.0	4.0787	0.0	0.0	5.3573
	2000	0.0	0.0	5.473	0.0	0.0	7.609
	5000	0.0	0.0	5.4229	0.0	0.0	8.5041
	10000	0.0	0.0	6.2989	0.0	0.0	9.7488
(Transformation der Eingabe, Netzwerke mit Biasneuron, Linearkombination mit Konstante)							

Tabelle 60: Ergebnisse zum ersten Datensatz und der Funktion SinZ

Skalierung	Neuronenanz.	mit Biasneuron			ohne Biasneuron		
z		dataset	trainingsset	testset	dataset	trainingsset	testset
1	0	3.8241	3.7992	4.0972	3.85	3.8262	4.0889
	500	1.0023	0.9884	64.1373	0.9744	0.9857	219.4224
	1000	0.9896	0.9883	60.0452	1.0036	0.9901	149.2419
	2000	1.011	0.9884	78.6363	1.0256	1.0058	524.9676
	5000	1.0201	1.0027	61.7472	1.0141	0.9979	602.3451
	10000	1.0295	1.0117	92.2375	1.0033	0.9896	815.1495
7	0	3.4016	3.3804	3.5479	3.4069	3.3893	3.5014
	500	0.0098	0.0103	0.0519	0.0126	0.0126	0.0706
	1000	0.0101	0.0107	0.0529	0.0142	0.0144	0.0725
	2000	0.0128	0.0133	0.0802	0.0169	0.0181	0.124
	5000	0.0162	0.0165	0.0779	0.0212	0.0212	0.13
	10000	0.0187	0.0193	0.0975	0.0263	0.0263	0.1783
15	0	3.4004	3.3793	3.5462	3.4122	3.3949	3.5038
	500	0.0095	0.0098	0.0262	0.0086	0.0090	0.032
	1000	0.0079	0.0096	0.0319	0.0069	0.0071	0.0293
	2000	0.0094	0.0107	0.0365	0.0080	0.0093	0.0519
	5000	0.0063	0.0072	0.0286	0.0077	0.0093	0.0484
	10000	0.0064	0.0074	0.0398	0.0101	0.0112	0.0499
50	0	3.4004	3.3793	3.5461	3.4144	3.3972	3.5054
	500	0.0042	0.0047	0.0091	0.0079	0.0094	0.0247
	1000	0.0027	0.0031	0.0064	0.0054	0.0067	**0.0176**
	2000	0.0033	0.0043	0.0107	0.0065	0.0070	0.0199
	5000	0.0031	0.0035	0.0088	**0.0051**	**0.0066**	0.018
	10000	0.0032	0.0039	0.0108	0.0054	0.0066	0.019
150	0	3.4004	3.3793	3.5461	3.4147	3.3974	3.5055
	500	0.0020	0.0019	0.0033	0.0283	0.0281	0.065
	1000	0.0015	0.0015	0.0029	0.0235	0.0254	0.0591
	2000	0.0019	0.0019	0.0035	0.0255	0.0243	0.0515
	5000	0.0016	0.0019	0.0036	0.0232	0.0235	0.0451
	10000	0.0025	0.0025	0.0047	0.0238	0.0233	0.049
400	0	3.4004	3.3793	3.5461	3.4147	3.3974	3.5056
	500	$\mathbf{6.0E-4}$	$\mathbf{6.0E-4}$	$\mathbf{9.0E-4}$	0.0427	0.044	0.0936
	1000	$6.0E-4$	$6.0E-4$	$9.0E-4$	0.0417	0.0444	0.086
	2000	$7.0E-4$	$7.0E-4$	0.0011	0.0451	0.0424	0.0911
	5000	$7.0E-4$	$7.0E-4$	0.0010	0.0444	0.0416	0.0874
	10000	$9.0E-4$	$9.0E-4$	0.0013	0.0436	0.0411	0.0893
(Transformation der Eingabe, Netzwerke mit Biasneuron, Linearkombination mit Konstante)							

