Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Sven Lund

Optimierung von Neuronalen Netzen mit Hilfe Genetischer Algorithmen

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Sven Lund

Optimierung von Neuronalen Netzen mit Hilfe Genetischer Algorithmen

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang Angewandte Informatik
am Studiendepartment Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. Kai von Luck
Zweitgutachter : Prof. Dr. Andreas Meisel

Abgegeben am 16. März 2006

Sven Lund

Thema der Diplomarbeit

Optimierung von Neuronalen Netzen mit Hilfe Genetischer Algorithmen

Stichworte

Maschinelles Lernen, Neuronale Netze, Genetische Algorithmen

Kurzzusammenfassung

Die vorliegende Arbeit dokumentiert die Suche nach einem Klassifikator durch die Kopplung Genetischer Algorithmen mit Neuronalen Netzen. Aus der Familie der Neuronalen Netze werden Feedforward-Netze verwendet. Die Zusammenarbeit zwischen den Neuronalen Netzen und den Genetischen Algorithmen ist dergestalt, dass die Genetischen Algorithmen die Neuronalen Netze optimieren. Die Validierung dieses Vorgehens findet anhand von Bildmustern statt.

Sven Lund

Title of the paper

Optimization of neural networks via genetic algorithms

Keywords

Machine learning, neural networks, genetic algorithms

Abstract

The work under consideration documents the search for a classifier through linking genetic algorithms with neural networks. From the family of neural networks feed-forward networks are used. The neural networks and genetic algorithms collaborate in such a way that the genetic algorithms optimize the neural networks. This procedure is validated by means of patterns of pictures.

Danksagung

Kai von Luck und Andeas Meisel für die gute Betreuung und vor allem für die Vermittlung der Begeisterung am Thema, ohne die es diese Arbeit nicht gegeben hätte.

Carolin Thoma für intensive und produktive Textkritik.

Anna Hewener für die initiale Zündung, ohne die diese Arbeit noch nicht fertig wäre, sowie ihre Unterstützung in orthografischen Fragen.

Dominik Hübner für die Unterstützung bei der Arbeit mit Photoshop.

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Einführung

1.1. Motivation

In dem Moment, in dem wir die Augen öffnen, begegnen wir einer Welt voller Bilder. Ob es ein Monitor ist, das Gesicht eines Freundes oder eine gelbe Ampel - für gewöhnlich bleibt es nicht beim bloßen Sehen. Wir interpretieren und kategorisieren. Wir stehen an einem Ort umgeben von Bäumen und gefragt, wo wir uns aufhalten, werden wir ohne nachzudenken sagen: „In einem Wald“. Das ist für Menschen sehr leicht. Andererseits neigen wir dazu, einfachste geometrische Formen „falsch“ zu sehen. So werden die meisten Betrachter in Abbildung 1.1 den unteren Balken für länger halten als den oberen, obwohl beide gleich lang sind. Einem Computer mit einer Kamera bereitet ein solcher Vergleich keine Schwierigkeiten. Dafür bekommt dieser große Probleme, wenn er einfachste Straßenszenen aus unterschiedlichen Blickwinkeln richtig zuordnen soll[1]. Das Erkennen solcher Bilder oder Szenen nennt man Bildmustererkennung[2].

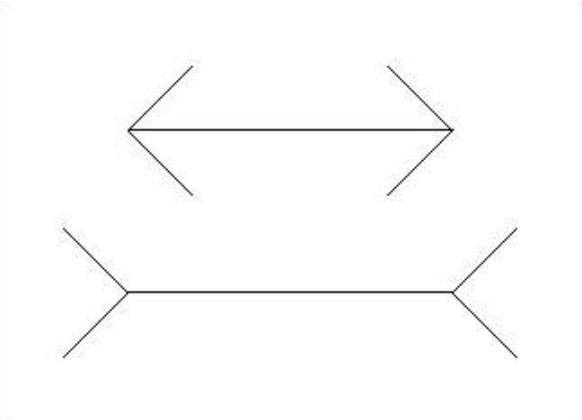

Abbildung 1.1.: Müller-Lyer-Illusion

[1] Siehe hierzu Cruse u. a. (2001) Seite 80f

[2]

1.2. Mustererkennung

Mustererkennung ist ein Prozess, der bei sämtlichen Wirbeltieren laufend völlig automatisch von statten geht. Für die verschiedensten Technologien wie z.B. Biometrie und Diktatsoftware wird Mustererkennung ebenfalls oft gebraucht.

Um für Probleme bei der Mustererkennung zu sensibilisieren, soll hier ein kurzes Beispiel genannt werden. Die vorgestellte Szene soll ein blaues Spielfeld - für Roboterfußball - mit einem gelben Ball darauf enthalten. Ziel ist es, mit einem Roboter den Ball zu entdecken.

Die reine Suche nach gelben Bildpunkten in der Kameraaufnahme des Roboters wird nicht ausreichen. Die Projektion eines blauen Spielfeldes wird vermutlich ebenso wie der Ball einige gelbe Bildpunkte enthalten.

An dieser Stelle könnte man dazu übergehen „gelb“ neu zu definieren. Das kann z.B. bedeuten, dass nur noch ein bestimmter Teilbereich des Farbbereiches, den Menschen als gelb wahrnehmen, als gelb akzeptiert wird. Eine andere Möglichkeit besteht darin, nur eine Mindestanzahl zusammenhängender gelber Bildpunkte als Ball zu akzeptieren. Das Problem ist also lösbar.

Erleichtert wird eine solche Aufgabe auch dadurch, dass auf einem entsprechenden Spielfeld keine anderen Gegenstände in der Farbe des Balles erlaubt werden.

Schwieriger wird es, wenn man sich der natürlichen Umgebung des Menschen zuwendet. Hier finden wir oft Strukturen vor, die sich nicht trivial beschreiben und unterscheiden lassen. Dort setzt die vorliegende Arbeit an.

1.3. Thema

Das zentrale Thema dieser Arbeit ist nicht die Mustererkennung als Ganzes. Der Autor konzentriert sich auf einen Teilaspekt der Mustererkennung, nämlich die automatische Klassifikation.

In dieser Arbeit werden exemplarisch zwei so unterschiedliche Objekttypen wie handschriftliche Zahlen und Blätter verschiedener Baumarten untersucht.

Das Erkennen handschriftlicher Zahlen ist bereits erschöpfend unter dem Thema OCR[3] bearbeitet worden. Dieser Objekttyp soll hier als Vergleichsmaßstab herangezogen werden.

[3] Optical Character Recognition

Abbildung 1.2.: Ahorn 1

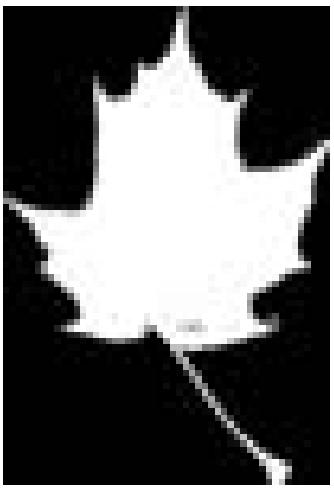

Abbildung 1.3.: Ahorn 2

Die Klassifizierung von Blättern hingegen ist bislang kaum untersucht. Sie bietet insofern eine Herausforderung, als dass selbst Menschen bisweilen Schwierigkeiten haben, verschiedene Blattsorten zu unterscheiden. Als Beispiel werden vier Scherenschnitte von zwei Blattarten präsentiert. Die Blätter auf den Abbildungen 1.2 und 1.3 sind Ahornblätter; die Abbildungen 1.4 und 1.5 zeigen jeweils Blätter der Platane. Der äußere Anschein suggeriert aber eher, dass jeweils die Blätter auf den Abbildungen 1.2 und 1.5 und jene, welche die Abbildungen 1.3 und 1.4 zeigen, der selben Art angehören.[4]

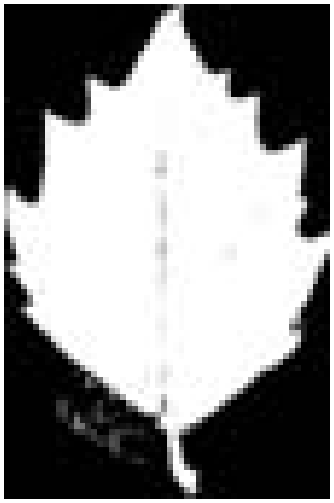

Abbildung 1.4.: Platane 1

Abbildung 1.5.: Platane 2

[4] Die Scherenschnittdarstellung wird hier nur zur Verdeutlichung gewählt.

Offensichtlich liefert die Vermessung von Größe und Proportion eines Blattes kein Alleinstellungsmerkmal zur Unterscheidung der Blattarten. Auch Farben und Kontraste sind keine spezifischen Eigenschaften, da sie im Lauf der Jahreszeiten stärker variieren als über Artgrenzen hinweg.

Zudem wird die Kategorisierung von Blattoberflächen durch natürliche Störungen wie Insektenfraß oder die Hinterlassenschaften von Vögeln erschwert. Menschen sind meist in der Lage, von solchen Störungen zu abstrahieren. Hingegen ist die mathematische Beschreibung zum Herausfiltern dieser Störungen alles andere als trivial.

Es stehen zwei Wege offen, um eine Klassifikation zu automatisieren. Zum einen wäre es denkbar, der Fachliteratur Alleinstellungsmerkmale der einzelnen Blattarten zu entnehmen und in eine mathematische Beschreibung dieser Arten zu überführen. Zum anderen bietet sich ein Verfahren an, das diese Merkmale selbstständig erkennt. In dieser Arbeit wird der zweite Weg verfolgt.

Das Klassifikationsverfahren, das im Rahmen dieser Arbeit verwendet werden soll, gehört zur Familie der Künstlichen Neuronalen Netze. Bei der Nutzung Neuronaler Netze gibt es eine immense Zahl an Parametern. Die optimale Einstellung dieser Parameter ist mehr eine „Kunst als eine Wissenschaft“. Da es kein mathematisch fundiertes Verfahren zur Festlegung optimaler Netzparameter gibt, ist man entweder auf allgemeine *Heuristiken* angewiesen oder bedient sich der Erfahrung eines Experten.

Eine weitere Möglichkeit Neuronale Netze zu konfigurieren stellen automatisierte Optimierungsverfahren dar. Diese Arbeit wird sich explizit der Optimierung der Neuronalen Netze mit Hilfe der Genetischen Algorithmen widmen. Dass eine solche Optimierung grundsätzlich funktioniert und brauchbare Ergebnisse hervorbringt, ist bekannt und erforscht[5].

Ziel vorliegender Arbeit ist es, ein Verfahren zur Klassifikation zu finden, das auch ohne vertiefte Kenntnisse über Genetische Algorithmen und Neuronale Netze Klassifikatoren liefert. Welche Rahmenbedingungen zu guten Ergebnissen führen, soll in verschiedenen Experimenten untersucht werden [6].

1.4. Ausblick

In Kapitel 2 „Grundlagen“ werden die in der Arbeit verwendeten Verfahren eingeordnet und beschrieben (Kapitel 2.1 und 2.2). Am Ende des Kapitels folgen noch einige weiterführende Gedanken (Kapitel 2.3).

[5] Siehe hierzu Schaffer u. a. (1992).
[6] Siehe Kapitel 5 ab Seite 72

Das Kapitel 3 „Aufbau des Optimierungssystems“ erläutert den verwendeten Genetischen Algorithmus (Kapitel 3.1). Weiterhin befindet sich hier die Beschreibung der im Rahmen dieser Arbeit erstellten Software (Kapitel 3.2), die mit Hilfe eines Simulators zur Berechnung Neuronaler Netze, den beschriebenen Algorithmus umsetzen wird.

Im darauf folgenden Kapitel 4 „Versuchsaufbau“ werden die Versuchsbedingungen (Kapitel 4.1) und das verwendete Programm zur Berechnung Neuronaler Netze (Kapitel 4.2) vorgestellt. Aspekte der Benutzung der hier erstellten Software (Kapitel 4.3) werden ebenfalls erwähnt. Am Ende dieses Kapitels wird noch auf den Umgang mit aufgetretenen Fehlern (Kapitel 4.4) eingegangen.

Kapitel 5 „Versuchsdurchführung und Ergebnisse“ beschreibt die im Rahmen der Arbeit durchgeführten Versuche sowie deren Ergebnisse.

Den Schluss bildet Kapitel 6 „Zusammenfassung und Ausblick“. Dies zieht Resümee (Kapitel 6.1) über das in dieser Arbeit Erreichte, wagt aber auch einen Ausblick über das Gegebene hinaus (Kapitel 6.2).

2. Grundlagen

2.1. Maschinelles Lernen und Wissensrepräsentation

Verfahren, die in der Lage sind, Wissen „selbstständig“ zu generieren subsumiert man unter dem Begriff *Maschinelles Lernen*. Derartige Verfahren lernen durch Beispiele und sind nach Abschluss der Lernphase[1] in der Lage das Gelernte zu verallgemeinern.

Unter Wissensrepräsentation versteht man die Art und Weise, in der das Wissen in einem System vorgehalten wird.

Im Folgenden werden drei Bereiche des Maschinellen Lernens mit der jeweils dazugehörigen Form der Wissensrepräsentation vorgestellt[2].

Symbolbasiert

Das Lernen wird als Akquisition von explizit beschriebenem Domänenwissen modelliert. Auf Basis seiner Erfahrung konstruiert und modifiziert das lernende System Ausdrücke in einer formalen Sprache. Diese Ausdrücke werden der Wissensbasis hinzugefügt. Das Systemverhalten wird primär durch das explizit beschriebene Domänenwissen beeinflusst.

Konnektionistisch

Die Basis bilden einfache, miteinander verbundene Einheiten. Das Systemwissen steckt in der Struktur und der Interaktion der Verbindungen zwischen diesen Einheiten. Die Erweiterung des Systemwissens geschieht durch Veränderungen an den Verbindungen.

[1] Manche Verfahren weichen davon ab. Sie lernen auch später noch hinzu.
[2] Die hier folgende Einordnung ist stark an Luger (2001) angelehnt.

Sozial und emergent

Eine Gruppe von Lösungsmöglichkeiten konkurrieren miteinander. Die besten Lösungsmöglichkeiten werden ausgewählt. Auf deren Basis wird eine neue Gruppe generiert. Aus diesem Prozess wird ein Zyklus aufgebaut, der an die biologische Evolution erinnert. Das Wissen wird von jeder Lösungsmöglichkeit repräsentiert und kann auf unterschiedliche Art abgelegt sein.

2.2. Grundlagen der gewählten Verfahren

Das Verfahren, das im Rahmen dieser Arbeit zur Klassifikation verwendet wird, sind die sog. Künstlichen Neuronalen Netze. Sie gehören zu den konnektionistischen Lernverfahren. Künstliche Neuronale Netze werden schon seit Jahren zur Klassifikation benutzt. Die ihnen anhaftenden Nachteile sind deshalb hinlänglich bekannt. Für die vorliegende Arbeit ist vor allem der Nachteil bzw. die Schwierigkeit, das Netz richtig zu konfigurieren, relevant. Zur Lösung dieses Problems werden die Künstlichen Neuronalen Netze mit einem anderen Verfahren gekoppelt. Das andere Verfahren ist ein sog. Genetischer Algorithmus. Genetische Algorithmen gehören zu den sozialen und emergenten Lernverfahren.

Neuronale Netze

Allgemeines

Neuronale Netze lernen an Hand von Beispielen. Sie werden mit möglichst guten Repräsentanten einer Kategorie gespeist und erkennen im Idealfall das „allgemeine Prinzip“ dieser Kategorie. Wenn dies gelingt, können Neuronale Netze auch solche Repräsentanten der Kategorie erkennen, die nicht zu den Beispielen gehören.

Neuronale Netze wurden stark durch den Aufbau des menschlichen Gehirns inspiriert, sind allerdings grob vereinfacht. Richtigerweise muss man in Abgrenzung zum natürlichen Vorbild von Künstlichen Neuronalen Netzen sprechen. In der Literatur wird oftmals der Begriff *Neuronale Netze* ohne das Adjektiv *künstlich* verwendet, insbesondere dann, wenn aus dem Kontext hervorgeht, um welchen Untersuchungsgegenstand es sich handelt. Im Text dieser Arbeit wird das Adjektiv *künstlich* ebenfalls weggelassen, da auch hier der Kontext eindeutig ist.

Varianten

Es gibt verschiedene Lernverfahren zum Klassifizieren durch Neuronale Netze:

- Überwachtes Lernen: Die gewünschten Ausgabedaten des Neuronalen Netzes sind bekannt und werden beim Lernprozess verwendet.
- Bestärkendes Lernen: Die gewünschten Ausgabedaten sind nicht bekannt. Dem Neuronalen Netz werden lediglich dichotome Informationen - wie z.B. gut und schlecht - zum Netzergebnis gegeben.
- Unüberwachtes Lernen: Den Neuronalen Netzen wird beim Lernen keine Information zu erwünschten Ausgabedaten übergeben.

Diese Arbeit wird sich nur mit dem überwachten Lernen beschäftigen. Überwachtes Lernen bedeutet, dass es zu jedem Dateneingangsvektor, wie zum Beispiel der *8* in Abbildung 2.6 einen vorher festgelegten Datenausgangsvektor gibt, der die gewünschte Ausgabe darstellt. Aus dieser gewünschten Ausgabe und der tatsächlichen Ausgabe des Neuronalen Netzes wird ein Differenzvektor gebildet. Dieser wird von hinten nach vorn pro Layer zur Korrektur der Einzelgewichte herangezogen.

Ein weiteres Spezifikum für die hier verwendeten Neuronalen Netze ist, dass es Feedforward-Netze sind, die nach der Error-Backpropagation-Lernregel lernen[3]. Wenn im Folgenden also von Neuronalen Netzen die Rede ist, sind immer diese gemeint.

Überwachtes Lernen als Funktionsapproximation Eine alternative Sichtweise auf ein Neuronales Netz sieht dieses als eine mathematische Funktion. Das Training verändert die Ergebnisse dieser Funktion. Es soll Ergebnisse erzeugen, die einen Dateneingang einer gewünschten Kategorie möglichst genau zuordnen. In der Mathematik nennt man einen solchen Vorgang Funktionsapproximation. Die dazugehörigen Stützstellen liefert beim überwachten Lernen der zum Dateneingang mitgegebene Datenausgang.

[3] Mehr dazu in Kapitel 3.1 auf Seite 40

Topologie

Neuronale Netze bestehen aus einer Anzahl von einfachen Einheiten, die ein Problem virtuell parallel [4] bearbeiten. Die einfachen Einheiten nennt man Unit, Neuron oder Knoten. In dieser Arbeit wird der Begriff Neuron verwendet. Die einzelnen Neuronen sind in Schichten, sogenannten Layern, angeordnet, wie auf Abbildung 2.1 zu erkennen ist[5].