Tabelle 61: Ergebnisse zum ersten Datensatz und der Funktion TanhZZ

Skalierung	Neuronenanz.	mit Biasneuron			ohne Biasneuron		
z		dataset	trainingsset	testset	dataset	trainingsset	testset
1	0	3.8241	3.7992	**4.0972**	3.85	3.8262	**4.0889**
	500	1.0023	0.9884	64.1373	**0.9744**	**0.9857**	219.4224
	1000	**0.9896**	**0.9883**	60.0452	1.0036	0.9901	149.2419
	2000	1.011	0.9884	78.6363	1.0256	1.0058	524.9676
	5000	1.0201	1.0027	61.7472	1.0141	0.9979	602.3451
	10000	1.0295	1.0117	92.2375	1.0033	0.9896	815.1495
7	0	3.8241	3.7992	4.0972	3.85	3.8262	4.0889
	500	1.732	1.726	33999.7939	1.7541	1.7464	7970.1157
	1000	1.7351	1.7263	49968.8767	1.7536	1.7556	6610.995
	2000	1.7522	1.7462	8659.397	1.996	1.9266	7388.8623
	5000	1.7975	1.8316	8048.0757	2.0	1.9797	6343.0686
	10000	1.894	1.8943	7083.6299	2.1532	2.072	3075.3111
15	0	3.8241	3.7992	4.0972	3.85	3.8262	4.0889
	500	2.5043	2.4798	6.0868	2.4996	2.475	102.1213
	1000	2.5043	2.4798	6.0824	2.4996	2.475	5.8053
	2000	2.4996	2.475	87.3094	2.4971	2.4725	5.8708
	5000	2.4996	2.475	6.9482	2.4971	2.4725	25.1983
	10000	2.4996	2.475	6.2989	2.4971	2.4725	545.7759
50	0	3.8241	3.7992	4.0972	3.85	3.8262	4.0889
	500	2.5043	2.4798	266020.4625	2.5043	2.4798	811434.6355
	1000	2.5043	2.4798	6.7916	2.4997	2.4755	809267.5248
	2000	2.5043	2.4798	7.8044	2.4998	2.4756	712316.4024
	5000	2.4996	2.475	6.9225	2.5053	2.4801	611882.5629
	10000	2.4996	2.475	5.9603	2.5064	2.4809	251111.9154
150	0	3.8241	3.7992	4.0972	3.85	3.8262	4.0889
	500	2.51	2.4855	7.6007	2.51	2.4855	8.7256
	1000	2.5043	2.4798	6.8838	2.5053	2.4807	7.2664
	2000	2.5043	2.4798	7.907	2.5053	2.4807	7.009
	5000	2.4996	2.475	6.9958	2.5048	2.4801	6.0431
	10000	2.4996	2.475	6.0056	2.5048	2.4801	6.0256
400	0	3.8241	3.7992	4.0972	3.85	3.8262	4.0889
	500	2.5594	2.5342	7.8066	2.5594	2.5342	9.1431
	1000	2.5043	2.4798	6.9417	2.5548	2.5293	7.545
	2000	2.5043	2.4798	7.9067	2.5548	2.5293	7.2524
	5000	2.4996	2.475	7.0125	2.5542	2.5287	6.3746
	10000	2.4996	2.475	6.027	2.5542	2.5287	6.3247
(Transformation der Eingabe, Netzwerke mit Biasneuron, Linearkombination mit Konstante)							