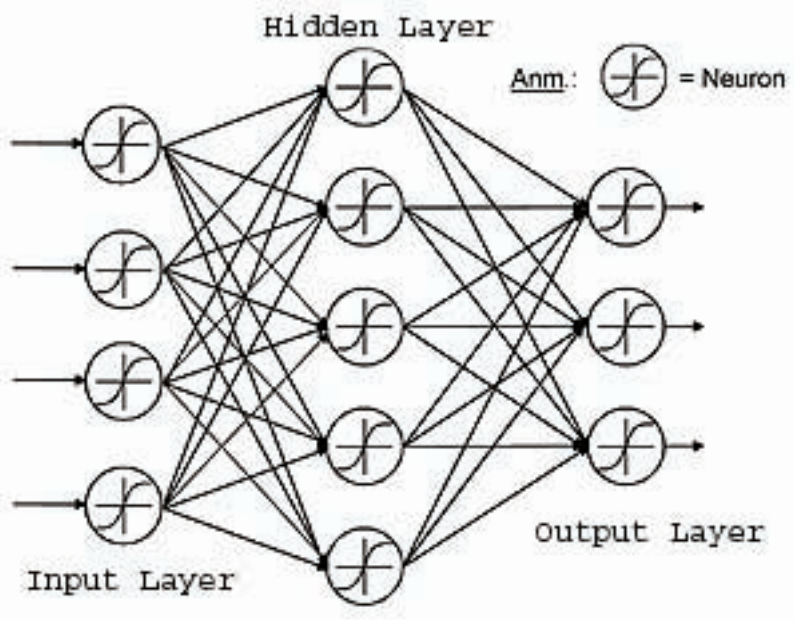

Abbildung 2.1.: Schema Neuronales Netz (Quelle: [Meisel:2004] / modifiziert)

Die Input Neuronen sind der Dateneingang des Neuronalen Netzes. Hier werden die zu klassifizierenden Daten als Zahlenvektoren eingespeist. Die Hidden Neuronen oder besser die Verbindungen zu, zwischen und von den Hidden Neuronen repräsentieren das „Gedächtnis“ des Neuronalen Netzes. Dieses liegt in Form von Gewichtungswerten der einzelnen Verbindungen vor[6]. An den Output Neuronen liegt das Ergebnis der von dem Gesamtnetz vorgenommenen Klassifikation ebenfalls als Zahlenvektor vor. Während des Lernens wird dieser Ausgangsvektor vorgegeben. Nach dem Lernen erzeugt das Neuronale Netz diesen Vektor aus dem Zusammenwirken von Eingangsdaten und Netzgewichten selbst.

[4] Neuronale Netze arbeiten in den meisten Fällen nicht wirklich parallel, da bei der Simulation von Neuronalen Netzen z.B. auf einem Einprozessorcomputer nur sequenziell gearbeitet werden kann

[5] Siehe hierzu auch die Abbildungen 3.4 und 3.5

[6] In Abbildung 2.1 der Übersichtlichkeit wegen nur als Pfeile eingezeichnet

Propagierungsfunktion

Jedes Neuron[7] j bekommt von den Neuronen der Vorgängerschicht jeweils einen Zahlenwert o_i $(i = 1..n)$ übergeben. Dieser Zahlenwert wird mit dem „Gewicht“ w_{ij} $(i = 1..n)$ der Verbindung zwischen aktuellem und dem jeweiligen Vorgängerneuron verrechnet[8]. Dies ist die Eingabeinformation des Neurons. Zu jedem Neuron gehört zudem ein sogenanntes Bias Θ_j, welches einen Schwellwert darstellt. Dieser ist negativ und wird zur Summe aller Eingabeinformationen $\sum o_i w_{ij}$ addiert. Daraus ergibt sich die *Propagierungsfunktion* net_j.

$$net_j = \sum o_i w_{ij} + \Theta_j$$

Das Ergebnis der Propagierungsfunktion liefert den Eingabewert für die anschließende Berechnung des Neuronenausgabewertes o_j durch die *Aktivierungsfunktion* $f(net_j)$.

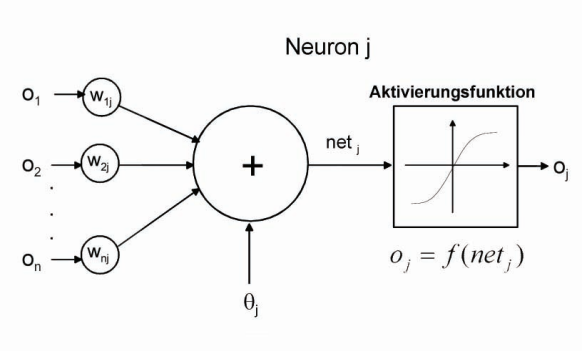

Abbildung 2.2.: Einzelnes Neuron (Quelle: [Meisel:2004])

Aktivierungsfunktion

Es gibt eine ganze Reihe von Aktivierungsfunktionen. Einige wichtige sind auf den Abbildungen 2.3, 2.4 und 2.5 zu sehen. Die Abszisse der jeweiligen Graphen steht hierbei für die Ergebnisse der Propagierungsfunktion. Die Ordinate steht für die Ergebnisse der Aktivierungsfunktion, die gleichzeitig die Ausgabe des Neurons sind.

[7] Eine Ausnahme stellen hier die Input Neuronen dar
[8] Klassischerweise werden o_x und w_{xj} miteinander multipliziert

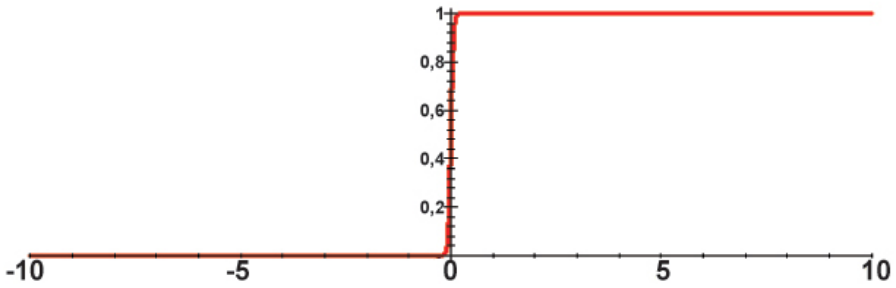

Abbildung 2.3.: Schrittfunktion (Quelle: [Meisel:2004] / modifiziert)

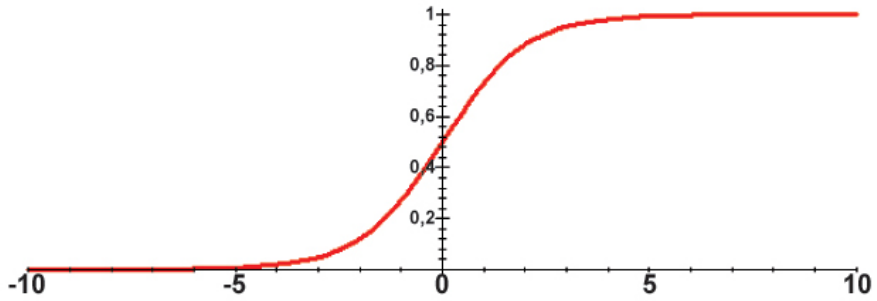

Abbildung 2.4.: Logistische Funktion (Quelle: [Meisel:2004] / modifiziert)

Grundidee

Die Grundidee der Verwendung von Neuronalen Netzen bei Problemlösungen ist, dass ein entsprechendes Netz die Lösung des Problems selbstständig approximiert[9]. Diese Annäherung nennt man Lernen und analog zu unserer alltäglichen Begriffswelt spricht man vom Trainieren des Netzes. Bei der Backpropagation-Familie findet das Training mittels eines speziellen mathematischen Verfahrens statt. Dieses Verfahren stellt die Verbindungsgewichtungen von hinten nach vorne so ein, dass die Antwort des Netzes auf die Eingangsdaten im Input Layer im Output Layer dem gewünschten Output angenähert wird.

[9] Bei logikbasierten Ansätzen muss bereits bei der Entwicklung eines entsprechenden Problemlösers eine Idee vom Lösungsweg vorhanden sein.

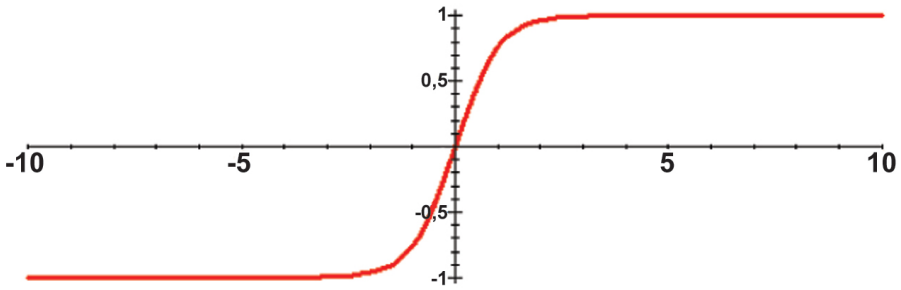

Abbildung 2.5.: Tangens-Hyperbolicus-Funktion (Quelle: [Meisel:2004] / modifiziert)

Nachteile

- Neuronale Netze arbeiten nach dem Prinzip der Fehlerminimierung durch Gradientenabstieg[10]. Daraus folgt immer die Gefahr, in ein lokales Minimum[11] im Fehlergebirge zu laufen, wie auf Abbildung 2.7 zu sehen ist. Das lokale Minimum ist deshalb problematisch, weil bei klassischem Gradientenabstieg kein Weg mehr dort heraus führt.
- Es ist kaum vorhersagbar, welche Belegungen der Netz-Parameter zu guten oder sehr guten Ergebnissen beim Lernen führen. Dies ist deshalb ein Problem, weil es so viele mögliche Belegungen gibt und der darüber aufgespannte Suchraum folglich sehr groß ist.
- Ein sehr gutes Ergebnis bezüglich des Fehlerquadrats bedeutet nicht notwendig, dass ein Netz ein Problem gut generalisiert [12].
- Es kann einen beträchtlichen Aufwand bedeuten, repräsentative Trainingsdaten zu finden.

[10] Gradientenabstieg bedeutet, dass die Richtungsbestimmung anhand des steilsten Gefälles innerhalb des Lösungsraumgebirges gefällt wird. Das Problem wird deutlich, wenn man sich den Abstieg von einem Berg vorstellt, bei dem man stets den steilsten Weg nimmt, um möglichst schnell nach unten zu kommen. Führt der Weg in eine Mulde am Hang, kommen wir, wenn wir streng nach dem Prinzip des Gradientenabstiegs vorgehen, nicht mehr aus besagter Mulde heraus.

[11] Lokales *Minimum*, weil für die hier gemachten Versuche ein niedriges Ergebnis erwünscht ist

[12] Das Phänomen, dass ein Neuronales Netz trotz niedrigen Fehlers im Training schlecht generalisiert nennt man *Overfitting* oder auch *Overlearning*

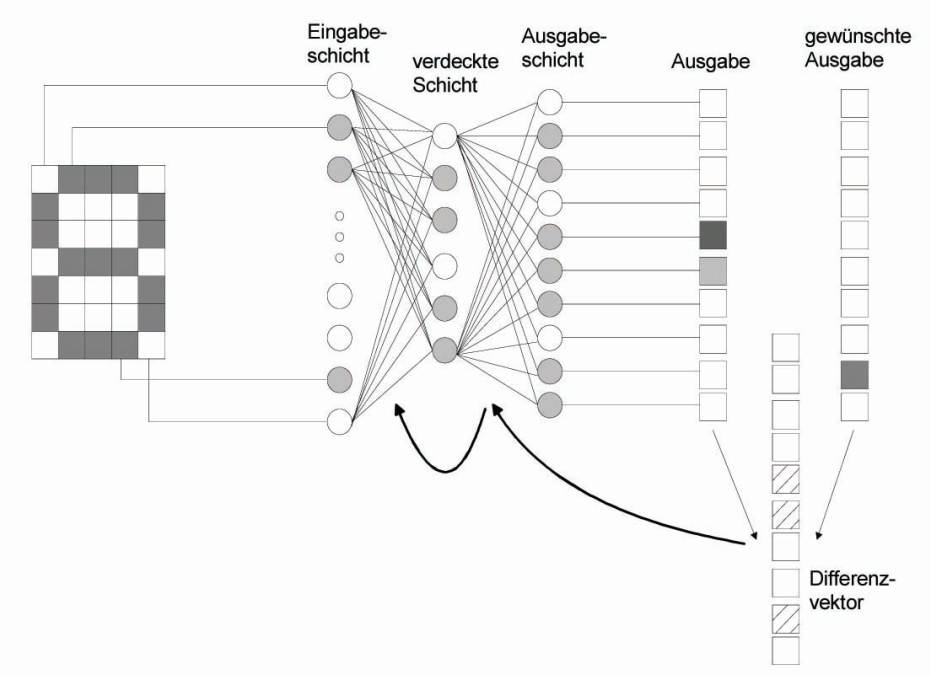

Abbildung 2.6.: Überwachtes Lernen mit Backpropagation (Quelle: [Meisel:2004])

Genetische Algorithmen

Einordnung

Genetische Algorithmen gehen auf John H. Holland [13] zurück. Sie gehören zu den evolutionären Algorithmen, die wiederum der Klasse der heuristischen Suchverfahren zugeordnet sind. Ihre Anwendung ist insbesondere beim Durchsuchen großer Suchräume sinnvoll.

Prinzip

Genetische Algorithmen sind ein Optimierungsverfahren, das stark von dem Vorbild der biologischen Evolution inspiriert wurde. Melanie Mitchell schreibt dazu:

> Biological evolution is an appealing source of inspiration for addressing these problems. Evolution is, in effect, a method of searching among an enormous number of possibilities for „solutions“.[14]

[13] Holland (1992)
[14] Mitchell (1996)

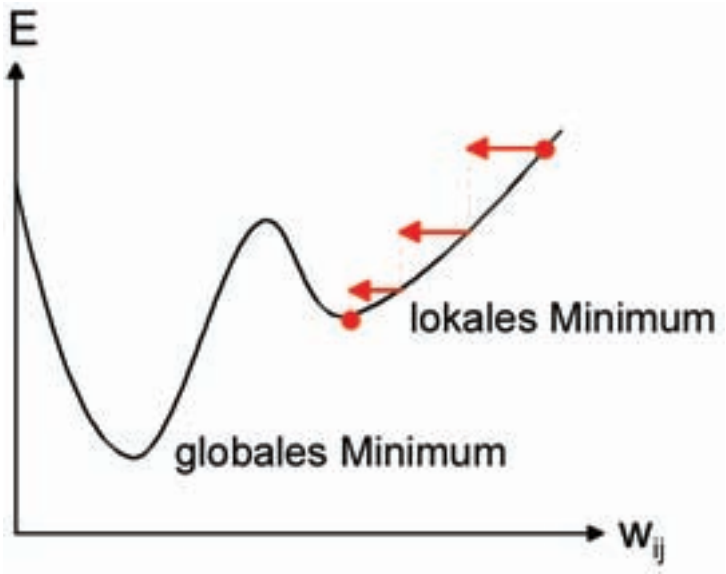

Abbildung 2.7.: Lokales Minimum (Quelle: [Meisel:2004])

Bei den Genetischen Algorithmen wird zu Beginn eine Gruppe von Individuen erstellt, welche die Anfangspopulation bildet. Diese Individuen treten gegeneinander an. Die besten Individuen werden selektiert und zur Bildung einer neuen Population herangezogen. Die Erzeugung der neuen Generation geschieht mit Hilfe von Mutation und Rekombination (Cross-Over). Diese neue Population tritt nun wieder gegeneinander an und so fort. Das Prinzip wiederholt sich, bis ein gegebenes Abbruchkriterium erfüllt wird.

Individuum Ein Individuum kann je nach Sichtweise als *Genotyp* oder als *Phänotyp* betrachtet werden. Der Genotyp liefert den „Bauplan“, während der Phänotyp das tatsächliche Subjekt/Objekt darstellt. Im Fall der Neuronalen Netze enthält der Genotyp beispielsweise die Anzahl der Neuronen in jedem Layer. Hierzu gibt es eine unendliche Zahl an passenden Phänotypen, in denen dann die Neuronen konkret existieren und mit beliebigen *Gewichten* untereinander verbunden sein können.

Population Eine Population ist eine Gruppe von Individuen. Dies kann sowohl alle Phänotypen als auch alle Genotypen meinen, nicht aber beide gleichzeitig.

Generation Unter Generation versteht man eine Population zu einem Zeitpunkt des Genetischen Algorithmus.

Selektion Bei der Selektion handelt es sich um einen Entscheidungsmechanismus, welche Individuen an der Erzeugung der Folgegeneration mitwirken dürfen. Bei der Ermittlung dieser Individuen spielt deren Fitness eine große Rolle. Diese wird über die so genannte Fitnessfunktion ermittelt. Es ist nicht notwendigerweise so, dass immer die fittesten Individuen zur Reproduktion herangezogen werden. In allen Verfahren erhalten aber Individuen mit einer höheren Fitness eine größere Chance zur Reproduktion als solche mit einer niedrigeren. Die konkrete Auswahl hängt vom Ersetzungsschema ab.

Ersetzungsschema Das Ersetzungsschema eines Genetischen Algorithmus gibt an, auf welche Weise die Individuen für die Erzeugung der Folgegeneration im Selektionsprozess gefunden werden.

Eine Auswahl von Ersetzungsschemata:

Zunächst werden alle Individuen anhand ihrer Reihenfolge sortiert.

- Elitism

  Die n fittesten Individuen werden ausgewählt.

- Roulette Wheel

  Den Individuen wird entsprechend ihrer Reihenfolge eine Wahrscheinlichkeit zugeordnet, nach der sie zur Generierung der Folgegeneration herangezogen werden. Über die Auswahl entscheidet dann ein Zufallsgenerator.

- Elitism Roulette Wheel

  Von n zu selektierenden Individuen werden k Individuen nach Elitism und $n - k$ nach Roulette Wheel bestimmt.

Fitnessfunktion/Fitness Die *Fitness* soll die Entfernung des Individuums vom Optimum im Lösungsraum abbilden. Als Beispiel sei die Suche nach dem Scheitelpunkt einer Parabel genannt. Eine denkbare *Fitnessfunktion* wäre hier der Betrag der ersten Ableitung.

Mutation Die zufällige Änderung eines oder mehrerer Merkmale in einem Genotyp[15] wird als *Mutation* bezeichnet.

[15] Im Kontext der Genetischen Algorithmen

Rekombination (Cross-Over) Für die *Rekombination* werden nach definiertem Verfahren einzelne Merkmale von verschiedenen Genotypen ausgewählt und zu einem neuen Genotyp zusammengesetzt.

Durchquerung des Suchraums

Auf Abbildung 2.8 ist ein dreidimensionaler Graph zu sehen. X- und Z-Achse stehen jeweils für die Ausprägung eines Parameters[16]. Die Y-Achse steht für die Fitness. Das Optimum liegt in der Abbildung also auf dem Maximum[17].

Der Genetische Algorithmus bestimmt die Anfangskoordinaten bei seiner Initialisierung per Zufall. Im Hinblick auf Abbildung 2.8 muss man sich die Ausgangspopulation als eine Menge von Punkten vorstellen, die zufällig verstreut im dargestellten Lösungsgebirge liegen. Die Individuen auf den höchsten Punkten werden für die Reproduktion ausgewählt.

Rekombination stellt eine Möglichkeit dar, den Suchraum zwischen zwei Punkten zu durchsuchen. Die Mutation ermöglicht das Durchqueren des Suchraumes auch in andere Richtungen. Wird ein Individuum aus Generation n nach Generation $n+1$ übernommen, ist dies als Verbleiben auf einem Punkt zu interpretieren[18].

Die Auswirkungen von Mutationen lassen sich in Abbildung 2.9 betrachten. Diese sind in keiner Weise determiniert. Anders als bei Neuronalen Netzen sind Richtung und Entfernung vom Ausgangspunkt zufällig.

Nachteile

- Bei einer schlecht gewählten Fitnessfunktion wird unter Umständen am Ziel vorbei optimiert.
- Genetische Algorithmen können sehr rechenaufwändig sein. Dieser Nachteil tritt insbesondere dann auf, wenn die Ermittlung der Fitness durch die Fitnessfunktion sehr rechenaufwändig ist. Dies ergibt sich aus der Tatsache, dass nicht nur ein Individuum berechnet werden muss, sondern alle Individuen einer Generation und dies über mehrere Generationen hinweg.

[16] Der Lösungsraum für die Generierung möglicher Neuronaler Netze besitzt deutlich mehr Parameter und damit auch Dimensionen.

[17] Der besseren Anschaulichkeit wegen ist hier als Optimum das Maximum und nicht, wie im eigentlichen Verfahren, das Minimum gewählt.

[18] Dies trifft nur zu, wenn ein bestimmter Genotyp einen bestimmten Phänotyp deterministisch erzeugt. Somit nicht für das in dieser Arbeit vorgestellte Verfahren.

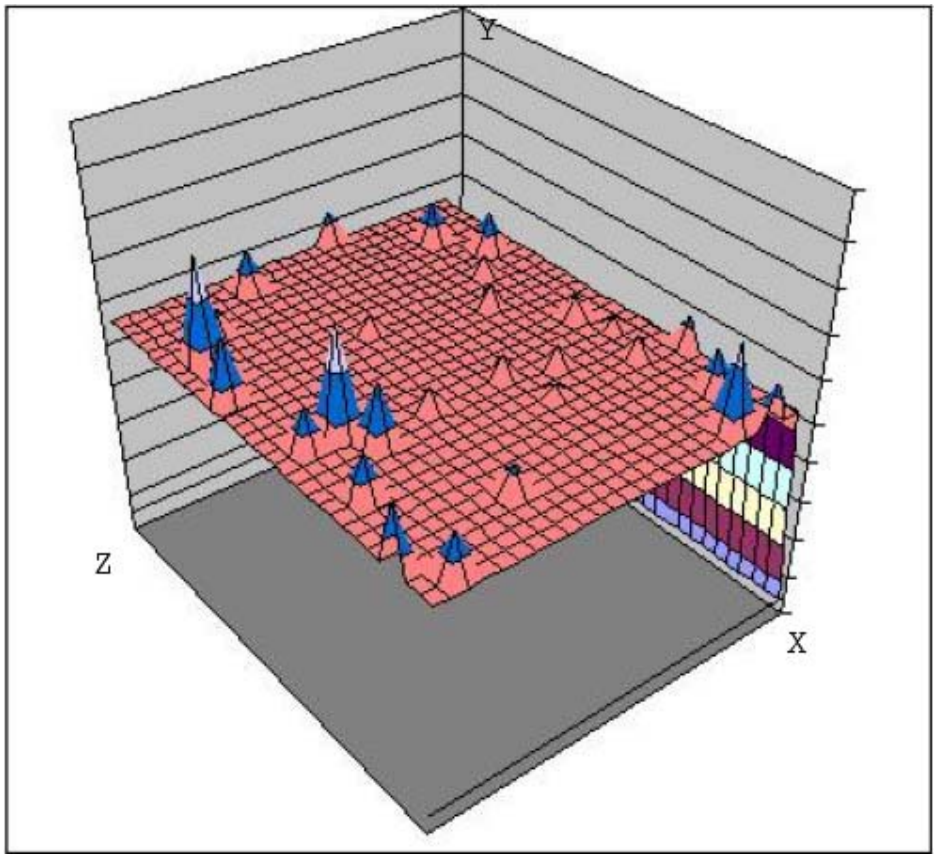

Abbildung 2.8.: Suchraum (Quelle: [Porschke:2002] / modifiziert)

Warum ein hybrider Ansatz?

Die Idee zu einer Kombination zwischen Genetischen Algorithmen und Neuronalen Netzen entstammt der Betrachtung der Natur selbst. J. David Schaffer, Darrell Whitley und Larry J. Eshelman schreiben dazu:

> Perhaps the most intuitively obvious way to combine a genetic algorithm with a neural network it (sic) to try to imitate nature by having the genotype code the architecture or topology (how many neurodes to use and how to connect them) of the network and then use another local learning method to fine tune the weights.[19]

[19] Schaffer u. a. (1992)

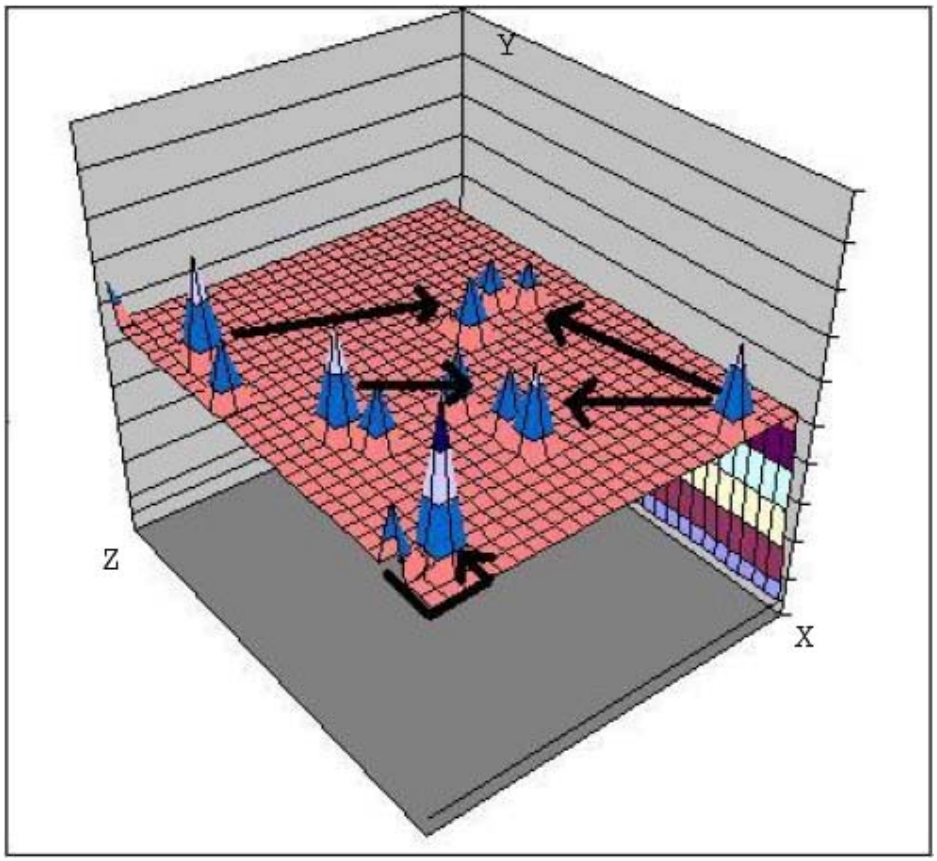

Abbildung 2.9.: Bewegung im Suchraum (Quelle: [Porschke:2002] / modifiziert)

Wie bereits erwähnt, basieren Neuronale Netze auf dem Prinzip der Fehlerminimierung durch Gradientenabstieg im Fehlergebirge, was das Risiko, in ein lokales Minimum zu laufen, mit sich bringt. Zweck der Kopplung ist es, mit Hilfe der Genetischen Algorithmen solche Neuronale Netze zu finden, die eher in ein globales Minimum laufen. Diese Idee stößt allerdings an Grenzen. So laufen Neuronale Netze *während* eines Lernvorganges in ein lokales Minimum. Der Genetische Algorithmus greift aber immer erst *nach* dem Lernvorgang für eine ganze Generation. D.h. Letzterer kann dem Neuronalen Netz nicht direkt helfen, aber durch die Selektion über mehrere Generationen werden solche Neuronalen Netze bevorzugt, die weniger früh in ein lokales Minimum laufen als andere.

Ein alternativer Weg, ein gutes Neuronales Netz zu finden, ist das Ausprobieren per Hand. Dies bringt einige Probleme gegenüber dem hybriden Ansatz mit sich. Um mit Neuronalen Netzen vernünftig umzugehen, bedarf es einiger Erfahrung. D.h. man braucht zu deren händischen Optimierung eine Fachkraft. Der andere Nachteil ist, dass ein Experte, wenn auch mit Sinn und Verstand, bei der Konstruktion seiner Neuronalen Netze auf Stichproben angewiesen ist. Daraus folgt, dass die Suche nach einem guten Netz auch in Expertenhand immer zu einem guten Teil auf Glück beruht. Genetische Algorithmen führen zwar auch nicht deterministisch zu einem optimalen Ergebnis. Ihre Fähigkeiten beim Durchsuchen großer Suchräume lassen aber die begründete Hoffnung zu, gute Ergebnisse mit ihnen erzielen zu können.

Problem des hybriden Ansatzes

Das Hauptproblem dieses Lösungsweges ist die Rechenzeit, die vor allem durch die Ermittlung der Fitness durch die Fitnessfunktion beansprucht wird. Wenn das Vorhaben darin besteht, Mustererkennung durch Neuronale Netze mit den Genetischen Algorithmen zu optimieren, ist es unerlässlich, die Erkennungsleistung oder ein brauchbares Äquivalent, wie zum Beispiel den quadratischen Fehler am Ende des Netztrainings, mit in die Fitnessfunktion einzubeziehen.

Schon das Training eines Netzes kann einen enormen zeitlichen Aufwand bedeuten. Durch die Kopplung mit einem Genetischen Algorithmus wird die Rechenzeit noch um ein Vielfaches erhöht.

Bei 48 Netzen pro Generation und 15 Generationen kommt man bereits im Mittel auf eine Verlängerung der Laufzeit um den Faktor 720. Dies ist nicht akzeptabel, wenn man zu Grunde legt, dass auf einem handelsüblichen Rechner, wie er für diese Arbeit Verwendung findet, die Trainingszeit für ein Netz abhängig von Lernzyklen, Topologie und Lernfunktion deutlich im Minuten- oder gar Stundenbereich liegt.

Es besteht also Zwang zur Optimierung.

2.3. Nebenüberlegungen

Hints

Um Neuronale Netze bei der Klassifikation zu unterstützen, kann versucht werden, zusätzliche Informationen über den zu erkennenden Gegenstand aus dem Dateneingang zu gewinnen. Solche Informationen, die im weiteren Verlauf auch in das Netztraining einfließen, nennt man *Hints*. Diese sollen dem Netz die Wichtigkeit bestimmter Details vermitteln.

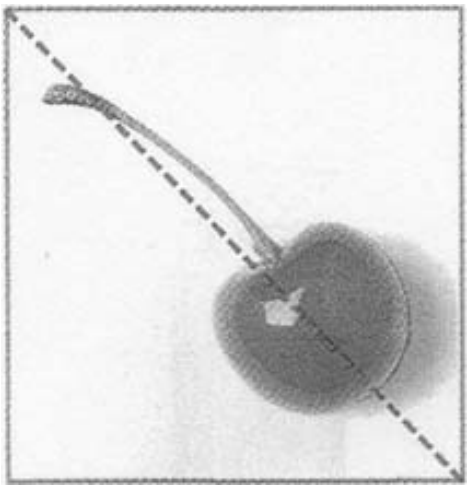

Abbildung 2.10.: Kirsche (Quelle: [Abu-Mostafa:1995])

Abbildung 2.11.: Uhr (Quelle: [Abu-Mostafa:1995])

Ein einfaches Beispiel ist das Vorhandensein von Symmetrie[20]. So könnte ein vorgeschalteter Algorithmus erkennen, ob ein Eingabemuster mehr oder weniger symmetrisch ist und diese Information zusätzlich zu den eigentlichen Eingabedaten an das Neuronale Netz übergeben.

Ein weiteres Beispiel für einen Hint ist dem Verfahren zu vermitteln, dass eine bestimmte Information unwichtig ist. Die Abbildungen 2.12 und 2.13 zeigen Eingangsinformationen, die für einen Computer sehr unterschiedlich sind, obwohl ein Mensch deren Äquivalenz sofort erkennt. Die Information, dass im einen Fall die Position des Objektes und im anderen Fall dessen Größe keinerlei Auswirkung auf dessen Kategoriezugehörigkeit hat, kann sehr wichtig für ein Neuronales Netz sein.

Abbildung 2.12.: Köpfe (Quelle: [Abu-Mostafa:1995])

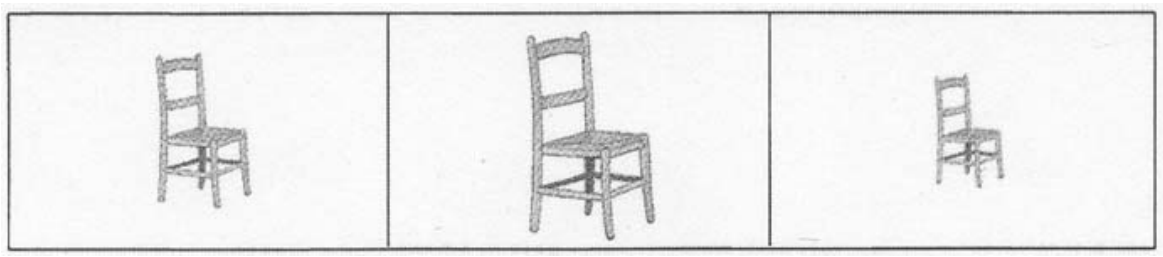

Abbildung 2.13.: Stühle (Quelle: [Abu-Mostafa:1995])

Bewertung

Für das verwendete Verfahren stellt sich die Frage, wie hilfreich solche Hints sein können. So hat zum Beispiel die Information, ob ein Objekt symmetrisch ist oder nicht, eine gewisse Bedeutung für die Unterscheidung von maschinengeschriebenen Druckbuchstaben. Liegen die selben Buchstaben von Hand geschrieben und als Schreibschrift vor, ist die Information bedeutungslos.

[20] Siehe hierzu die Abbildungen 2.10 und 2.11

Ähnliches gilt für die Position von Objekten. Diese spielt bei der Erkennung von Blättern keinerlei Rolle, für Buchstaben aber sehr wohl.

Bei der Größe von Objekten kann die Bewertung sehr ambivalent sein. Während diese Information für die Unterscheidung zwischen einem Ahorn- und einem Birkenblatt von großer Bedeutung sein kann, ist sie beim Auseinanderhalten von Ahorn und Platane wenig hilfreich.

Im vorgestellten Verfahren werden Hints nicht verwendet, obwohl der Autor diese in vielen Fällen für durchaus sinnvoll hält. Dafür gibt es zwei Gründe:

- In dieser Arbeit werden zwei sehr unterschiedliche Objekttypen untersucht. Das entwickelte Verfahren soll beide Objekttypen ohne umfangreiche Änderungen verarbeiten können. Um Hints zu finden, die in beiden Fällen hilfreich sind, müsste erhebliche Zeit aufgewendet werden. Dies würde aber den Schwerpunkt der Arbeit auf eine Art und Weise verlagern, wie es nicht gewünscht ist.
- Das Problem der Rechenzeit wurde für den hybriden Ansatz, den diese Arbeit verfolgt, bereits weiter oben angesprochen. Da Hints zusätzliche Eingangsdaten generieren, könnte deren Verwendung dieses Problem noch weiter verschärfen[21].

Vergleichbarkeit

Zur Beurteilung des beschriebenen Verfahrens bedarf es einer Referenz in Form von anderen Neuronalen Netzen. Anhand dieser kann die Leistungsfähigkeit des Verfahrens eingeordnet werden. Denkbares Herangehen bei der Erstellung solcher Referentznetze wäre die Verwendung einer Heuristik, welche die folgenden Parameter berücksichtigt:

- Komplexität der Eingangsdaten (womit hier die Größe des Eingangsvektors gemeint ist)
- Anzahl der Eingabemuster
- Anzahl der Kategorien, die das Netz lernen soll

[21] Die Wirkung kann sehr unterschiedlich ausfallen. Im Fall von Buchstaben, denen noch drei zusätzliche Neuronen mitgegeben werden, um damit anzuzeigen in welchen Bereichen (oben, mitte, unten) sich Teile eines Buchstabens befindet, ergeben sich vermutlich keine Probleme. Wird mit einem Bild aber zusätzlich noch die kantengefilterte Variante des gleichen Bildes mit übergeben, verdoppelt sich die Anzahl der Eingangsdaten, was zu erheblichen Verlängerung des Verfahrens führen dürfte.

In keiner seriösen Quelle konnte eine Heuristik aufgefunden werden[22], die solchen Kriterien entspricht. Als denkbare Alternative bietet sich ein Referenznetz mit einer Netzarchitektur an, die die selben Daten auf „konventionelle“ Weise, also ohne genetische Komponente, verarbeitet und somit Ergebnisse erzeugt, die als Maßstab dienen können[23]. Die dazu verwendete Netztopologie würde dem entsprechen, was nach der Erfahrung des Autor eine sinnvolle Einstiegstopologie für eine händische Optimierung ist.

[22] Ein Überblick über die Problematik findet sich unter Sarle (2004)
[23] Mehr dazu in Kapitel 5.1 Seite 72

3. Aufbau des Optimierungssystems

3.1. Genetische Komponente

Individuum

Genotyp

Der Genotyp besteht zum einen aus den topologischen Netzdaten auf Ebene der Neuronen respektive der Layer. Hier wird angegeben, wie viele Neuronen es in welchem Layer gibt und welche Aktivierungsfunktion jeweils in den Neuronen verwendet wird. Die Zuteilung der Aktivierungsfunktion geschieht auf Ebene der Layerarten. Es gibt also eine Aktivierungsfunktion für alle Input Neuronen, eine für alle Hidden Neuronen und eine für alle Output Neuronen.

Der andere Teil des Genotyps besteht aus der beim Training des Netzes verwendeten Lernfunktion und den dazugehörigen Parametern, die je nach Funktion variieren. Welche Lernfunktionen hier genau verwendet werden, wie sie funktionieren und welche Stellgrößen zu den einzelnen Lernfunktionen gehören, wird weiter unten erläutert.

Der Genotyp wird in der Literatur oft als Bitstring codiert [1]. In dieser Arbeit wird hingegen eine objektorientierte Implementierung verwendet. Diese bietet nach Auffassung des Autors gegenüber der Bitstring-Codierung den Vorteil besserer Lesbarkeit.

Das gewählte Design sieht ein Objekt der Klasse *Individuum* vor, das eine Aggregationsbeziehung zu je einem Objekt der Klasse *Net* sowie zu einer Implementierung der Schnittstelle *LearningFunction* besitzt. Letztgenannte Implementierungen stellen die *Alleles* des Gens Lernfunktion dar. Es gibt je eine Klasse für die vier hier verwendeten Lernfunktionen mit den dazugehörigen Parametern. Die Informationen über die Aktivierungsfunktionen der Neuronen des Neuronalen Netzes werden in der Klasse *ActivationSetting* zusammengefasst, welche als Teil in einer Aggregationsbeziehung zur Klasse *Net* steht[2].