Tabelle 62: Ergebnisse zum ersten Datensatz und der Funktion TanhZ

Skalierung	Neuronenanz.	SinZZ und Biasneuron			TanhZZ und Biasneuron		
z		dataset	trainingsset	testset	dataset	trainingsset	testset
0.01	10	3.2884	3.2595	3.6503	3.3184	3.2921	3.6113
	100	0.0020	0.0027	0.036	2.8233	2.7977	3.2327
	500	0.0028	0.0031	0.152	2.8233	2.789	3.491
0.1	10	3.3999	3.3633	3.8953	3.3184	3.2921	3.6113
	100	**0**	**0**	0.2459	2.8233	2.7977	3.2343
	500	0	0	3.4253	2.8233	2.789	3.5011
0.25	10	3.4004	3.3645	3.887	3.3183	3.2915	3.6205
	100	0	0.0012	0.4406	1.8715	1.8508	26381.6949
	500	$6.0E-4$	0	4.4832	1.7671	1.8041	88809.6418
0.5	10	3.3994	3.363	3.9266	3.3141	3.285	3.6909
	100	0.0257	0.0353	0.5982	1.1519	1.1521	212351.943
	500	0.0055	0.0099	11.2278	1.2177	1.269	10543.4942
0.75	10	3.3836	3.3489	3.8785	3.3075	3.2787	3.6725
	100	0	0	0.322	0.9774	0.9557	1143.916
	500	0	0	1.3703	0.8958	0.8836	247.3913
1	10	3.3049	3.2765	3.6697	3.3003	3.272	3.6506
	100	0	0	0.1597	0.6265	0.7698	11929.1398
	500	0	0	0.7856	0.6918	0.6993	95.887
7	10	3.2938	3.2643	3.6715	3.2911	3.2619	3.661
	100	0.0012	0.0015	0.0066	0.0049	0.0051	0.0233
	500	0.0033	0.0036	0.0188	0.0064	0.0066	0.0311
15	10	3.2942	3.2646	3.6729	3.2935	3.264	3.6702
	100	0.0011	0.0012	0.0031	0.0085	0.0085	0.0152
	500	0.0035	0.0041	0.0105	0.0088	0.0092	0.0249
50	10	3.2943	3.2647	3.6733	3.2942	3.2646	3.673
	100	0.0013	0.0013	0.0023	0.0017	0.0026	0.0043
	500	0.0015	0.0018	0.0041	0.0042	0.0047	0.0092
150	10	3.2943	3.2647	3.6732	3.2943	3.2647	3.6732
	100	0.0011	0.0011	0.0016	0.0023	0.0022	0.0033
	500	0.0010	$9.0E-4$	0.0015	0.0013	0.0019	0.0039
400	10	3.2943	3.2648	3.6713	3.2943	3.2647	3.6727
	100	0.0027	0.0039	0.0062	0.0027	0.0027	0.0039
	500	$8.0E-4$	$9.0E-4$	**0.0012**	$\mathbf{6.0E-4}$	$\mathbf{6.0E-4}$	$\mathbf{9.0E-4}$
(KEINE Transformation der Eingabe, Netzwerke mit Biasneuron, Linearkombination mit Konstante)							

Tabelle 63: Ergebnisse zum ersten Datensatz und den Funktionen SinZZ, TanhZZ

Skalierung	Neuronenanz.	mit Biasneuron			ohne Biasneuron		
z		dataset	trainingsset	testset	dataset	trainingsset	testset
1	0	2090731	2030757	2275801	2189975	2141677	2133968
	500	**0**	**0**	2491372	**0**	**0**	4115173
	1000	0	0	2377490	0	0	3675794
	2000	0	0	2328943	0	0	3360460
	5000	0	0	2344959	0	0	3621646
	10000	0	0	2226315	0	0	3551576
7	0	2090731	2030757	2275801	2189975	2141677	2133968
	500	0	0	1996912	0	0	1769978
	1000	0	0	1902338	0	0	1582995
	2000	0	0	1773924	0	0	1488570
	5000	0	0	1598960	0	0	1377496
	10000	0	0	1526227	0	0	1308678
15	0	2090731	2030757	2275801	2189975	2141677	2133968
	500	0	0	1528312	0	0	1413283
	1000	0	0	1455244	0	0	1343678
	2000	0	0	1398356	0	0	1254455
	5000	0	0	1257336	0	0	1215776
	10000	0	0	1211119	0	0	1201863
50	0	2090731	2030757	2275801	2189975	2141677	2133968
	500	0	0	1543346	0	0	1502523
	1000	0	0	1423843	0	0	1331706
	2000	0	0	1343975	0	0	1245842
	5000	0	0	1234345	0	0	1220090
	10000	0	0	**1192333**	0	0	1174962
150	0	2090731	2030757	2275801	2189975	2141677	2133968
	500	0	0	1632616	0	0	1369404
	1000	0	0	1419279	0	0	1332085
	2000	0	0	1274413	0	0	1254395
	5000	0	0	1239436	0	0	1218570
	10000	0	0	1195149	0	0	1164177
400	0	2090731	2030757	2275801	2189975	2141677	2133968
	500	0	0	1644293	0	0	1467585
	1000	0	0	1498342	0	0	1338692
	2000	0	0	1339776	0	0	1264637
	5000	0	0	1254303	0	0	1201562
	10000	0	0	1219461	0	0	**1146388**
(Transformation der Eingabe, Netzwerke mit Biasneuron, Linearkombination mit Konstante)							