[1] Siehe dazu Mitchell (1996),Lämmel und Cleve (2001) und Luger (2001)

[2] Eine Grafische Darstellung dieser Zusammenhänge findet sich auf Abbildung 3.1

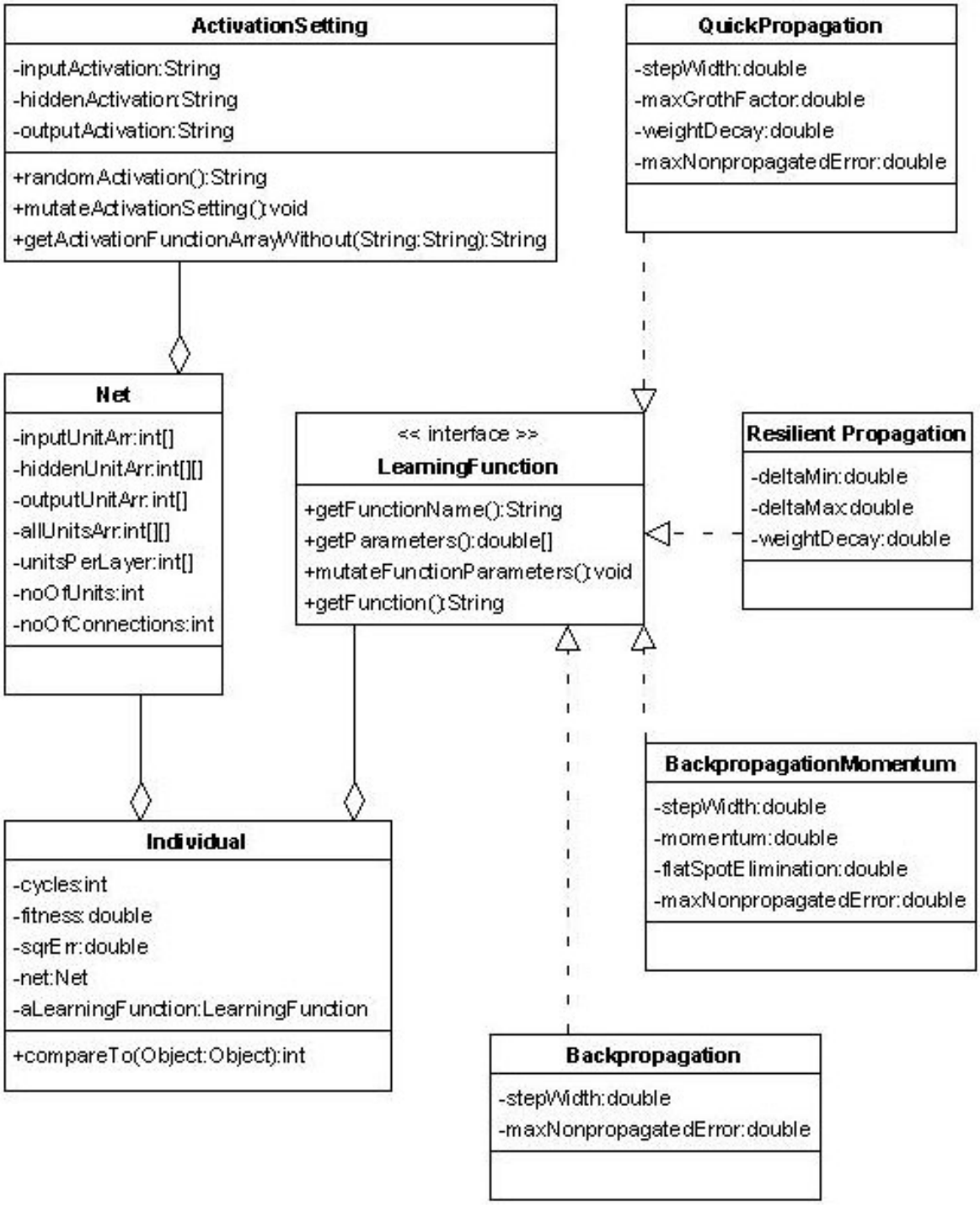

Abbildung 3.1.: Aufbau eines Individuums

Phänotyp

Der Phänotyp ist hier ein komplett initialisiertes Neuronales Netz, das die Informationen des Genotyps, darüber hinaus aber auch die Gewichte zwischen den einzelnen Neuronen sowie das jeweilige Bias pro Neuron enthält. Es sei hier explizit darauf hingewiesen, dass sowohl die Gewichte zwischen den Neuronen als auch die Bias nicht Teil der Optimierung sind.

Die Daten des Genotyps, der als Grundlage für den Phänotyp dient, werden immer als vollständig verbundenes Feedforward Netz ohne Shortcut Connections[3] interpretiert, was bedeutet, dass alle Neuronen des Layers n (vom Input Layer aus gesehen) mit allen Neuronen des Layers $n + 1$ verbunden sind. Vom Output Layer gehen keine Verbindungen ab. Es gibt keine Verbindungen vom hinteren zum vorderen Layer und auch keine Verbindungen innerhalb einer Schicht.

Population

Eine Population besteht aus maximal 48 Individuen. Nach der Erzeugung der Folgegeneration, in die jeweils die Genotypinformationen der besten 16 Individuen, deren Mutanten sowie die 16 Rekombinationsergebnisse einfließen, wird zunächst ermittelt, ob es Genotypen gibt, die mehrfach vorkommen. Diese werden ggf. gelöscht. Die übrigen Genotypen bilden die Population einer neuen Generation.

Selektion

Jedes Netz wird solange trainiert, bis eines der folgenden Ereignisse eintritt:

1. Der gewünschte Quadratische Fehler (der pro Generation etwas abnimmt) wird unterschritten.
2. Die Anzahl der Trainingszyklen ist größer als die Anzahl der Trainingszyklen des 16. schlechtesten Individuums aus der Vorgeneration multipliziert mit einem Faktor x[4].

Nach dem Lauf wird für jedes Individuum die Fitness ermittelt. Die 16 Netze mit der besten Fitness werden zur Fortpflanzung für die Folgegeneration selektiert, was dem Elitism-Ersetzungsschema[5] entspricht.

[3] Siehe hierzu Abbildung 3.4 „Feedforward“
[4] Bei meinen Voruntersuchungen hat sich ein x von 1,2 bewährt
[5] Siehe Kapitel 2.2 Seite 23

Als ein Vorteil des Elitism-Ersetzungsschemas gilt die nicht abnehmende Qualität von Generation zu Generation[6]. Dies gilt aber nur dann, wenn aus einem Genotypen immer der gleiche Phänotyp entsteht. Letzteres ist im vorgestellten Verfahren nicht der Fall. Somit gilt dieser Vorteil hier nicht.

Fitnessfunktion

In die Fitnessfunktion fließen drei Werte ein. Der wichtigste Wert ist das Fehlerquadrat, das der Simulator für Neuronale Netze am Ende der Validierung des Netzes zurückgibt. Dies ist das Hauptqualitätsmerkmal eines Netzes, auch wenn man durch diesen Wert nur indirekt auf die Erkennungsqualität des jeweiligen Netzes schließen kann. Die beiden übrigen Werte beziehen sich auf die Rechenaufwändigkeit des Algorithmus:

- Die Anzahl der Zyklen, welche ein Netz bis zum Ende der Berechnung benötigt
- Die Anzahl der Verbindungen im Neuronalen Netz

Von diesen beiden Werten ist jeweils der Logarithmus an der Berechnung der Fitnessfunktion beteiligt. Sie dienen der Beschleunigung des Verfahrens, da weniger Lernzyklen und Netzverbindungen auch weniger Rechenzeit bedeuten.

Die Bestimmung der Fitness verursacht einen weitaus größeren Rechenaufwand als alle anderen Teilaufgaben der Software. Dies liegt nicht etwa an der Berechnung der eigentlichen Fitnessfunktion, sondern an dem Umstand, dass zur Ermittlung der in die Funktion einfließenden Parameter das Trainieren und Validieren der Neuronalen Netze unumgänglich ist.

Der Vergleich der Fitness zweier Netze aus unterschiedlichen Generationen ist nicht zulässig. Der Grund dafür ist in der Variablität der erlaubten Lernzyklen in verschiedenen Generationen zu suchen.

Mutation

Von jedem selektierten Netz wird genau eine Mutante erzeugt. Diese muss sich nicht notwendigerweise von ihrem Vorgänger unterscheiden. Zur Ermittlung, ob und was während der Mutation verändert wird, wird die Mutationsrate herangezogen. Diese liegt bei allen durchgeführten Experimenten bei 0,3. D.h. es gibt für jedes mutierbare Merkmal eine 30% Wahrscheinlichkeit für eine Veränderung.

[6] Wenn Qualität die Fitness des besten Individuums meint.

Durch Mutation veränderliche Parameter:

1. Lernfunktion mit dazugehörigen Stellgrößen
2. Aktivierungsfunktion
3. Netztopologie (in Form der Anzahl der Neuronen in den Hidden Layer)

Mutation - Lernfunktion

Verwendete Lernfunktionen sind Standard Backpropagation, Resilient Propagation, Backpropagation Momentum sowie Quickpropagation[7]. Die neue Lernfunktion wird nach dem Zufallsprinzip bestimmt. Dass im Falle einer Mutation die bereits verwendete Lernfunktion erneut zu einer Eigenschaft des Individuums wird, ist ausgeschlossen. Kommt es bei der Mutation zur Veränderung der Lernfunktion, erhält diese Standard Parameter. Dies ist deshalb nötig, weil jede Lernfunktion andere Stellgrößen besitzt. Selbst die Lernrate ist nicht in allen Lernfunktionen als Parameter enthalten.

Unabhängig davon, ob sich die Lernfunktion geändert hat oder nicht, kommt es im nächsten Schritt zu einer Mutation der Lernfunktionsparameter. Hierbei gilt für jeden Parameter die oben genannte Wahrscheinlichkeit für eine Veränderung. Wird festgestellt, dass eine Mutation auftritt, wird der entsprechende Wert mit einer 50% Wahrscheinlichkeit erhöht oder gesenkt.

Mutation - Aktivierungsfunktion

Bei der Mutation der Aktivierungsfunktionen[8] wird für jede Layer-Art einmal mit der gegebenen Veränderungswahrscheinlichkeit überprüft, ob diese ausgetauscht wird oder nicht. Die neue Aktivierungsfunktion wird nach dem Zufallsprinzip bestimmt. Dass im Falle einer Mutation die bereits verwendete Aktivierungsfunktion erneut zu einer Eigenschaft des Individuums wird, ist ausgeschlossen.

Mutation - Netztopologie

Die Mutation der eigentlichen Netze geschieht auf Ebene der Hidden Layer. Für jeden Hidden Layer im Netz wird gegen die Mutationswahrscheinlichkeit geprüft, ob es zu einer Veränderung kommt. Kommt es zu einer Veränderung, wird ein Zufallswert ermittelt und die Neuronenanzahl des betreffenden Layers auf den ermittelten Wert gesetzt.

[7] Eine Erläuterung der Lernfunktionen erfolgt in diesem Kapitel ab Seite 40
[8] Zu den verwendeten Aktivierungsfunktionen siehe Kapitel 3.1 ab Seite 45

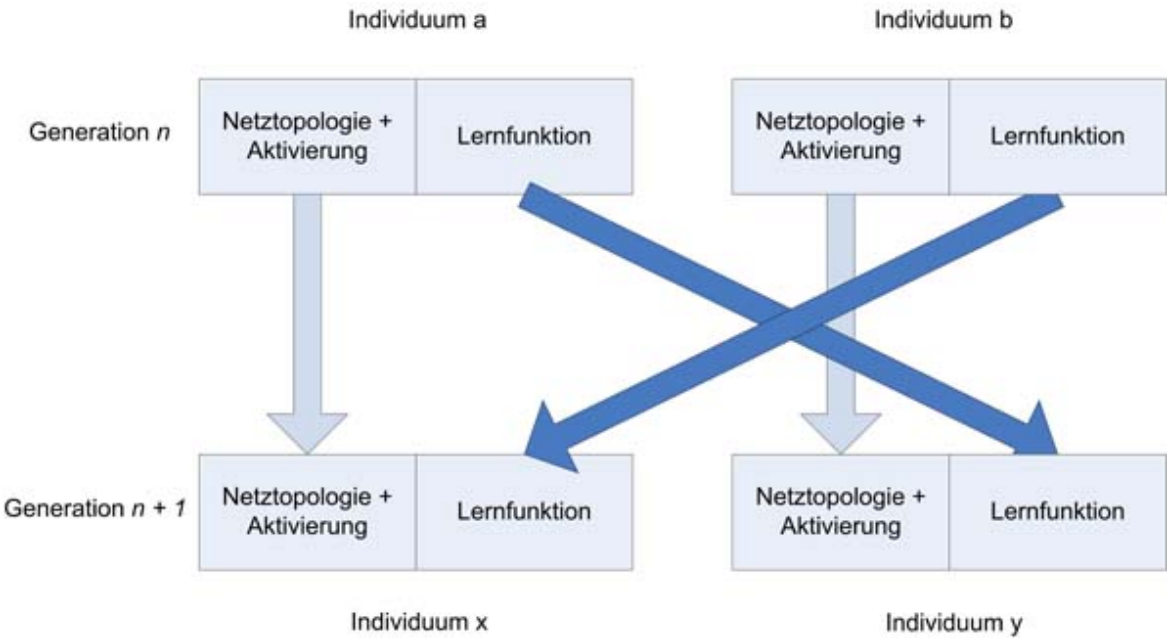

Abbildung 3.2.: Rekombination

Rekombination (Cross-Over)

Aus der Rekombination entstehen 16 Individuen, die jeweils aus Paarungen des Erstbesten mit dem Zweitbesten, des Drittbesten mit dem Viertbesten und so weiter bis zum Sechzehntbesten Individuum der Vorgeneration zusammengesetzt werden. Der eine Partner bringt die Netztopologie und die Aktivierungsfunktionen ein, während der andere Partner seine Lernfunktion mit den entsprechenden Parametern dazugibt. Dies geschieht pro Paarung reziprok[9].

[9] Zur Verdeutlichung siehe Abbildung 3.2. Zum Verstehen des Rekombinationsprozesses lohnt sich auch ein Blick auf das Klassendiagramm in Abbildung 3.1. Daraus geht hervor, dass für die hier realisierte Form der Rekombination lediglich zwei Klassen, nämlich *Net* und *LearningFunction*, ausgetauscht werden müssen.

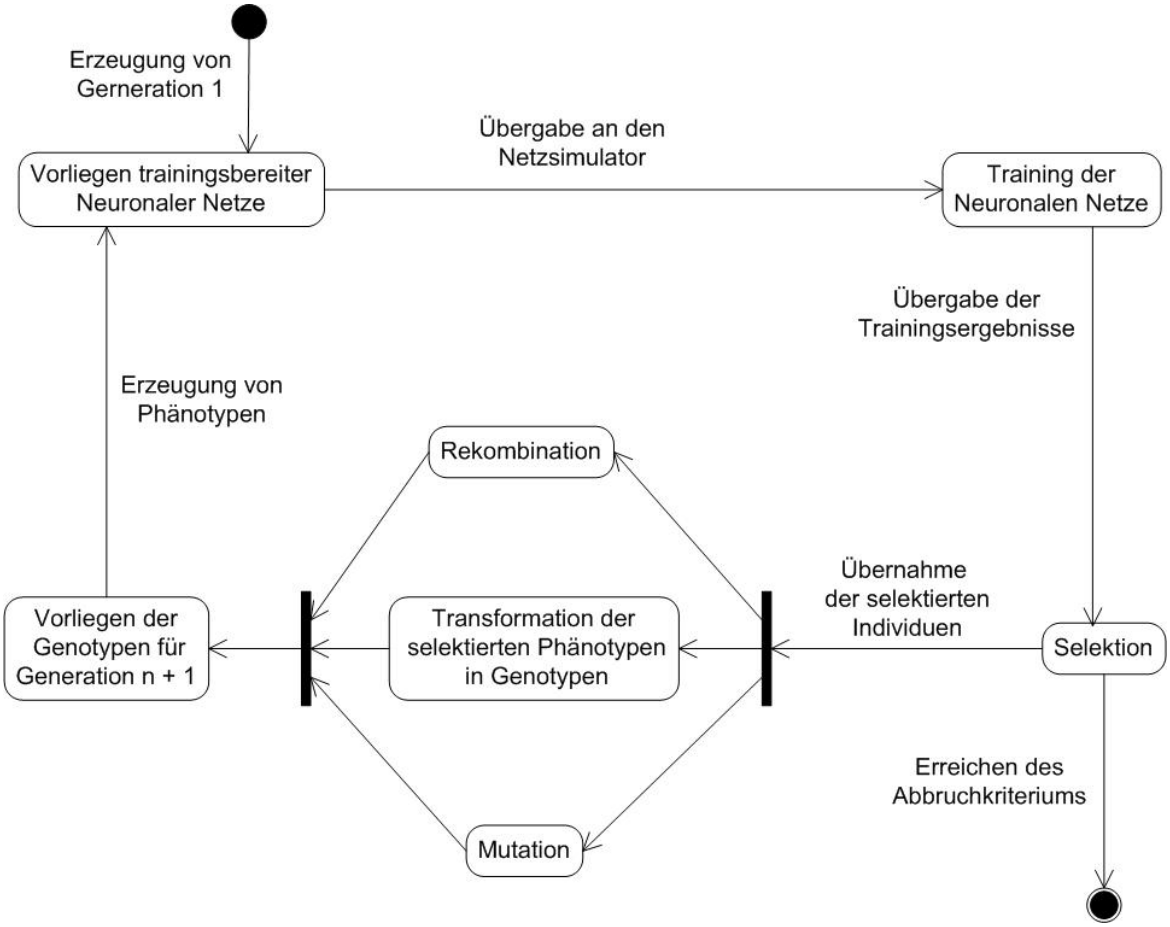

Abbildung 3.3.: Zyklus des verwendeten Genetischen Algorithmus

Zyklus

Nachdem die wichtigsten Teilaspekte des Genetischen Algorithmus besprochen sind, wird nun der komplette Ablauf pro Generation beschrieben. Hierbei wird an Stellen, an denen die Software Wahlmöglichkeiten ließe, nur auf die tatsächlich getroffene Wahl eingegangen. Ein Beispiel für eine solche Wahlmöglichkeit ist die Anzahl der Individuen pro Generation. Für das vorgestellte Verfahren kommen alle Generationsgrößen, die ganzzahlig durch sechs teilbar sind, in Betracht. In dieser Arbeit wird jedoch ausschließlich eine Generationsgröße von achtundvierzig[10] Individuen verwendet.

Zunächst werden per Zufall achtundvierzig Individuen erzeugt. Der Zufall bestimmt hierbei pro Netz die Netztopologie, die Aktivierungsfunktionen der einzelnen Layer, die Lernfunktion[11], das Bias der einzelnen Neuronen sowie die Gewichtungen der Verbindungen zwischen den Neuronen.

Diese gesammelten Netze werden mit entsprechendenen Steueranweisungen an einen Simulator von Neuronalen Netzen[12] übergeben. Der Simulator trainiert die Neuronalen Netze anhand der gegebenen Trainingsmuster. Zur Kontrolle überprüft der Simulator nach jedem Lernzyklus mit Hilfe der Validierungsmuster die Generalisierungsfähigkeit des jeweiligen Netzes, was hier über den quadratischen Fehler geschieht. Abbruchkriterium ist das Erreichen eines vorgegebenen quadratischen Fehlers oder das Erreichen einer bestimmten Anzahl von Lernzyklen. Beide Abbruchkriterien variieren von Generation zu Generation.

Nach dem Training gibt der Netzsimulator die erreichten Fehlerquadrate und die jeweils benötigten Lernzyklen für sämtliche Netze zurück. Diese Werte sowie die Größe der Neuronalen Netze fließen in die Fitnessfunktion[13] ein, anhand derer eine Rangfolge der Individuen erstellt wird. Aus dieser Rangfolge werden die 16 besten Individuen selektiert[14].

Aus den Genotypen dieser 16 Individuen wird nun auf drei parallel laufenden Wegen die Folgegeneration gebildet:

- Die Individuen fließen unverändert in die Folgegeneration ein.
- Aus den Individuen werden durch Rekombination wie in Kapitel 3.1 neue Phänotypen erzeugt.
- Aus den Individuen werden durch Mutation wie in Kapitel 3.1 neue Phänotypen erzeugt.

[10] Achtundvierzig Individuen haben sich bei den Voruntersuchungen als brauchbare Anzahl bewährt.
[11] Die zur Lernfunktion gehörigen Parameter sind die in Kapitel 3.1 Seite 40 erwähnten Funktionsparameter mit entsprechender Standardeinstellung
[12] Zum gewählten Simulator siehe Kapitel 4.2 Seite 57
[13] Siehe Kapitel 3.1 Seite 35
[14] Siehe Kapitel 3.1 Seite 34

Nach diesem Prinzip entstehen 48 neue Genotypen. Es kann durchaus geschehen, dass hierbei zwei oder mehr gleiche Genotypen entstehen. Tritt dieser Fall auf, wird jeweils nur ein Individuum trainiert, so dass in den meisten Fällen nicht 48, sondern weniger Individuen durch den Selektionsprozess laufen. Dieses Vorgehen ist von dem Wunsch motiviert, „Monokulturen“[15] zu vermeiden. Dass solche Monokulturen entstehen können, hat sich bei den Voruntersuchungen gezeigt. Die Suche im Lösungsraum soll in „Breite“[16] und nicht in „Tiefe“[17] erfolgen.

Ist dieser Prozess einmal durchlaufen, beginnt der Zyklus von vorne, bis ein entsprechendes Abbruchkriterium erfüllt ist.