Tabelle 64: Ergebnisse zum zweiten Datensatz und der Funktion SinZ

Skalierung	Neuronenanz.	mit Biasneuron			ohne Biasneuron		
z		dataset	trainingsset	testset	dataset	trainingsset	testset
1	0	2072139	2007818	2459039	2205332	2155412	2246644
	500	3.19	2.1231	1860273	17912	5304	$4.0564E8$
	1000	3.1766	1.9872	1866447	14333	4134	$3.7886E8$
	2000	3.1162	2.3182	1859660	12026	5128	$4.1095E8$
	5000	3.3579	2.4792	**1825005**	6999	3357	$3.7332E8$
	10000	3.2008	2.3698	1836080	2178	2580	$3.4478E8$
7	0	2106142	2041617	2442759	2217062	2167748.8691	2185591
	500	**0**	**0**	2515677	**0**	**0**	2474609
	1000	0	0	2422851	0	0	2357548
	2000	0	0	2286372	0	0	2328105
	5000	0	0	2318627	0	0	2282329
	10000	0	0	2291422	0	0	2228911
15	0	2157654	2096208	2439758	2250452	2203941	2157486
	500	0	0	2403294	0	0	2116584
	1000	0	0	2348056	0	0	2074237
	2000	0	0	2219337	0	0	2145112
	5000	0	0	2168722	0	0	2103270
	10000	0	0	2168648	0	0	2121522
50	0	2257085	2204812	2318911	2334707	2296098	**2001222**
	500	0	0	2312150	0	0	2332391
	1000	0	0	2358443	0	0	2395737
	2000	0	0	2204532	0	0	2380971
	5000	0	0	2279276	0	0	2541359
	10000	0	0	2347584	0	0	2501319
150	0	2242962	2183709	2338651	2329535	2285353	2041624
	500	$1.0E-4$	0	$1.2035E7$	0	0	8005800
	1000	0	0	$1.0542E7$	0	0	7696889
	2000	0	0	6574161	0	0	5341912
	5000	0	0	6556532	0	0	5246630
	10000	0	0	6372815	0	0	4270989
400	0	2208251	2146633	2254517	2289186	2235597	2022389
	500	36989	7610	$1.6843E8$	19899	2093	$1.1818E8$
	1000	40319	229.3748	$1.3103E8$	21682	496.4502	$9.5247E7$
	2000	31135	8.2789	$8.6376E7$	17783	1018	$5.3241E7$
	5000	117.8176	1.1052	$7.9713E7$	1157	197.1366	$4.5309E7$
	10000	423.5611	2.657	$6.7146E7$	798.8892	35.0164	$3.5812E7$
(Transformation der Eingabe, Netzwerke mit Biasneuron, Linearkombination mit Konstante)							

Tabelle 65: Ergebnisse zum zweiten Datensatz und der Funktion TanhZZ

Funktions-name	Skal. z	Neuronen-anzahl	dataset RMSE	trainingsset	testset	dataset Fitnesswert	trainingsset	testset
SinZ	400	2500	**0**	**0**	107.333	**0**	**0**	−14.7982
SinZZ			5.4558	−0.3418	5.4343	17.9538	−0.3414	−0.364
TanhZZ			5.4347	−0.3405	5.4174	18.0191	−0.3403	−0.3563
SinZ	400	5000	0	0	96.4665	0	0	−4.9897
SinZZ			5.4418	−0.3408	5.4195	18.3075	−0.3404	−0.3778
TanhZZ			5.4242	−0.3403	5.4048	18.1766	−0.3401	−0.381
SinZ	150	2500	0	0	131.5834	0	0	−28.168
SinZZ			5.4098	−0.3394	5.3985	18.0039	−0.3396	−0.3467
TanhZZ			5.3988	−0.3387	5.374	18.4741	−0.3382	−0.3689
SinZ	150	5000	0	0	102.1345	0	0	−10.6812
SinZZ			5.4038	−0.3388	5.3832	18.3039	−0.3387	−0.356
TanhZZ			5.3879	−0.3379	5.368	18.164	−0.3377	−0.3633
SinZ	50	2500	0	0	114.0234	0	0	−14.7546
SinZZ			5.3242	−0.3364	5.3172	18.185	−0.3362	−0.3576
TanhZZ			5.2968	−0.3351	5.2874	17.9738	−0.3346	−0.3586
SinZ	50	5000	0	0	97.6439	0	0	−3.6296
SinZZ			5.3504	−0.336	5.3266	18.7127	−0.3353	−0.3807
TanhZZ			5.2942	−0.3338	5.2783	17.7833	−0.3334	−0.3779
SinZ	15	2500	0	0	95.4889	0	0	−44.5873
SinZZ			5.1619	−0.3322	5.1536	17.446	−0.3318	−0.3565
TanhZZ			5.0705	−0.3271	5.0596	16.6049	−0.3263	−0.3763
SinZ	15	5000	0	0	80.418	0	0	−3.5057
SinZZ			5.1483	−0.3281	5.1375	17.4843	−0.3276	−0.3829
TanhZZ			5.0467	−0.324	5.0418	17.4691	−0.3235	−0.4214
SinZ	7	2500	0.5944	0.0261	108.8487	−0.0219	−0.0010	−23.1993
SinZZ			5.0509	−0.3275	5.0373	17.5498	−0.3267	−0.3656
TanhZZ			4.9243	−0.3208	4.9086	14.4246	−0.3193	−0.3861
SinZ	7	5000	0	0	66.4238	0	0	−2.8419
SinZZ			4.9827	−0.3219	4.9685	17.7191	−0.321	−0.3807
TanhZZ			4.8533	−0.3151	4.8281	10.9484	−0.3135	−0.4166
SinZ	1	2500	4.4456	4.3917	10.2338	−0.2932	−0.2894	−0.5499
SinZZ			4.4456	−0.2932	4.3917	10.2338	−0.2894	−0.5499
TanhZZ			3.9247	−0.2521	3.8349	12.6635	−0.2458	−0.7042
SinZ	1	5000	3.87	3.7867	15.0049	−0.2527	−0.2464	−0.8186
SinZZ			3.87	−0.2527	3.7867	15.0049	−0.2464	−0.8186
TanhZZ			3.5091	−0.2231	3.4006	16.0412	−0.2153	−0.9972
(Transformation der Eingabe, Netzwerke mit Biasneuron, Linearkombination mit Konstante)								

Tabelle 66: Ergebnisse zum dritten Datensatz und den Funktionen SinZ, SinZZ, TanhZZ

Hommage an Gödel

Münchhausens Theorem, Pferd, Sumpf und Schopf,
ist bezaubernd, aber vergiss nicht:
Münchhausen war ein Lügner.

Gödels Theorem wirkt auf den ersten Blick
Etwas unscheinbar, doch bedenk:
Gödel hat recht.

"In jedem genügend reichhaltigen System
lassen sich Sätze formulieren,
die innerhalb des Systems
weder beweis- noch widerlegbar sind,
es sei denn das System
wäre selber inkonsistent."

Du kannst deine eigene Sprache
in deiner eigenen Sprache beschreiben:
aber nicht ganz.
Du kannst dein eigenes Gehirn
mit deinem eigenen Gehirn erforschen:
aber nicht ganz
Usw.

Um sich zu rechtfertigen
muss jedes denkbare System
sich transzendieren,
d.h. zerstören.

"Genügend reichhaltig" oder nicht:
Widerspruchsfreiheit ist eine Mangelerscheinung
oder ein Widerspruch.

(Gewissheit = Inkonsistenz)

Jeder denkbare Reiter,
also auch Münchhausen,
also auch du bist ein Subsystem
eines genügend reichhaltigen Sumpfes.

Und ein Subsystem dieses Subsystems
Ist der eigene Schopf,
dieses Hebezeug
für Reformisten und Lügner.
In jedem genügend reichhaltigen System
also auch in diesem Sumpf hier,
lassen sich Sätze formulieren,
die innerhalb des Systems
weder beweis- noch widerlegbar sind.