Lernfunktionen

Alle verwendeten Lernfunktionen gehören zu den Verfahren überwachten Lernens[18]. Die Teilaspekte bei überwachtem Lernen sind:

- Es gibt eine Menge von Mustern, die die zu lernenden Kategorien repräsentieren
- Für jedes Muster ist eine eindeutige Zuordnung zu einer dieser Kategorien möglich
- Diese Zuordnung wird dem Lernverfahren mitangegeben

Die Muster werden über einen Zahlenvektor im Input Layer angelegt. Die Zuordnung zu den entsprechenden Kategorien im Output Layer erfolgt über einen dort anliegenden Zahlenvektor.

Wie bereits weiter oben erwähnt, handelt es sich bei den verwendeten Neuronalen Netzen um sog. Feedforward Netze. Diese sind vom Input Layer über eine Anzahl von Hidden Layern mit dem Output Layer verbunden. Am Input Layer wird jeweils das Eingabemuster angelegt und im Output Layer die Kategorie, zu der dieses Muster gehört. Die Abbildungen 3.4 und 3.5 zeigen verschiedene Varianten von Feedforward-Netzen. Für die Backpropagation-Familie werden lediglich Feedforward-Netze ohne Shortcuts verwendet. Feedforward-Netze mit Rückkopplung finden immer dann Anwendung, wenn es darum geht mit Hilfe von Neuronalen Netzen eine zeitliche Dimension abzubilden.

[15] Gemeint sind Generationen, in denen sich die Genotypen nur auf Grund ganz weniger Merkmale oder gar nicht unterscheiden.
[16] Untersuchung möglichst vieler verschiedener Genotypen
[17] Untersuchung möglichst vieler verschiedener Phänotypen, die sich aus einem Genotyp erzeugen lassen
[18] Siehe hierzu 2.2 Seite 16

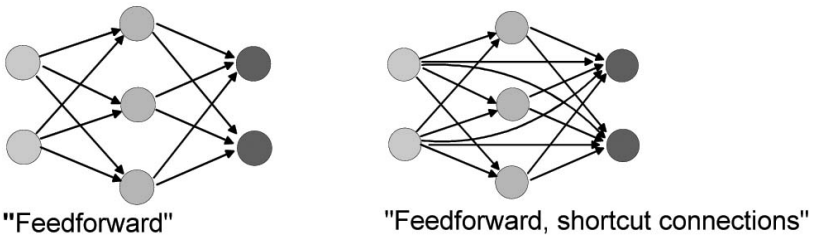

Abbildung 3.4.: Einfache Feedforward-Netze (Quelle: [Meisel:2004])

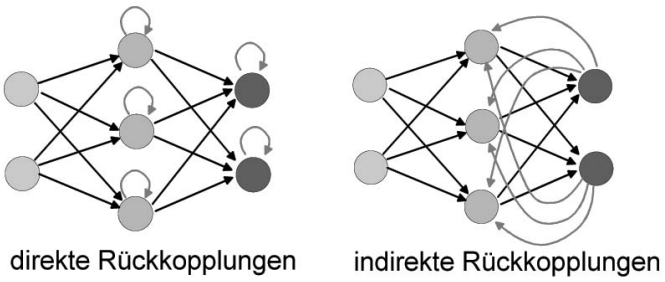

Abbildung 3.5.: Feedforward-Netze mit Rückkopplungen (Quelle: [Meisel:2004])

Die zwischen Input Layer und Output Layer liegenden Schichten stellen wir uns hier als Blackbox vor, da die Erläuterung der genaueren Zusammenhänge den Rahmen dieser Arbeit sprengen würden. Die im Input Layer angelegten Werte fließen in besagte Blackbox ein. Die Blackbox bildet die Werte auf dem Output Layer als Zahlenvektor ab.

Im Normalfall gibt es eine Abweichung zwischen den tatsächlich abgebildeten Werten und den Werten, die von der Zielkategorie als Zahlenwert angelegt wurden. Überwachtes Lernen versucht die Abweichung zwischen den abgebildeten und den angelegten Werten über Veränderungen in der Blackbox zu verringern[19].

Die zur Verringerung der Abweichung eingesetzte Funktion ist die Lernfunktion.

[19]Wie dies genau umgesetzt wird, ist sehr anschaulich in Lämmel und Cleve (2001) Kapitel 6 beschrieben

Backpropagation

Sämtliche in vorliegender Arbeit verwendeten Verfahren beruhen auf dem Grundprinzip von Standard Backpropagation (bzw. modifizieren es leicht). Standard Backpropagation geht auf D. E. Rumelhart, G. E. Hinton und R. J. Williams zurück[20]. Bei Backpropagation kennt man den Ergebnisvektor, den ein Netz bei einer Eingabe erzeugen soll. Am Anfang sind die Gewichte im Netz annähernd zufällig eingestellt. Durchlaufen nun die Eingabedaten des Inputlayers dieses Netz, wird ein Ergebnisvektor erzeugt, der nicht dem erwarteten Ergebnis entspricht. Abhängig vom Δ (also der Abweichung) zwischen gegebenem und erwartetem Ergebnis werden die Gewichte im Netz abhängig vom Gradienten verändert und das Verfahren beginnt von vorn.

Die Verwendung von Standard Backpropagation bringt drei Probleme mit sich:

- Das Risiko, in ein lokales Minimum[21] zu laufen, resultiert aus dem Gradientenabstieg im Fehlergebirge und betrifft zahlreiche Such-Algorithmen. In vorliegender Arbeit gilt das Problem für Standard Backpropagation und alle verwendeten Nachfolger.
- Backpropagation bei „flachem Gefälle“[22]. Das Gefälle hat direkten Einfluss auf die Veränderungen der interneuronalen Gewichte. Befindet sich die Fehlerfunktion des Standard Backpropagationverfahrens auf einem Plateau, kommt es kaum zu Veränderungen der Verbindungsgewichte. Dies sorgt für Zeitverluste.

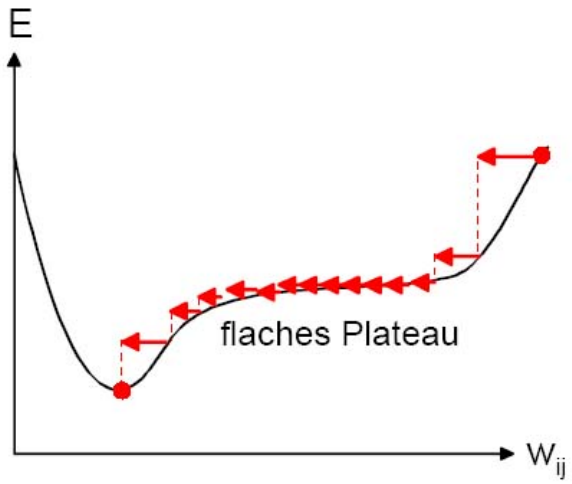

Abbildung 3.6.: Auf Plateaus kaum Fortschritt in Fehlerfunktion (Quelle: [Meisel:2004])

[20] Siehe auch Rumelhart u. a. (1986)
[21] Siehe hierzu: Abbildung 2.7 auf Seite 22
[22] Siehe hierzu: Abbildung 3.6 auf Seite 42

- Das Verhalten des Verfahrens in „Schluchten“. Hier kann es auf Grund des sehr hohen Gefälles zum Schwingen[23] zwischen den „Wänden“ der Schlucht oder aber zum Aussprung[24] kommen.

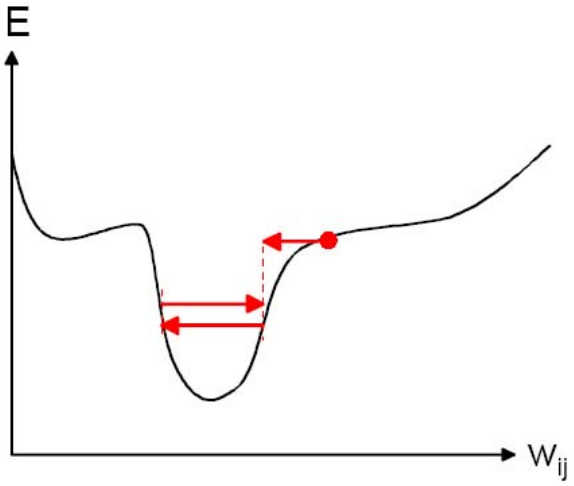

Abbildung 3.7.: Oszillation in steilen Minima der Fehlerfunktion (Quelle: [Meisel:2004])

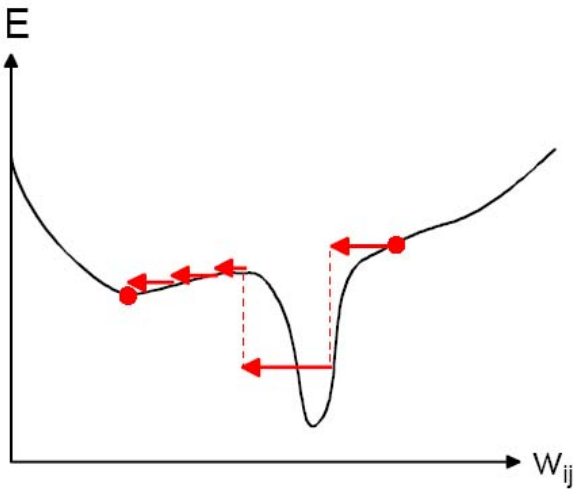

Abbildung 3.8.: Aussprung aus steilen Minima der Fehlerfunktion (Quelle: [Meisel:2004])

[23] Siehe hierzu: Abbildung 3.7 auf Seite 43
[24] Siehe hierzu: Abbildung 3.8 auf Seite 43

Diese drei verfahrensbedingten Probleme sind noch längst nicht alle Schwierigkeiten, die im Zusammenhang mit Standard Backpropagation auftreten können. Bezüglich der Funktionsweise des Neuronalen Netzes stellen sie jedoch die Haupt-Herausforderung dar. Für andere Verfahren bilden die genannten Probleme oft den Ausgangspunkt zur Weiterentwicklung des Grundprinzips von Standard Backpropagation.

Backpropagation Momentum

Zwei der oben genannten Probleme versucht das sogenante Backpropagation Momentum (in der Literatur oft unter Backpropagation mit Momentum Term und seltener unter „conjugate gradient descent“ zu finden) zu lösen. Dem oben beschriebenen „Schwingen“ begegnet das Verfahren durch eine Verringerung der Schrittweite nach einem Vorzeichenwechsel. Auf die Schwierigkeiten im „flachen Gefälle“ reagiert das Verfahren, indem es seine Schrittweite verlängert, sobald ein Plateau erreicht wird. Erreicht wird dieses Verhalten, indem der vorherige Schritt in die Berechnung mit einfließt. Das Verfahren geht auf Hinton und Williams[25] zurück.

Quickpropagation

Gestützt auf die Annahme, dass sich mit Hilfe einer quadratischen Funktion die Fehlerfunktion eines Neuronalen Netzes besser approximieren lässt, wählt das Quickpropagation Verfahren über eine Interpolation zwischen zwei Punkten t und $t-1$, bei der eine quadratische Funktion zu Grunde gelegt wird, den Punkt $t+1$ anhand des Scheitelpunktes der erwähnten Funktion.

Die erste Gewichtsänderung wird über Standard Backpropagation vorgenommen. Durch die Natur der quadratischen Funktion macht Quickpropagation an flachen Stellen weite Schritte und an steilen Stellen eher kleine. Das Verfahren geht auf Scott Fahlman[26] zurück.

Resilient Propagation

Dieses Verfahren lässt den Betrag der Steigung völlig außer Acht. Hier gilt im Wesentlichen: Erhöhe die Schrittweite für $t+1$, wenn es zwischen t und $t-1$ nicht zu einem Vorzeichenwechsel gekommen ist und wiederhole t mit verringerter Schrittweite, wenn das Vorzeichen wechselte. Resilient Propagation wurde von Martin Riedmiller und Heinrich Braun[27] vorgestellt.

[25] Siehe Hinton und Williams (1986)
[26] Siehe Fahlman (1988)
[27] Siehe Riedmiller und Braun (1992)

Aktivierungsfunktionen

Da die verwendeten Aktivierungsfunktionen nur einen Nebenaspekt vorliegender Arbeit darstellen, wird auf eine detaillierte Beschreibung verzichtet.

Auflistung der verwendeten Aktivierungsfunktionen:

- Logistische Funktion

$$a_j(t) = \frac{1}{1 + e^{-(net_j(t)+\Theta_j)}} \tag{3.1}$$

- Signum-Funktion

$$a_j(t) = \begin{cases} 1 & \text{für} \quad net_j(t) > 0 \\ 0 & \text{für} \quad net_j(t) = 0 \\ -1 & \text{für} \quad net_j(t) < 0 \end{cases} \tag{3.2}$$

- Tangens-Hyperbolicus-Funktion

$$a_j(t) = \tanh(net_j(t) + \Theta_j) \tag{3.3}$$

- Schrittfunktion

$$a_j(t) = \begin{cases} 1 & \text{für} \quad net_j(t) > 0 \\ 0 & \text{für} \quad net_j(t) \leq 0 \end{cases} \tag{3.4}$$

- Produktfunktion

$$a_j(t) = \prod_i w_{ij} o_i \tag{3.5}$$

- Identitätsfunktion

$$a_j(t) = net_j(t) \tag{3.6}$$

- Sinusfunktion

$$a_j(t) = \sin(net_j(t)) \tag{3.7}$$

- Exponentialfunktion

$$a_j(t) = \exp(net_j(t)) \tag{3.8}$$

3.2. Softwaredesign

Architektur des Genetischen Algorithmus

Zur Umsetzung des verwendeten Genetischen Algorithmus sind zwei Komponenten notwendig:

- Eine Genetikmaschine
- Ein Simulator für Neuronale Netze

Die internen Aufgaben der Genetikmaschine sind:

- Selektion von fortpflanzungswürdigen Individuen aus einer gegebenen Menge von Individuen.
- Die Generierung einer neuen Generation von Individuen auf Basis der zuvor selektierten Individuen.

Die internen Aufgaben des Simulators für Neuronale Netze sind:

- Trainieren der Netze anhand geeigneter Trainingsmustern
- Validieren der Netzqualität anhand geeigneter Validierungsmuster

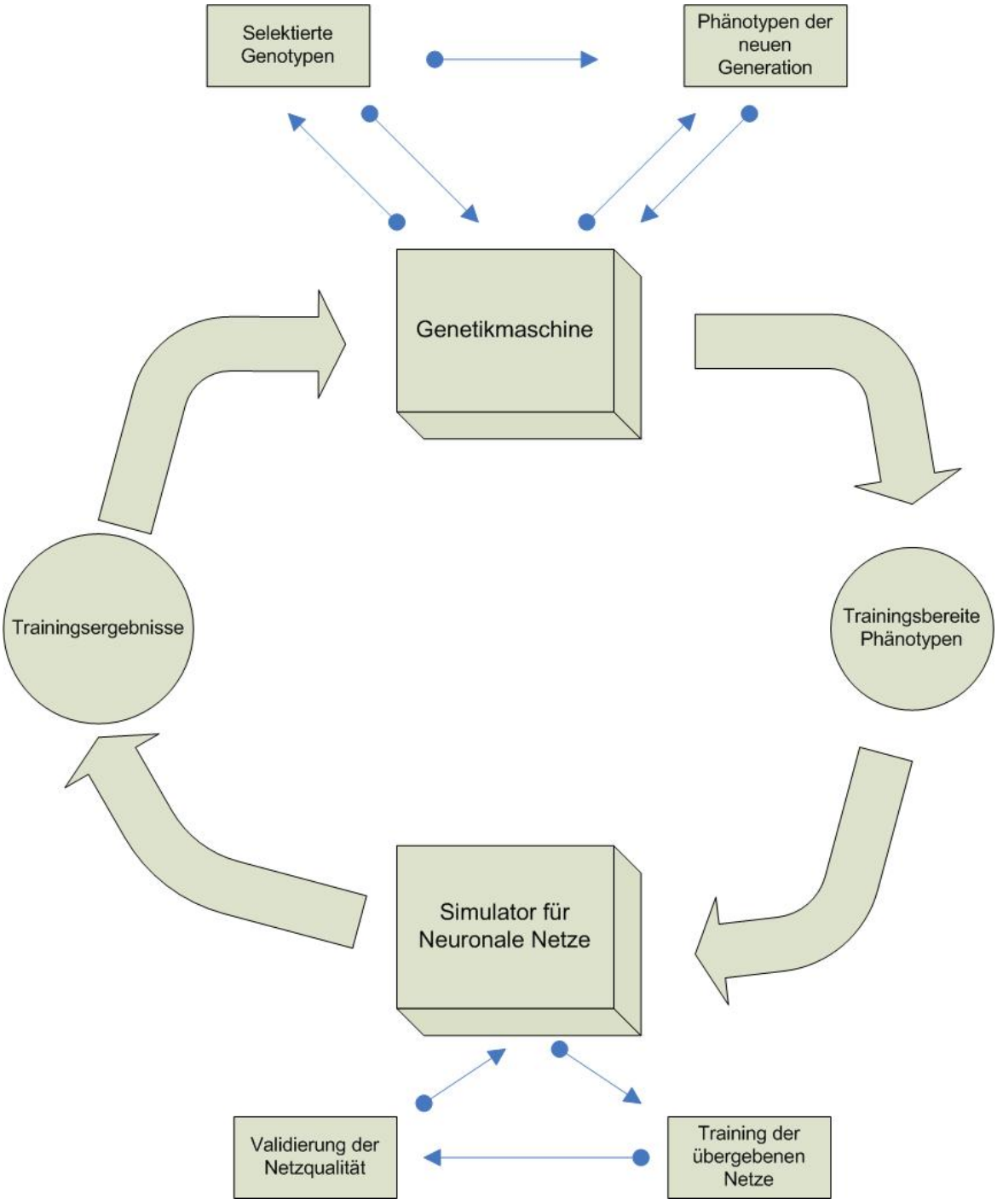

Abbildung 3.9.: Architektur: Zyklus

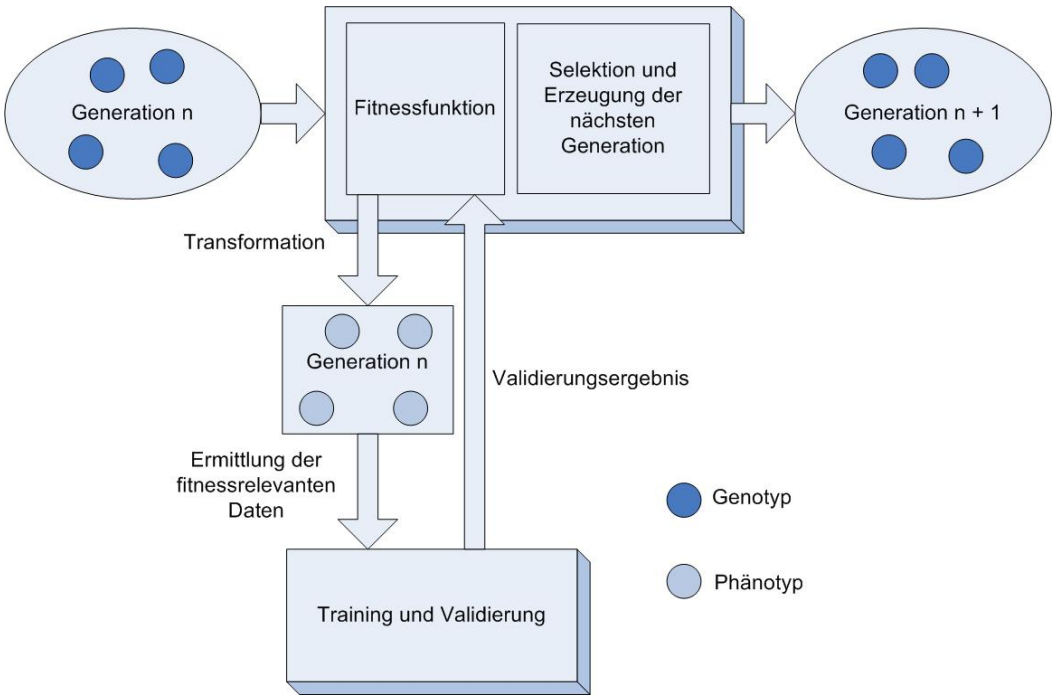

Abbildung 3.10.: Logische Kopplung

Darüber hinaus müssen beide Komponenten miteinander kommunizieren:

- Die Genetikmaschine muss eine neu erschaffene Generation von Individuen an den Netzsimulator in einer für diesen verständlichen Form übergeben.
- Der Netzsimulator muss die Ergebnisse aus Netztraining und -validierung an die Genetikmaschine in einer für diese verständliche Form übergeben.

Das Zusammenspiel der eben genannten Aspekte ist schematisch in Abbildung 3.9 zu sehen.

Logische Kopplung zwischen Genetikmaschine und Netzsimulator

Eine etwas andere Perspektive nimmt Abbildung 3.10 ein. Hier soll herausgearbeitet werden, dass die logische Kopplung von Genetikmaschine und Netzsimulator ausschließlich über die Fitnessfunktion geschieht.

Die logische Kopplung gestaltet sich wie folgt:

1. Eine Generation n liegt in Form von Genotypen vor.
2. Zur Bewertung der Individuen in Generation n wird die Fitnessfunktion aufgerufen.
   a) Die Genotypen werden in Phänotypen transformiert.
   b) Die Phänotypen werden dem Netzsimulator übergeben.
   c) Der Netzsimulator trainiert und validiert die Phänotypen.
   d) Die Ergebnisse der Validierung werden an die Fitnessfunktion zurückgegeben.
   e) Auf Basis der Ergebnisse berechnet die Fitnessfunktion die Fitness der Individuen.
3. Die besten Netze werden selektiert.
4. Auf Basis der selektierten Netze wird Generation $n + 1$ erzeugt.

Designentscheidungen

Wie bereits in Kapitel 3.1 auf Seite 35 erwähnt, ist das eigentliche Trainieren der Neuronalen Netze der mit Abstand rechenaufwändigste Teil des Verfahrens. Auf Grund dessen ist es notwendig, dass der Simulator zum Berechnen dieser Netze effizient arbeitet. Es wird daher ein vorhandenes Produkt verwendet, dessen Effizienz erprobt ist.

Die Genetikmaschine ist eine Eigenentwicklung, die entsprechend dem Rahmen dieser Arbeit einfach gehalten wurde. Wegen ihrer Einfachheit der benötigten Genetikmaschine ist von der Suche und Einarbeitung in ein vorhandenes Produkt abgesehen worden.

Der Kern der Softwareentwicklung für diese Arbeit ist also eine genetisch getriebene Steuereinheit für die Berechnung Neuronaler Netze.

Bereits bei den Vorstudien zu dieser Arbeit wurde klar, dass es eine ganze Reihe von Einstellungsmöglichkeiten für die Berechnung von Neuronalen Netzen gibt, die sich für eine Genetische Optimierung eignen. Diese betreffen zum einen so grobgranulare Teile wie z.B. die Netztopologie und die Art der Vernetzung der Neuronen untereinander und zum anderen solche feingranulare Elemente wie die Einstellung der Schrittweite in einem Backpropagation Algorithmus.

Zu den Teilen, die auf grobgranularer Ebene durch das Verfahren optimiert werden, findet sich mehr unter Kapitel 3.1 ab Seite 32. Die Einzelparameter respektive deren Rahmen für die Optimierung werden im Folgenden beschrieben.

Netzparameter

Einstellbare Netzparameter

Rahmensetzung für die Gewichte bei der Netzinitialisierung

1. Gewichtsminimum
2. Gewichtsmaximum

Das Gewichtsminimum liegt klassischer Weise zwischen -1 und 0 während das Gewichtsmaximum zwischen 0 und 1 liegt. Beide Parameter werden nicht optimiert, sondern bleiben immer gleich.

Größe von Input und Output Layer Die Größe von Input und Output Layer sind durch die zu klassifizierenden Muster festgelegt. Diese Einstellung ist notwendig, um das Verfahren an ein spezielles Muster anzupassen.

Rahmensetzung für die Topologie des Hidden Layers Die Topologie des Neuronalen Netzes ist eine der wichtigsten Stellgrößen in Neuronalen Netzen. Mit den hier genannten Parametern werden Rahmenbedingungen für den Zufallsgenerator bei der initialen Genotyperzeugung und bei der Mutation der Neuronalen Netze gesetzt.

Die einzelnen Parameter sind:

- Anzahl der Hidden Layer

  Die Anzahl der Hidden Layer kann im vorgestellten Verfahren jeden Wert annehmen, der größer als 0 ist. Dieser Parameter unterliegt als einziger hier aufgeführter Parameter nicht der Mutation. Wenn also eine Gruppe von Individuen, die sich durch die Anzahl der Hidden Layer von den übrigen Individuen einer Generation unterscheidet, „ausstirbt“, kann kein neues Individuum mit dieser Anzahl von Hidden Layern entstehen. Die Anzahl der Hidden Layer hat starke Auswirkungen auf die Geschwindigkeit des Netztrainings.

- Maximale Anzahl der Neuronen pro Hidden Layer

  Eine Beschränkung des Suchraums für das genetische Verfahren macht eine Begrenzung unerlässlich. Außerdem ist zu erwarten, dass bestimmte Netzgrößen keinen Sinn machen. Die Generalisierungsfähigkeit bei steigender Netzgröße nimmt tendenziell ab[28].

  Die Anzahl der Neuronen pro Hidden Layer hat außerdem starke Auswirkungen auf die Geschwindigkeit des Netztrainings.

- Minimale Anzahl der Neuronen pro Hidden Layer

  Das vom Neuronalen Netz „Gelernte“ wird in den Gewichten zwischen den Neuronen abgelegt. Abhängig von der Komplexität des zu Lernenden, muss also eine Mindestanzahl von Neuronen vorhanden sein, um das Gelernte speichern zu können[29].

Fehlervariante In dieser Arbeit findet nur der *aufsummierte quadratische Fehler* - im Folgenden auch *SSE* oder nur *Fehlerquadrat* genannt - Verwendung. Da andere Fehlervarianten möglicherweise noch interessant werden könnten, werden diese ebenfalls für das Verfahren wählbar sein.

- SSE (sum of squared errors)

  Summe der Fehlerquadrate aller Output Neuronen.

- MSE (mean square error)

$$MSE = \frac{SSE}{AnzahlderTrainingsmuster} \tag{3.9}$$

- SSEPU (sum of squared errors per Unit)

$$SSEPU = \frac{SSE}{AnzahlderOutputneuronen} \tag{3.10}$$

Destination Error Dieser Wert legt eine der zwei möglichen Abbruchbedingungen fest. Sobald das Training eines Netzes in einer Generation diesen Wert erreicht, bricht das Training für dieses Netz ab. Die restlichen Individuen der Generation werden noch weiter trainiert. Nach diesem Training wird das Verfahren beendet.

[28] Siehe hierzu Lämmel und Cleve (2001)

[29] Außerdem führten bei den Voruntersuchungen solche Netze zu Fehlern, deren Hidden Layer nur ein Neuron enthielt. Siehe hierzu auch Kapitel 4.4 Seite 69.

Parameter zur Laufzeitoptimierung Ziel des vorgestellten Verfahrens ist, einen Simulator für Neuronale Netze beim Finden solcher Netze zu unterstützen, die ein bestimmtes Fehlerquadrat erreichen oder sogar darunter bleiben. Dieses „Ziel“ wird im *Destination Error* festgeschrieben.

Nicht alle Netze können diesen Wert erreichen. Dadurch besteht die Gefahr, dass das Verfahren nicht terminiert.

Abhilfe kann geschaffen werden durch:

- einen SSE, der in der Anfangsphase des Verfahrens größer ist als in der Endphase und
- eine maximale Anzahl von Lernyklen, die ein Netz zum Training erhält.

Dazu dienen der *Error Threshold* und der *Beginning Cycle Threshold*. Beide werden wie folgt verwendet:

- Error Threshold

  Die erste Belegung für den *Error Threshold* muss von Hand eingegeben werden. Danach wird der Wert nach folgender Formel in jeder Generation reduziert:

$$ET^{30}{}_{neu} = ET_{alt} * (1 - SoD^{31}{}^{32}) \tag{3.11}$$

- Beginning Cycle Threshold

  Gibt an, wieviele Trainingszyklen in der entsprechenden Generation maximal pro Individuum zur Verfügung stehen.

  Dieser Wert variiert in seiner Höhe und hängt davon ab, wieviele Zyklen das schlechteste der selektierten Individuen der Vorgängergeneration brauchte.

  Für die erste Generation muss dieser Wert von Hand bestimmt werden.

Kleinstmöglicher Learningfunction Parameterwert Die Parameter der Lernfunktionen sollten nicht den Wert 0 oder gar einen kleineren Wert annehmen. Deswegen ist hier eine Begrenzung notwendig.

[30] Error Threshold
[31] Speed of Decline
[32] Siehe hierzu Kapitel 3.2 auf Seite 55

Formatangabe für die Verbindungsgewichte im Neuronalen Netz Dieser Parameter dient im Wesentlichen der besseren Lesbarkeit der Neuronalen Netze, die an den entsprechenden Netzsimulator übergeben werden.

Begrenzung der Größe der Hidden Layer in der Ausgangspopulation Wie bereits erwähnt, gibt es gute Gründe, die Größe der Hidden Layer zu beschränken[33]. Da in der ersten Generation ein sehr hoher Anteil an ungeeigneten Individuen vorhanden ist, kann es sinnvoll sein, zu Beginn kleinere Netze als im späteren Verlauf zuzulassen.

Es besteht die Gefahr, dass der Optimierungsprozess bei der Wahl eines zu kleinen Wertes beeinträchtigt wird. Dieses Risiko tritt besonders dann auf, wenn die optimale Netzkonfiguration eine Netztopologie mit deutlich größeren Hidden Layern bräuchte, als durch diesen Parameter erlaubt wird.

Faktor zur Erweiterung der erlaubten Lernzyklen bei zunehmender Anzahl von Generationen Nach jedem Selektionsprozess wird überprüft, wieviele Lernzyklen das schlechteste der Gewinnernetze verwendet hat. Die so ermittelte Zahl wird mit dem hier genannten Faktor multipliziert und somit die Anzahl der erlaubten Lernzyklen für die Folgegeneration ermittelt.

Unverstellbare Netzparameter

Sowohl die Zusammensetzung der Lernfunktionen als auch die Menge der Aktivierungsfunktionen lassen sich nicht verändern[34].

Lernfunktionen [35]

1. Standard Backpropagation
2. Backpropagation Momentum
3. Quickpropagation
4. Resilient Propagation

[33] Siehe hierzu Kapitel 3.2 auf Seite 50

[34] Außer man verändert den Sourcecode des Programms an den entsprechenden Stellen. In der grafischen Benutzeroberfläche ist dies nicht möglich.

[35] Siehe hierzu Kapitel 3.1 auf Seite 40

Aktivierungsfunktionen [36]

1. Logistische Funktion
2. Signum-Funktion
3. Tangens-Hyperbolicus-Funktion
4. Schrittfunktion
5. Produktfunktion
6. Identitätsfunktion
7. Sinusfunktion
8. Exponentialfunktion

Genetikparameter

Einstellbare Genetikparameter

Generationsgröße Es gibt im vorgestellten Verfahren zwei Vorbedingungen an unterschiedlichen Stellen im Algorithmus:

- Für die Fortpflanzung durch Rekombination[37], wie sie im vorgestellten Verfahren vorgenommen wird, ist es notwendig, dass die Anzahl der selektierten Individuen ganzzahlig durch zwei teilbar ist.
- Die drei Prozesse zur Generierung der Folgegeneration[38] tragen zu gleichen Teilen zu dieser bei. Auf Grund dessen muss die Generationsgröße ganzzahlig durch drei teilbar sein.

Daraus ergibt sich als Vorbedingung für das ganze Verfahren eine ganzzahlig durch sechs teilbare Generationsgröße.

Für das Verfahren vorgesehen sind die Belegungen: 6, 12, 18, 24, 36, 48.

[36] Siehe hierzu Kapitel 3.1 auf Seite 45
[37] Rekombination3.137
[38] Rekombination, Mutation und die Übernahme der Gewinnernetze aus der Vorgängergeneration

Anzahl der maximal zu durchlaufenden Generationszyklen Es ist möglich, dass der gewünschte Destination Error vom Verfahren nicht erzielt werden kann. Um dennoch einen kontrollierten Abbruch zu erreichen, ist es möglich, einen Höchstwert für die zulässigen Generationenzyklen setzen.

Ersetzungsschema Die Entscheidung, welche Individuen zur Erzeugung der Folgegeneration ausgewählt werden, bestimmt das Ersetzungsschema[39]. Auswählbare Ersetzungsschemata sind:

1. Elitism
2. Roulette Wheel
3. Elitism Roulette Wheel

Speed of Decline Mit Hilfe dieser Stellgröße kann direkt beeinflusst werden, in welchem Maße sich der *Error Threshold*[40] mit jeder Generation verringert.

[39] Siehe Kapitel 2.2 Seite 23
[40] Seihe auch Kapitel 3.2 Seite52

4. Versuchsaufbau

4.1. Versuchsbedingungen

Plattform

Alle Versuche werden auf einem handelsüblichen I86-Computer (AMD Athlon(TM) XP 2400+ / 1GB RAM) durchgeführt. Das zu Grunde liegende Betriebssystem ist Windows 2000 mit Service Pack 4. Als Java Runtime Environment steht Java 1.5 zur Verfügung. Weitere externe Java-Bibliotheken sind JDOM und JAI (Java Advanced Imaging). Entwickelt wurde mit Eclipse 3.0. Als Netzsimulator wird für einzelne Trainingsläufe der JavaSNNS benutzt, während der Genetische Algorithmus mit der dazugehörigen Batchauskopplung Batchman des SNNS (Stuttgarter Neuronale Netze Simulator) arbeitet.

Ein System auf Linux oder Mac als Plattform wäre ebenso gut geeignet. Auch die Verwendung von Java als Sprache für die Bildvorverarbeitung ist keineswegs zwingend. Ausschlaggebend waren hier die techniche Ausstattung und die persönlichen Präferenzen des Entwicklers. Der Performanceaspekt, dem durch eine andere Programmiersprache wie zum Beispiel C++ sicher mehr Rechnung getragen worden wäre, war für den programmierbaren Teil vernachlässigbar, da die rechenintensiven Teile des Genetischen Algorithmus, nämlich das Trainieren der Neuronalen Netze, ohnehin vom Netzsimulator übernommen wird.

Testdatensätze

Zahlen

Die Basis des Versuchs zur Ziffernerkennung bilden 2090 Zahlen von 0 - 9 im *.pat-Datenformat. Zum Training dienen davon 1500 Muster. Zur Validierung werden 590 Muster verwendet.

Blätter

Für das andere Teilprojekt dieser Arbeit stehen 1249 Blätter verschiedener Bäume aus dem Volkspark Friedrichshain in Berlin zur Verfügung. Davon sind 154 Ahorn-, 282 Birken-, 264 Eichen-, 211 Haselnuss-, 170 Linden-, 168 Platanenblätter. Diese Blätter wurden mit einem handelsüblichen Scanner bei 100 dpi im 256-bit-Graustufenmodus eingescannt. Die eingescannten Daten wurden in Photoshop 7.0 als JPG-Format abgespeichert. Die Grauwerte der Bilder wurden mit Hilfe selbstgeschriebener Klassen auf Werte zwischen 0 und 1 normiert und in das vom SNNS vorgeschriebene *.pat-Datenformat umgewandelt und zusammengefasst. Von den 1249 Blattmustern werden 940 für das Training verwendet und 309 dienen der Validierung.

4.2. SNNS/JavaNNS/Batchman

Was ist SNNS/JNNS/Batchman

Der Java Neuronale Netzwerk Simulator (JavaNNS) ist eine Software zur Simulation von Neuronalen Netzen. Der JavaNNS wurde am Wilhelm-Schickard-Institute in Tübingen entwickelt und basiert auf dem Kernel des Stuttgarter Neuronale Netzwerk Simulators (SNNS). Die dazugehörige Software Batchman ermöglicht die Batchverarbeitung von mehreren Neuronalen Netzen.

Netztopologie

Zur Erstellung der gewünschten Netztopologie stehen im JavaNNS zwei Werkzeuge bereit, die es erlauben, layerweise Neuronen anzulegen, diesen Rollen zuzuweisen[1] und die Schichten dann in verschiedenen Varianten zu vernetzen. Die so erzeugten Netze lassen sich speichern und gegebenenfalls wieder laden oder aber vom Batchman verwenden.

Dem Batchman muss die Netztopologie übergeben werden. Dies kann, wie eben beschrieben, ein im JavaNNS oder SNNS konfiguriertes Netz sein oder aber ein oder mehrere Textdateien, die entsprechende Netzbeschreibungen im vorgeschriebenen Format enthalten.

[1] Für die Arbeit sind nur die Rollen Input Layer, Output Layer und Hidden Layer von Bedeutung

Dateneingang

Die Eingabedaten für den Batchman und den JavaNNS stellen sogenannte Patterndateien dar, deren Dateiendung *.pat* lautet. Die Dateien bestehen aus zwei Teilen:

1. Am Anfang steht immer eine dreizeilige Metainformation zu der Patterndatei[2]. Diese enthält:
   a) Die Anzahl der in der Datei vorhandenen Muster.
   b) Die Anzahl der Eingangsneuronen
   c) Die Anzahl der Ausgangsneuronen
2. Die darauf folgenden Muster[3] bestehen jeweils aus zwei Teilen:
   a) Dem eigenlichen Muster, das in Form von Zahlwerten für jedes Eingangsneuron vorliegt.
   b) Der Kategorie, der dieses Muster zugeordnet ist. Diese Kategorie liegt ebenfalls in Form von Zahlenwerten vor.

```
No. of patterns : 590
No. of input units : 160
No. of output units : 10
```

Abbildung 4.1.: Metainformationen einer JavaNNS-Patterndateien-Datei

Dem durch JavaNNS/Batchman vorgegebenen Patternformat ist es geschuldet, dass zu klassifizierende Muster vorverarbeitet werden müssen. Diese Vorverarbeitung wird in Kapitel 4.1 auf Seite 57 kurz angedeutet.

Im besprochenen Verfahren werden JavaNNS und Batchman pro Training zwei Patterndateien übergeben. Beide Muster müssen im gleichen Format vorliegen. Eine dieser Dateien wird zum eigentlichen Netztraining benutzt, während die andere der Validierung der Trainingsergebnisse dient.

[2] Siehe dazu Abbildung 4.1
[3] Siehe dazu Abbildung 4.2

```
0 0 0 0 0 1 1 1 0 0
0 0 0 1 1 1 1 1 1 1
0 0 1 1 1 1 1 1 1 1
0 1 1 1 1 0 0 1 1 1
0 1 1 0 0 0 0 0 1 1
0 1 1 0 0 0 0 1 1 1
1 1 1 0 0 0 0 1 1 1
1 1 1 0 0 0 0 0 1 1
1 1 1 0 0 0 0 0 1 1
1 1 1 0 0 0 0 0 1 1
1 1 1 0 0 0 0 1 1 0
1 1 1 0 0 0 1 1 1 0
1 1 1 1 1 1 1 1 1 0
0 1 1 1 1 1 1 1 0 0
0 1 1 1 1 1 1 0 0 0
0 0 1 1 1 1 0 0 0 0
#result
1 0 0 0 0 0 0 0 0 0
```

Abbildung 4.2.: Einzelnes Muster aus einer JavaNNS-Patterndateien-Datei

Bei der Erstellung beider Patterndateien ist Folgendes zu beachten:

- In einer Datei vorkommende Muster sollten nicht in der jeweils anderen Datei vorkommen.
- Beide Dateien sollten gleich gute Muster für die zu lernenden Kategorien beinhalten.
- In die Validierungsdatei sollten etwa 10-30% [4] der vorhandenen Muster fließen, der Rest geht in die Trainingsdatei ein.