Diese Sätze nimm in die Hand
und zieh!

(Text entstammt Enzensberger (2002))

Datenmedium

Kurze Inhaltsangabe des Datenmediums:

- Dokument „Parameteroptimierung bei der evolutionären Entwicklung neuronaler Topologien“ inklusive Quellen
- Entwicklerversion der Software Broccoli
- Datenstruktur Kanonisches (und Chaotisches) Netzwerk
- Datenstruktur LinearGenome
- Originaldaten und Hauptkomponentenanalyse des Benchmarks
- Testreihen und Quellcode zu NEAT, EANT, CMA-ES, TS, Stochastic Reflections und Stochastic Downhill Simplex u.a.

Literatur

[Balzert 1996] Balzert, Helmut: *Lehrbuch der Softwaretechnik, Teil 1: Softwareentwicklung.* Heidelberg, Germany : Spektrum Akademischer Verlag, 1996

[Balzert 1998] Balzert, Helmut: *Lehrbuch der Softwaretechnik, Teil 2: Softwaremanagement, Softwarequalitätssicherung, Unternehmensmodellierung.* Heidelberg, Berlin : Spektrum Akademischer Verlag, 1998

[Bishop 2004] Bishop, Christopher M.: *Neuronal Networks for Pattern Recognition.* Oxford University Press, 2004

[Borel 1913] Borel, Émile: Mécanique Statistique et Irréversibilité. In: *Journal de Physique* 3 (1913), S. 189–196

[Broyden 1970a] Broyden, C. G.: The convergence of a class of double-rank minimization algorithms. I: General considerations. In: *Journal of the Institute of Mathematics and its Applications* 6 (1970), S. 76–90

[Broyden 1970b] Broyden, C. G.: The convergence of a class of double-rank minimization algorithms, II: The new algorithm. In: *Journal of the Institute of Mathematics and its Applications* 6 (1970), S. 222–231

[Chaitin 1966] Chaitin, Gregory J.: On the Length of Programs for Computing Finite Binary Sequences. In: *Journal of the Association of Computing Machinery* 13 (1966), Nr. 4, S. 547–569

[Chaitin 1969] Chaitin, Gregory J.: On the Length of Programs for Computing Finite Binary Sequences: statistical considerations. In: *Journal of the Association of Computing Machinery* 16 (1969), Nr. 1, S. 145–159

[Chomsky 1956] Chomsky, Noam: Three models for the description of language. In: *IEEE: Transactions on Information Theory* 2 (1956), Nr. 3, S. 113–124

[Chomsky 1965] Chomsky, Noam: *Aspects of the Theory of Syntax.* Cambridge : MIT Press, 1965

[Chomsky u. Schützenberger 1963] Chomsky, Noam ; Schützenberger, Marcel P.: The Algebraic Theory of Context-Free Languages. In: Brafford, P. (Hrsg.) ; Hirshberg, D. (Hrsg.): *Computer Programming and Formal Systems.* North-Holland Publishing, 1963, S. 118–161

[Christey 2000] Christey, S.: The Infinite Monkey Protocol Suite (IMPS) / The Internet Society. United States : RFC Editor, 2000. – Forschungsbericht

[Church 1936] Church, Alonzo: An Unsolvable Problem of Elementary Number Theory. In: *American Journal of Mathematics* 58 (1936), Nr. 2, S. 345–363

[Duda u. a. 2004] Duda, Richard O. ; Hart, Peter E. ; Stork, David G.: *Pattern Classification.* Wiley Singapore, 2004

[Enzensberger 2002] Enzensberger, Hans M.: *Die Elixiere der Wissenschaft. Seitenblicke in Poesie und Prosa.* Suhrkamp Verlag, 2002

[Fahlman 1988] Fahlman, Scott E.: Faster-learning variations on back-propagation: An empirical study. In: Sejnowski, T.J. (Hrsg.) ; Hinton, G.E. (Hrsg.) ; Touretzky, D.S. (Hrsg.): *Connectionist Models Summer School.* San Mateo, CA : Morgan Kaufmann, 1988

[Fletcher 1970] Fletcher, R.: A New Approach to Variable Metric Algorithms. In: *Computer Journal* 13 (1970), Nr. 3, S. 317–322