[4] Siehe Meisel (2004).

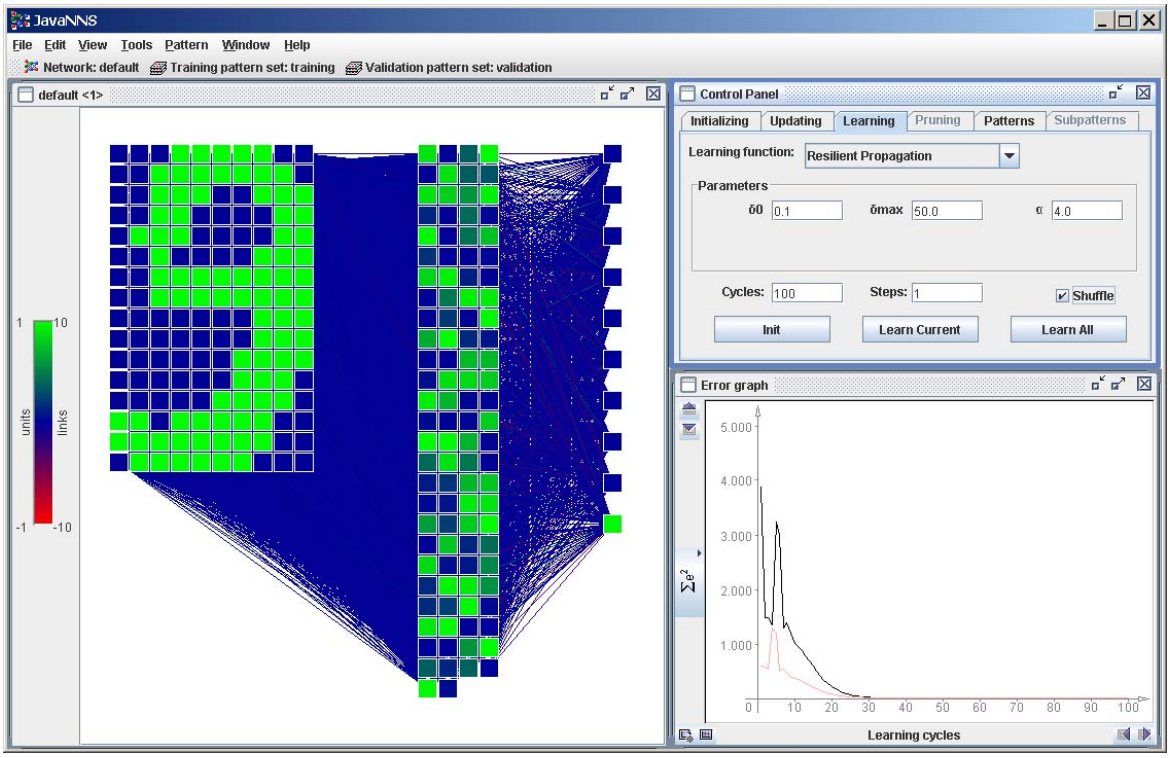

Abbildung 4.3.: JavaNNS GUI mit Neuronalem Netz, Control Panel und Fehlergraph

Programmsteuerung

Die Steuerung des JavaNNS geschieht über das Control Panel[5]. Das Control Panel bietet eine reichhaltige Zahl an Parametern, mit denen das Lernen des Neuronalen Netzes beeinflusst werden kann. Einige für die Arbeit relevante Stellgrößen werden hier aufgezählt:

- Minimal- und Maximalwert der Verbindungsgewichte bei der zufälligen Netzinitialisierung.
- Lernfunktionen mit den dazugehörigen Parametern.
- Anzahl der Lernzyklen.
- Die Möglichkeit, die Muster für die Lernvorgänge zu mischen.
- Rollenbestimmung von jeweils einer Patterndatei zum Trainings- oder zum Validierungspattern.

[5]Siehe Abbildung 4.3

Der Batchman wird im Rahmen dieser Arbeit mit Hilfe einer sog. *.bat-Datei gesteuert. Über diese Datei können dem Batchman alle Informationen übergeben werden, die der JavaNNS in der GUI über das Control Panel erhält. Darüber hinaus verfügt der Batchman aber noch über eine Reihe von Funktionen, mit Hilfe derer die Batchverarbeitung von Neuronalen Netzen ermöglicht wird. Diese werden dem Batchman ebenso mit Hilfe der *.bat-Datei übergeben.

Datenausgang

JavaNNS/Batchman ermöglicht es, ein trainiertes Netz[6] zu exportieren und als Klassifikator in eine Applikation einzubauen Da diese Funktion für vorliegende Arbeit nicht genutzt wird, wird nicht weiter darauf eingegangen.

Die für diese Arbeit bedeutsamste Ausgabe des JavaNNS ist die grafische Darstellung des aufsummierten Fehlerquadrates[7] des Neuronalen Netzes im Fehlergraphen[8]. Das Control Panel bietet darüber hinaus die Möglichkeit, durch die einzelnen Muster „durchzusteppen“. Auf diese Weise kann die Netzantwort auf ein bestimmtes Eingabemuster überprüft werden, was eine Kontrolle der genetisch optimierten Netze ermöglicht.

Die Ausgabe des Batchman ist rein textuell. Sie kann auf einer Textkonsole erfolgen oder in eine Textdatei geleitet werden. Im vorgestellten Verfahren wird die Ausgabe über die Textdatei genutzt. Die entsprechende Datei ist die *logfile*-Datei[9].

Nutzung

Die von der Software erzeugten *.net-Dateien[10] werden zunächst in ein vorgegebenes Verzeichnis geschrieben.

[6] Als c-File.
[7] SSE
[8] Auf Abbildung 4.3 als *Error graph* bezeichnet
[9] Ohne Datei-Endung
[10] Pro Individuum der Population eine.

Für diese Arbeit wird hauptsächlich der Batchman verwendet. Die von der Software erzeugten *.net-Dateien[11] werden zunächst in ein vorgegebenes Verzeichnis geschrieben. Danach werden die Anweisungen zum Training der Netze in der batchjob.bat im gleichen Verzeichnis abgelegt. Anweisungen über die gewünschte Ausgabe werden ebenfalls in der batchjob.bat gespeichert. Nun wird Batchman aus der Applikation über einen Systemaufruf synchron gestartet. Nach jedem für ein Netz abgeschlossenen Trainingsvorgang schreibt Batchman die Netznummer[12], die benötigte Anzahl der Lernzyklen und die summierten quadratischen Fehler für den jeweils letzten Validierungszyklus in die logfile-Datei. Nachdem alle Trainings abgeschlossen sind, entnimmt die Applikation der logfile-Datei die benötigten Daten für die Fitness-Berechnung der einzelnen Individuen.

Der JavaNNS wird in dieser Arbeit zur Berechnung der Referenznetze und zur Qualitätskontrolle herangezogen. Außerdem werden die in Kapitel 2.3 erwähnten Standardparameter den Grundeinstellungen des JavaNNS pro Lernfunktion entnommen.

4.3. Beschreibung der erstellten Software

Eine wichtige Rolle in der erstellten Software spielt Kommunikation. Diese findet zwischen den beiden Hauptkomponenten - Genetikmaschine und Batchman - sowie zwischen der Software und ihrem Bediener statt.

Der Austausch zwischen den Softwarekomponenten geschieht ausschließlich über Dateien. Die Genetikmaschine erzeugt die zu trainierenden Netze in Form von **.net-Dateien* und eine weitere Datei mit dem Namen *batchjob.bat*. Diese enthält die Steueranweisungen, die der Batchman zum Trainieren der Neuronalen Netze benötigt. Der Batchman schreibt die Trainingsergebnisse in die *logfile-Datei*, die von der Genetikmaschine im Rahmen der Fitnessbewertung gelesen wird.

Die Mensch-Maschine-Kommunikation geschieht im Wesentlichen über eine grafische Benutzeroberfläche. Mit Ihrer Hilfe kann der Benutzer vor dem Training verschiedene Parameter einstellen und somit die Rahmenbedingungen des genetischen Optimierers und der Neuronalen Netze verändern. Die Protokollierung eines aktuellen Laufes kann ebenfalls in der grafischen Benutzeroberfläche eingesehen werden.

Alle verwendeten Dateien müssen sich in einem Verzeichnis befinden, das dem Verfahren bei Programmstart übergeben wird.

[11] Pro Individuum der Population eine.
[12] Nullbasiert

Benötigte Dateien

Folgende Dateien müssen zu Beginn des Verfahrens bereit stehen:

- persistence.xml

  Die *persistence.xml* ist für die Datenspeicherung der Steuerdaten für den genetischen Optimierer zuständig. Was über einen Lauf hinaus an Daten benötigt wird, wird hier abgelegt. Dies betrifft die Rahmenbedingungen des genetischen Optimierers und der Neuronalen Netze sowie Pfad- und Dateinamen.

- batchman.exe

  Zur Berechnung der Neuronalen Netze dient die *batchman.exe*. Dies ist die bereits erwähnte Batchauskopplung des SNNS, die in einem Paket mit dem JavaNNS ausgeliefert wird.

- analyze.exe

  Zur *batchman.exe* gehört zur Analyse der Neuronalen Netze notwendig die *analyze.exe* dazu. Diese wird ausschließlich von der *batchman.exe* angesprochen.

- training.PAT

  Die *training.PAT* enthält die Muster zum Training des Neuronalen Netzes.

- validation.PAT

  Zur Überprüfung der Generalisierung des aus der *training.PAT* gelernten „Wissens“, verlangt das Verfahren das Vorhandensein der *validation.PAT*, die im gleichen Format vorliegen muss wie die *training.PAT*. Wenn keine Validierungsdaten vorliegen, sollte eine Kopie der *training.PAT* angefertigt werden und in *validation.PAT* umbenannt werden.

Erzeugte Dateien

Die folgenden Dateien werden im Verlauf des Verfahrens erzeugt:

- batchjob.bat

  Zur Steuerung des Batchman erzeugt die Software für das Training und die Validierung von jeder Generation die *batchjob.bat*.

- logfile

  Als Datenausgang des Batchman dient die *logfile-Datei*. Sie ist gleichzeitig der Dateneingang für die Fitnessfunktion der genetischen Komponente.

- netX.net

  Die im Verfahren erzeugten Netze werden als *netX.net* herausgeschrieben. Das *X* steht hier für eine Nummer zwischen 1 und der Generationsgröße des aktuellen Laufs.

- tracking.txt

  In der *tracking.txt* werden einige wichtige Netzdaten für jede Generation abgelegt. Dies ermöglicht dem Benutzer der Software später einen Überblick über den Verlauf des Verfahrens.

Grafische Benutzeroberfläche

Startfenster

Hier wird das Verzeichnis angegeben, in dem sich die im Verfahren verwendeten Dateien befinden. Bei einem Klick auf den Startknopf wird das Hauptfenster geöffnet.

Ein Screenshot des Startfensters befindet sich im Anhang auf Seite 107.

Hauptfenster

Das Hauptfenster besteht aus vier Reitern, die ihrerseits Unterfenster enthalten:

- Kontrollfenster
- Genetikfenster
- Netzwerkfenster
- Persistenzfenster

Kontrollfenster

Hier wird das Verfahren gestartet und bei Bedarf gestoppt. Der Stopp greift erst am Ende eines Generationszyklus. Die Hauptfunktion des Kontrollfensters besteht darin, dem Anwender einen Überblick über den bisherigen Verlauf des Verfahrens zu geben.

Dargestellt werden:

- Im Header jeder Generation:
  - Generationsnummer
  - Error Threshold der jeweils aktuellen Generation
  - Die bisher verbrauchte Gesamtzeit
  - Das arithmetische Mittel der selektierten Individuen
- Für die Individuen innerhalb der Generation:
  - Nummer des Individuums
  - Netztopologie
  - Aktivierungsfunktionen für die einzelnen Layer
  - Lernverfahren
  - Die zum Lernverfahren gehörigen Parameterbelegungen
  - Die verwendete Anzahl von Lernzyklen
  - Fitness
  - SSE

Ein Screenshot des Kontrollfensters befindet sich im Anhang auf Seite 108.

Genetikfenster

Einstellbare Größen:

- Wählbar mit Combobox
  - Generationsgröße
  - Selektionsgröße[13]
  - Ersetzungsschema
- Frei editierbare Textfelder
  - Generationszyklen (Ganzzahlige Werte größer Null)
  - Mutationsrate (Gleitkommawerte)
  - Fehlerabstieg (Gleitkommawerte)

Veränderte Daten können über den Saveknopf in die *persistence.xml* abgespeichert werden.

Ein Screenshot des Genetikfensters befindet sich im Anhang auf Seite 109.

[13] Diese ergibt sich aus der Generationsgröße geteilt durch drei und ist nur über diese zu verändern.

Netzwerkfenster

Einstellbare Größen:

- Frei editierbare Textfelder:
  - Minimalgewicht bei der Netzinitialisierung (Gleitkommawerte)
  - Maximalgewicht bei der Netzinitialisierung (Gleitkommawerte)
  - Größe des Input Layers (Ganzzahlige Werte)
  - Minimale Anzahl der Neuronen pro Hidden Layer (Ganzzahlige Werte)
  - Maximale Anzahl der Neuronen pro Hidden Layer (Ganzzahlige Werte)
  - Minimale Anzahl der Hidden Layer (Ganzzahlige Werte)
  - Maximale Anzahl der Hidden Layer (Ganzzahlige Werte)
  - Größe des Output Layers (Ganzzahlige Werte)
  - Error Threshold (Gleitkommawerte)
  - Beginning Cycle Threshold (Gleitkommawerte)
  - Kleinstmöglicher Lernfunktionsparameterwert (Gleitkommawerte)
  - Formatangabe für die Verbindungsgewichte im Neuronalen Netz
  - Begrenzung der Größe der Hidden Layer in der Ausgangspopulation (Ganzzahlige Werte)
  - Destination Error (Gleitkommawerte)
  - Fehlerabstieg (Gleitkommawerte)
- Wählbar mit Combobox
  - Fehlervariante

Nur zur Information und damit nicht editierbar:

- Lernfunktionen
  - Std_Backpropagation <-> Standard Backpropagation
  - BackpropMomentum <-> Backpropagation Momentum
  - Quickprop <-> Quickpropagation
  - Rprop <-> Resilient Propagation
- Aktivierungsfunktionen
  - Act_Logistic <-> Logistische Funktion
  - Act_Signum <-> Signum-Funktion
  - Act_TanH <-> Tangens-Hyperbolicus-Funktion
  - Act_StepFunc <-> Schrittfunktion
  - Act_Product <-> Produktfunktion
  - Act_Identity <-> Identitätsfunktion
  - Act_Sinus <-> Sinusfunktion
  - Act_Exponential <-> Exponentialfunktion

Veränderte Daten können über den Saveknopf in die *persistence.xml* abgespeichert werden.

Ein Screenshot des Netzwerkfensters befindet sich im Anhang auf Seite 110.

Persistenzfenster

Das Persistenzfenster dient der Anpassung an andere Testumgebungen sowie der Information über die Rolle einzelner Dateien. Alle Felder sind frei editierbar.

- Hauptverzeichnis der Applikation
- Name des Trainingspattern
- Name des Validierungspattern
- Name der Ergebnisdatei (findet keine Verwendung in dieser Arbeit)
- Name der Batchmansteuerdatei
- Name der Verlaufsdatei
- Präfix der Netzdateien
- Postfix der Netzdateien
- Name der Log-Datei des Batchman
- Präfix des Gewinnernetzes

Veränderte Daten können über den Saveknopf in die *persistence.xml* abgespeichert werden.

Ein Screenshot des Persistenzfensters befindet sich im Anhang auf Seite 111.

4.4. Fehler in der Ausgabe des Batchman und Fehlerbehebung

SSE = 0

Es kann vorkommen, dass der Batchman als Ergebnis der Netzvalidierung einen SSE von 0 ausgibt. Die Überprüfung der Netzqualität zeigt, dass in diesem Fall das Netz nichts gelernt hat. Der Wert des SSE stellt somit eine fehlerhafte Ausgabe des Batchman dar.

Ein SSE von 0 ohne tatsächlichen Lernerfolg kann nicht akzeptiert werden. Die Fitness solcher Netze wäre außerordentlich gut, das Ergebnis jedoch unbrauchbar.

Eine reproduzierbare Voraussetzung für dieses Verhalten ist eine zu geringe Anzahl von Neuronen in einem Hidden Layer. In anderen Fällen konnte die Ursache nicht eindeutig festgestellt werden.

Lösung (SSE = 0)

- Es wird ein Parameter eingeführt, der eine Mindestanzahl[14] von Neuronen pro Hidden Layer festlegt.
- Für den Fall, dass der SSE trotzdem bei 0 liegt, was auf weitere Fehler hinweist, die nicht reproduziert werden konnten, wird der SSE auf einen so hohen Wert gesetzt, dass sich das entsprechende Individuum nicht weiter fortpflanzen kann.

SSE = 1.#INF00 oder -1.#INF00

In einigen Fällen gibt der Batchman anstelle eines konkreten Wertes für den SSE die Werte „1.#INF00“ oder „-1.#INF00“ aus. Dieser Wert kann im weiteren Verlauf nicht verwendet werden.

Vermutlich liegt hier im Batchman eine Division durch 0 vor.

Lösung (SSE = 1.#INF00)

Der SSE des Individuums wird auf einen so hohen Wert gesetzt, dass sich das entsprechende Individuum nicht weiter fortpflanzen kann.

Anzahl der verwendeten Lernzyklen = 1

Es kann vorkommen, dass der Batchman als Ergebnis für die Anzahl der verwendeten Lernzyklen 1 ausgibt. Daraus ergeben sich 2 Probleme:

1. Der Logarithmus der verwendeten Lernzyklen fließt in die Fitnessfunktion ein. Der Logarithmus von 1 ergibt immer 0. Die Fitnessfunktion ist eine Multiplikation mit drei Faktoren. Wenn ein Faktor 0 ist, wird das Produkt ebenfalls 0. Dadurch dominiert dieser nur zu Optimierungszwecken gedachte Teilaspekt die gesamte Fitnessfunktion und birgt somit die Gefahr, dass der Genetische Algorithmus am eigentlichen Ziel vorbei optimiert.
2. Stichproben in Versuchsläufen ergeben, dass Netze, die für das Lernen nur einen einzigen Zyklus brauchen, trotz geringem SSE die Muster aus dem Validierungsset nicht erkennen.

[14] 16 (4x4) Neuronen. Dieser Wert ist willkürlich gewählt.

Lösung (Anzahl der verwendeten Lernzyklen = 1)

Für dieses Problem gibt es keine befriedigende Lösung. Es ist nicht ausgeschlossen, dass ein Neuronales Netz nach nur einem Lernzyklus sein Lernziel erreicht. Deshalb ist ein einfaches Verwerfen eines Netzes, das sein Ergebnis laut Batchman schon nach einem Lernzyklus erreicht, nicht ganz unproblematisch. Jedoch tut die Genetikmaschine genau das: Sie verwirft die entsprechenden Netze. Ausschlaggebend für diese Entscheidung war das zweite Problem. Dies trat häufig genug auf, um an dieser Stelle den möglichen Schaden durch das Problem höher zu bewerten, als die Chance nach einem Lernzyklus ein erfolgreiches Netz erhalten zu haben.

5. Versuchsdurchführung und Ergebnisse

5.1. Vorüberlegungen

Allgemeines

Für die Durchführung der Experimente werden beispielhaft ein Musterdatensatz mit Zahlen und ein Musterdatensatz mit Blättern verwendet. Auf jeden dieser Musterdatensätze wird das beschriebene Verfahren dreimal angewendet.