[Flueckiger 1995] Flueckiger, Federico: *Contributions Towards a Unified Concept of Information*, University of Berne, Diss., 1995. `http://mypage.bluewin.ch/a-z/federico.flueckiger/Download/Btrg2uci.pdf`

[Gerstner u. Kistler 2002] Gerstner, Wulfram ; Kistler, Werner: *Spiking Neuron Models: An Introduction.* New York, NY, USA : Cambridge University Press, 2002 `http://icwww.epfl.ch/~gerstner/`

[Gilks 1995] Gilks, W. R.: *Markov Chain Monte Carlo in Practice.* Chapman & Hall/CRC, 1995

[Goldfarb 1970] Goldfarb, Donald: A Family of Variable-Metric Methods Derived by Variational Means. In: *Mathematics of Computation* 24 (1970), Nr. 109, S. 23–26

[Hansen u. Kern 2004] Hansen, Nikolaus ; Kern, Stefan: Evaluating the CMA Evolution Strategy on Multimodal Test Functions. In: *Parallel Problem Solving from Nature - PPSN VIII, 8th International Conference, Birmingham, UK, September 18-22, 2004, Proceedings (PPSN)*, 2004, S. 282–291

[Hansen u. Ostermeier 1996] Hansen, Nikolaus ; Ostermeier, Andreas: Adapting arbitrary normal mutation distributions in evolution strategies: the covariance matrix adaptation. In: *Proc. of the 1996 IEEE Int. Conf. on Evolutionary Computation.* Piscataway, NJ : IEEE Service Center, 1996, S. 312–317

[Hastings 1970] Hastings, W. K.: Monte Carlo Sampling Methods Using Markov Chains and Their Applications. In: *Biometrika* 57 (1970), Nr. 1, S. 97–109

[Hebb 1949] Hebb, Donald O.: *The Organization of Behavior: A Neuropsychological Theory.* New York : Wiley, 1949

[Hestenes u. Stiefel 1952] Hestenes, Magnus R. ; Stiefel, Eduard: Methods of Conjugate Gradients for Solving Linear Systems. In: *Journal of Research of the National Bureau of Standards* 49 (1952), S. 409–436

[Hornik 1990] Hornik, K.: Approximation capabilities of multilayer feedforward neural networks. In: *Neural Networks* 4 (1990), S. 251–257

[Hornik u. a. 1989] Hornik, K. ; Stinchcombe, M. ; White, H.: Multilayer feedforward networks are universal approximators. In: *Neural Networks* 2 (1989), Nr. 5, S. 359–366. – ISSN 0893–6080

[Igel u. a. 2006] Igel, Christian ; Suttorp, Thorsten ; Hansen, Nikolaus: A computational efficient covariance matrix update and a (1+1)-CMA for evolution strategies. In: *GECCO '06: Proceedings of the 8th annual conference on Genetic and evolutionary computation*. New York, NY, USA : ACM Press, 2006, S. 453–460

[Kassahun 2006] Kassahun, Yohannes: *Towards a Unified Approach to Learning and Adaptation*. Germany, Cognitive Systems Group, Institute of Computer Science, Christian-Albrechts-University of Kiel, Diss., 2006

[Kassahun u. Sommer 2005] Kassahun, Yohannes ; Sommer, Gerald: Efficient Reinforcement Learning Through Evolutionary Acquisition of Neural Topologies. In: *Proceedings of the 13th European Symposium on Artificial Neural Networks (ESANN 2005)*. Bruges, Belgium, 2005, S. 259–266

[Kolmogorov 1965] Kolmogorov, A. N.: Three approaches to the quantitative definition of information. In: *Problems of Information Transmission* 1 (1965), Nr. 1, S. 1–11

[McCulloch u. Pitts 1943] McCulloch, Warren S. ; Pitts, Walter: A logical calculus of the ideas immanent in nervous activity. In: *Bulletin of Mathematical Biology* 5 (1943), Nr. 4, S. 115–133

[Metropolis u. a. 1953] Metropolis, Nicholas ; Rosenbluth, Arianna W. ; Rosenbluth, Marshall N. ; Teller, Augusta H. ; Teller, Edward: Equation of state calculations by fast computing machines. In: *Journal of Chemical Physics* 21 (1953), Nr. 6, S. 1087–1092

[Nelder u. Mead 1965] Nelder, J. A. ; Mead, R.: A simplex method for function minimization. In: *Computer Journal* 7 (1965), Nr. 4, S. 308–313

[Prechelt 1995] Prechelt, Lutz: *Konstruktive neuronale Lernverfahren auf Parallelrechnern*, Universität Karlsruhe, Diss., 1995

[Rosenblatt 1958] Rosenblatt, F.: The perceptron: A probabilistic model for information storage and organization in the brain. In: *Psychological Review* 65 (1958), Nr. 6, S. 386–408

[Rosenblatt 1962] Rosenblatt, Frank: *Principles of Neurodynamics: Perceptron and the Theory of Brain Mechanisms*. Washington, D.C. : Spartan Books, 1962

[Rumelhart u. a. 1988] Rumelhart, David E. ; Hinton, Geoffrey E. ; Williams, Ronald J.: Learning representations by back-propagating errors. In: Anderson, James (Hrsg.) ; Rosenfeld, Edward (Hrsg.): *Neurocomputing: foundations of research*. Cambridge, MA, USA : MIT Press, 1988, S. 696–699

[Saad u. Schultz 1986] Saad, Youcef ; Schultz, Martin H.: GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. In: *SIAM Journal on Scientific and Statistical Computing* 7 (1986), Nr. 3, S. 856–869

[Shanno 1970] Shanno, D. F.: Conditioning of Quasi-Newton Methods for Function Minimization. In: *Mathematics of Computation* 24 (1970), Nr. 111, S. 647–656

[Shannon 1948] Shannon, C. E.: A Mathematical Theory of Communication. In: *Bell System Technical Journal* 27 (1948), S. 379–423 and 623–656

[Siebel u. Kassahun 2006] Siebel, Nils T. ; Kassahun, Yohannes: Learning Neural Networks for Visual Servoing using Evolutionary Methods. In: *Proceedings of the 6th International Conference on Hybrid Intelligent Systems (HIS'06), Auckland, New Zealand*, 2006, S. 6 (4 pages)

[Siebel u. a. 2007] Siebel, Nils T. ; Krause, Jochen ; Sommer, Gerald: Efficient Learning of Neural Networks with Evolutionary Algorithms. In: *DAGM-Symposium*, 2007, S. 466–475

[Solomonoff 1964a] Solomonoff, Ray J.: A Formal Theory of Inductive Inference. Part I. In: *Information and Control* 7 (1964), Nr. 1, S. 1–22

[Solomonoff 1964b] Solomonoff, Ray J.: A Formal Theory of Inductive Inference. Part II. In: *Information and Control* 7 (1964), Nr. 2, S. 224–254

[Stanley u. Miikkulainen 2002] Stanley, Kenneth O. ; Miikkulainen, Risto: Evolving Neural Networks Through Augmenting Topologies. In: *Evolutionary Computation* 10 (2002), Nr. 2, S. 99–127

[Turing 1936] Turing, Alan M.: On Computable Numbers, with an Application to the Entscheidungsproblem. In: *Proceedings of the London Mathematical Society* 2 (1936), S. 230–265

[Widrow u. Hoff 1960] Widrow, Bernard ; Hoff, Marcian E.: Adaptive Switching Circuits. In: *IRE WESCON Convention Record* 4 (1960), S. 96–104

[Zvonkin u. Levin 1970] Zvonkin, A. K. ; Levin, L. A.: The Complexity of Finite Objects and the Development of the Concepts of Information and Randomness by Means of the Theory of Algorithms. In: *Russian Mathematics Surveys* 256 (1970), S. 83–124

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich meine Diplomarbeit mit dem Titel „Parameteroptimierung bei der evolutionären Entwicklung neuronaler Topologien“ selbständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe, und dass ich alle Stellen, die ich wörtlich oder sinngemäß aus Veröffentlichungen entnahm, als solche kenntlich gemacht habe. Ferner lag die Arbeit bislang in gleicher oder ähnlicher Form oder auszugsweise noch keiner Prüfungsbehörde vor.

Kiel, 8. November 2007

_____

Lars Jochen Krause