Das Experiment soll zeigen, inwiefern die Optimierung von Feedfoward-Netzen durch einen Genetischen Algorithmus für die Erkennung der untersuchten Musterobjekte sinnvoll ist. Kriterium bei dieser Bewertung ist der summierte quadratische Fehler (SSE) stellvertretend für die Erkennungsleistung der gefundenen Netze.

Die Einordnung der Ergebnisse wird anhand von Referenznetzen vorgenommen.

Vergleichbarkeit

Wie bereits erwähnt, gibt es kein mathematisch fundiertes Verfahren zur Erzeugung optimaler Neuronaler Netze.

Die Verwendung von Heuristiken zur Erzeugung von Vergleichsnetzen ist zwar theoretisch möglich, praktisch jedoch problematisch. Eine solche Heuristik müsste mindestens

- die Anzahl der Eingabemuster,
- die Komplexität der Eingabemuster und
- die Menge der zu lernenden Kategorien

berücksichtigen[1]. Nach derzeitigem Kenntnisstand gibt es keine solche Heuristik, die für die hier verwendeten Muster benutzbar wäre.

Alternativ bieten sich Referenznetze an, die wie folgt verwendet werden: Pro Muster werden vier „einfache“ Trainings mit dem JavaNNS durchgeführt. Jedem dieser Läufe wird eine andere Lernfunktion zugeordnet. Diese Lernfunktionen entsprechen den Funktionen, die auch bei der genetischen Optimierung Verwendung finden[2].

Der Aufbau der Vergleichsnetze ist wie folgt gestaltet:

- Die Größe des Input Layers wird durch die Gestalt der Eingangsmuster bestimmt.
- Es ist ein Hidden Layer vorgesehen.
- Die Größe des Hidden Layers beträgt zwei Drittel[3] der Neuronenzahl des Input Layers[4].
- Die Größe des Output Layers wird durch die Gestalt der Ausgangsmuster bestimmt.
- Aktivierungsfunktion ist für jedes Neuron die Act_Logistic [5].
- Die Zufallswerte bei der Initialisierung der Verbindungsgewichte liegen zwischen -0,1 und 0,1[6].
- Für alle anderen Parameter werden die im JavaNNS voreingestellten Standardwerte verwendet.

Rahmenbedingungen für das Training der Vergleichsnetze:

- Die Anzahl der Trainingszyklen beträgt 1000.
- Die Muster werden vor jedem Trainingszyklus gemischt.

[1] Ein Überblick über die Problematik findet sich unter Sarle (2004)
[2] Standard Backpropagation, Backpropagation Momentum, Resilient Propagation und Quickpropagation. Deren Beschreibung befindet sich in Kapitel 3.1 ab Seite 40
[3] Mathematisch gerundet.
[4] Diese Topologievorgabe erzeugt Netze, die nach Testerfahrung des Autors vergleichsweise niedrige Fehlerquadrate erzielen
[5] Siehe hierzu: Abbildung 2.4 auf Seite 19 und in Kapitel 3.1 auf Seite 45
[6] Der gewählte Wertebereich für die Zufallswerte geht auf eine Empfehlung von Andreas Meisel zurück, die sich in meinen Versuchen als gut herausgestellt hat

Abbruchkriterium

Standardabbruchbedingung für das genetische Verfahren ist das Erreichen eines gesetzten summierten Fehlerquadrates durch mindestens ein Individuum einer Generation. Die Restpopulation wird nach Erreichen dieser Abbruchbedingung noch weiter berechnet, da nicht auszuschließen ist, dass sich noch ein besseres Individuum findet.

Wie Voruntersuchungen zeigen, gibt es kein einfaches lineares Verhältnis zwischen der Größe des Input Layers, der Anzahl der Eingabemuster sowie der Größe des Output Layers und dem erreichbaren summierten Fehlerquadrat. Aus diesen drei Größen lässt sich also kein sinnvolles Abbruchkriterium ableiten.

Alternativ bietet sich an, die Ergebnisse[7] der oben beschriebenen Referenznetze zu verwenden. Diese Ergebnisse geben eine erste Idee dafür, wie weit das summierte Fehlerquadrat der gegebenen Netz-Muster-Konstellation absinken kann. Genutzt wird der Wert des besten Netzes. Dieser Wert wird halbiert und liefert so das gesuchte Abbruchkriterium - Destination Error - in Form eines angestrebten summierten Fehlerquadrates für das genetische Verfahren.

Ein zusätzliches Abbruchkriterium trägt der Zeitaufwändigkeit des Verfahrens Rechnung. Dies besteht in der Abarbeitung einer maximalen Anzahl von Generationszyklen.

Invariante Einstellungen für die Genetikmaschine

- Die maximale Generationsgröße wird auf 48 Individuen gesetzt.
- Die Selektionsgröße liegt bei 16 Individuen.
- Der Error Threshold nimmt zu Beginn den Wert des Destination Errors an.
- Der Speed of Decline liegt bei 0,0[8].
- Als Ersetzungsschema wird *Elitism* gewählt.
- Die Mutationsrate liegt bei 30%.
- Der Faktor zur Erweiterung der erlaubten Lernzyklen ist 1,2.

Die verwendeten Parameter haben sich bei den Voruntersuchungen als brauchbar erwiesen.

[7] Die summierten Fehlerquadrate

[8] Aus dieser und der Vorbedingung ergibt sich, dass das Verfahren immer gegen den gleichen Error Threshold arbeitet.

An dieser Stelle soll explizit darauf hingewiesen werden, dass in diesem Kapitel überwiegend über den summierten quadratischen Fehler geschrieben wird, obwohl dieser im verwendeten Verfahren nicht mit der Fitness gleichzusetzen ist. Tatsächlich geben die beiden anderen Parameter, die in die Fitnessfunktion aus Optimierungsgründen einfließen - Netzgröße und Anzahl der benötigten Lernzyklen - keinen Aufschluss über die Qualität des einzelnen Netzes im Sinne der Erkennungsleistung, um die es letztlich bei der Klassifikation geht.

Versuchsdokumentation

Die Dokumentation der Versuche gliedert sich wie folgt:

Zu Beginn werden pro Versuch die Ergebnisse der Referenznetze vorgestellt. Als nächstes folgen die Ergebnisse des Genetischen Algorithmus. Letztere spalten sich in zwei Teile auf:

- Eine Betrachtung der SSEs der Gewinnernetze pro Generation und Lauf.
- Eine Betrachtung der durchschnittlichen SSEs der selektierten Netze pro Generation und Lauf.

Hierin spiegeln sich zwei unterschiedliche Möglichkeiten zur Qualitätsbewertung einer Generation innerhalb eines genetischen Algorithmus wieder:

- Fitness des besten Individuums
- Fitness der zur Selektion zugelassenen Individuen

Am Ende der Versuche befindet sich eine Auswertung der Teilergebnisse.

5.2. Versuch 1: Klassifikation handgeschriebener Zeichen

Der für den ersten Versuch verwendete Datensatz besteht aus handgeschriebenen Zahlen von 0 - 9[9].

Vorliegende Musteranzahl:

- 1500 Trainingsmuster
- 590 Validierungsmuster

[9] Nähere Beschreibung in Kapitel 4.1 auf Seite 56

Versuch 1 Teil 1: Training mit dem JavaSNNS

Verwendete Netztopologie für Referenznetze:

- 160 Input Neuronen
- 106 Hidden Neuronen
- 10 Output Neuronen

Standard Backpropagation

Parameterbelegungen für Standard Backpropagation Referenznetz[10]:

- Step Width: 0,2
- max nonpropagated Error: 0,1

Bester erreichter SSE für die Validierungsfunktion: ca. 7,2[11].

Backpropagation Momentum

Parameterbelegungen für Backpropagation Momentum Referenznetz:

- Step Width: 0,2
- Momentum: 0,5
- Flat spot Elimination: 0,1
- max nonpropagated Error: 0,1

Bester erreichter SSE für die Validierungsfunktion: ca. 4,3[12].

[10] Für genauere Beschreibungen der Parameter der Lernfunktionen siehe Zell und andere (1998)

[11] Der entsprechende Fehlergraph befindet sich im Anhang B auf Seite B.1

[12] Der entsprechende Fehlergraph befindet sich im Anhang B auf Seite B.2. Dieser enthält nur 100 Zyklen. Nach dem die Trainingsfunktion einen SSE von 0 erreicht hat, sind keine Änderungen mehr zu erwarten. Mehr Zyklen würden dementsprechend keine weiteren Informationen bieten.

Quickpropagation

Parameterbelegungen für Quickpropagation Referenznetz:

- Step Width: 0,2
- Maximum groth factor: 2,0
- Weight decay: 1,0E-4
- max nonpropagated Error: 0,1

Bester erreichter SSE für die Validierungsfunktion: ca. 30[13].

Resilientpropagation

Parameterbelegungen für Resilientpropagation Referenznetz:

- δ min: 0,1
- δ max: 50,0
- Weight decay: 4,0

Bester erreichter SSE für die Validierungsfunktion: ca. 1,8[14].

Versuch 1 Teil 2: Trainingsläufe mit genetischer Optimierung

Rahmendaten für die Topologie der genetisch erzeugten Individuen:

- Größe des Input Layers: 160 Neuronen
- Ein Hidden Layer.
- Schwankungsbreite für die Größe des Hidden Layers liegt zwischen 16 und 80 Neuronen.
- Größe des Output Layers: 10 Neuronen.

Es werden maximal 12 Generationen durchgerechnet.

[13] Der entsprechende Fehlergraph befindet sich im Anhang B auf Seite B.3
[14] Der entsprechende Fehlergraph befindet sich im Anhang B auf Seite B.4

Im Folgenden werden die Ergebnisse dreier Läufe des vorgestellten Verfahrens anhand von Graphen und Tabellen dokumentiert. Abbildung 5.1 und Tabelle 5.2 zeigen die SSEs der jeweiligen Gewinnernetze pro Generation. Abbildung 5.3 und Tabelle 5.4 geben die durchschnittlichen SSEs der jeweils selektierten Netze pro Generation an.

Versuch 1 : Auswertung

Gewinnernetze

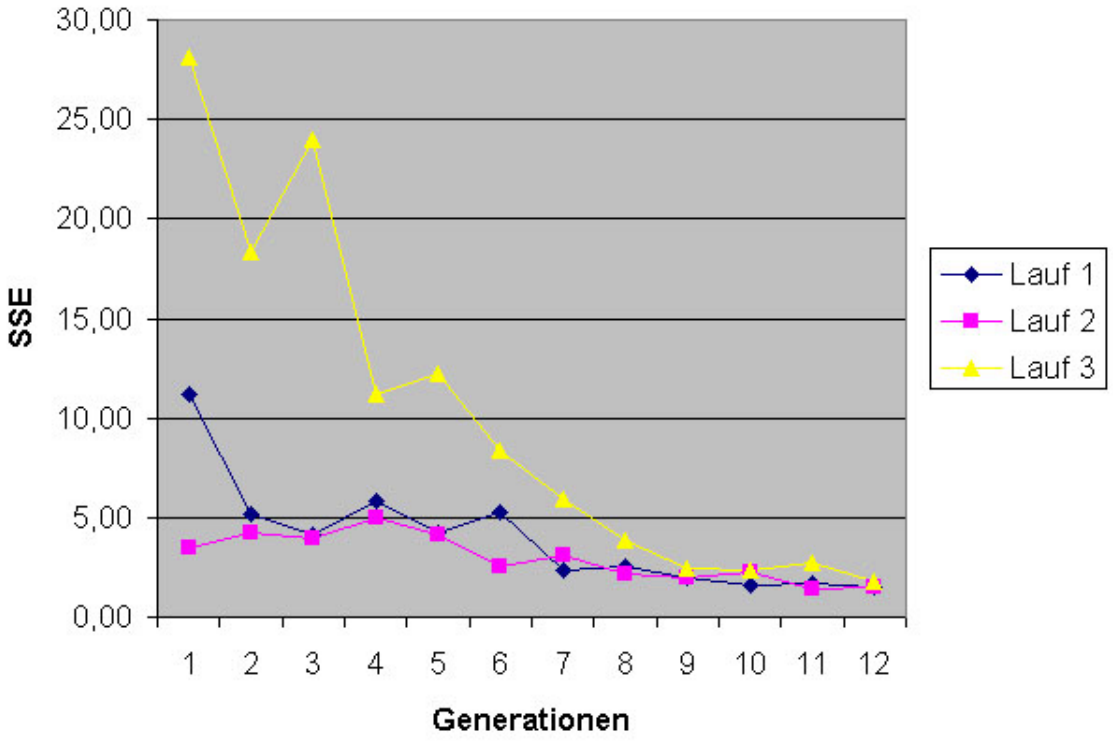

Abbildung 5.1.: Handgeschriebene Ziffern - Gewinnernetze

Die in Abbildung 5.1 und Tabelle 5.2 dokumentierten Läufe erreichen alle ein ähnliches Ziel, verlaufen dabei aber uneinheitlich. Während sich Lauf 3 über die Generationen hinweg deutlich gegenüber dem Ausgangswert verbessert, fällt die Verbesserung bei den Läufen 1 und 2 deutlich schwächer aus. Die Läufe 1 und 2 liefern allerdings bereits zu Beginn erheblich bessere Ergebnisse als Lauf 3.

Generation	SSE Lauf 1	SSE Lauf 2	SSE Lauf 3
1	11,20	3,49	28,07
2	5,18	4,24	18,33
3	4,11	3,96	23,97
4	5,81	5,02	11,23
5	4,22	4,16	12,26
6	5,29	2,53	8,38
7	2,34	3,10	5,96
8	2,51	2,17	3,83
9	1,95	1,95	2,42
10	1,57	2,28	2,39
11	1,74	1,40	2,71
12	1,53	1,46	1,83

Abbildung 5.2.: Handgeschriebene Ziffern - Gewinnernetze - Tabelle

Kein Lauf schafft es, das gesetzte Abbruchkriterium - ein summiertes Fehlerquadrat von 0,9 - zu erreichen. Das Ergebnis von Lauf 3 fällt sogar schlechter aus als das beste Ergebnis der Referenznetze.

Allen Läufen ist gemeinsam, dass sie sich stetig verbessern. Zwar verläuft keiner von ihnen monoton fallend, aber die Tendenz ist eindeutig.

Bemerkenswertes im Aufbau der drei letzten Gewinnernetze:

- Aktivierungsfunktion im Output Layer ist immer die Logistische Funktion
- Lernfunktion ist immer Resilient Propagation

Eine genaue Aufstellung der Konfiguration der Gewinnernetze befindet sich im Anhang A.

Durchschnitt der selektierten Netze

Die Verläufe der über die selektierten Netze gemittelten, summierten Fehlerquadrate in Abbildung 5.3 und Tabelle 5.4 zeigen sich einheitlicher als die Verläufe der Gewinnernetze. Sie fallen monoton.

Weiterhin geht aus Abbildung 5.3 hervor, dass alle drei Kurven abflachen. Dies sieht der Autor darin begründet, dass der Mustererkennung mit Neuronalen Netzen allgemein Grenzen gesetzt sind und das Verfahren bestenfalls gegen ein Fehlerquadrat von Null konvergieren kann. Somit können die Verringerungen bei bereits niedrigen Werten pro Generation nur noch gering ausfallen.

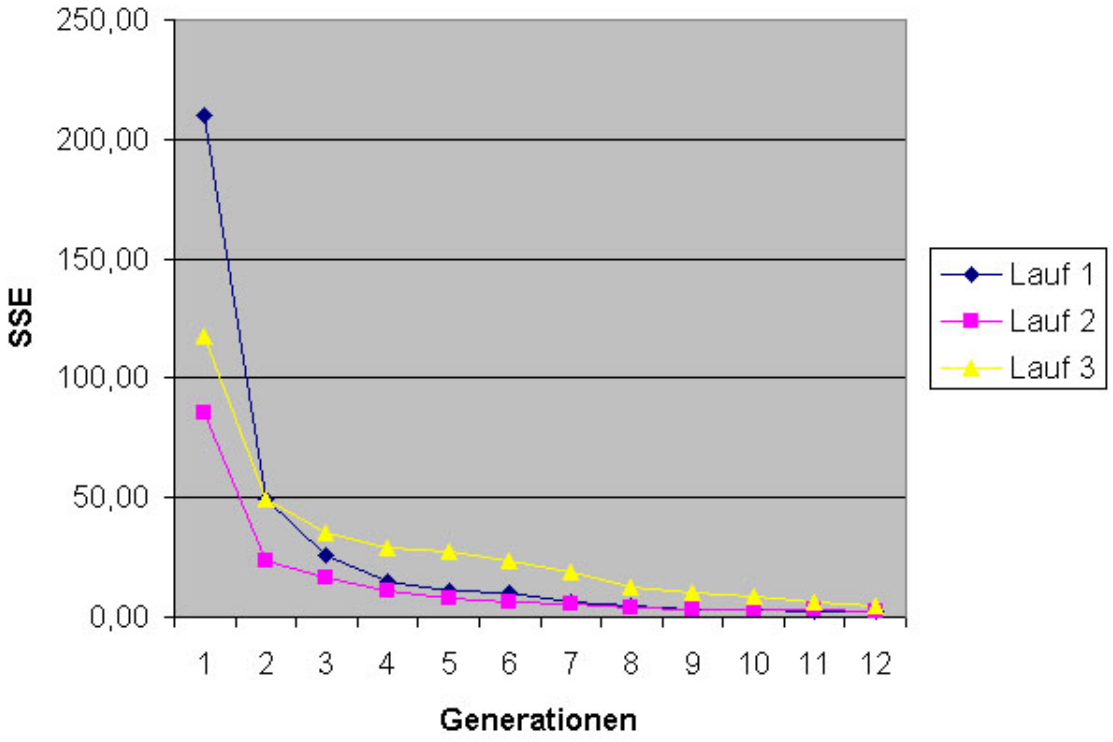

Abbildung 5.3.: Handgeschriebene Ziffern - Durchschnitt der selektierten Netze

Generation	SSE Lauf 1	SSE Lauf 2	SSE Lauf 3
1	210,11	85,48	117,52
2	49,70	23,14	49,10
3	25,89	16,62	35,16
4	14,69	11,36	29,22
5	11,22	7,78	27,79
6	9,87	6,08	23,15
7	5,99	5,27	18,71
8	4,74	3,96	12,89
9	3,41	3,36	10,21
10	3,12	3,33	8,99
11	2,57	3,23	6,30
12	2,26	2,44	4,76

Abbildung 5.4.: Handgeschriebene Ziffern - Durchschnitt der selektierten Netze - Tabelle

Bewertung

Die Ergebnisse von Lauf 3 waren schlechter als die des besten Referenznetzes. Dies wird vom Autor nicht als Scheitern des Verfahrens gesehen. Eine Kontrolle der tatsächlichen Bilderkennungsleistung ergab für das Gewinnernetz aus Lauf 3 einen hundertprozentigen Erfolg. D.h. es wurde nicht ein einziges Zahlenmuster falsch kategorisiert.

Es ist weiterhin festzuhalten, dass sich das schlechtere Abschneiden von Lauf 3 gegenüber dem besten Referenznetz lediglich auf die zweite Nachkommastelle beschränkt. Davon abgesehen unterbieten die beiden übrigen Läufe das Ergebnis des besten Referenznetzes, obwohl diesen eine deutlich geringere Anzahl an Lernzyklen als den Referensnetzen zur Verfügung stand. Dies ist ein deutlicher Hinweis darauf, dass das vorgestellte Verfahren in Versuch 1 funktioniert.

Der Verlauf der Kurven in Abbildung 5.3 gibt einen weiteren Hinweis für die Funktionstüchtigkeit des Verfahrens. Hier sieht man zunächst in Generation 1 wie unterschiedlich die SSEs der verschiedenen Läufe sind, obwohl allen die gleiche Anzahl Lernzyklen zur Verfügung steht. Dass die Kurven dennoch monoton fallen, schreibt der Autor in der Hauptsache der genetischen Optimierung zu.

5.3. Versuch 2: Klassifikation verschiedener Blattsorten

Der für den zweiten Versuch verwendete Datensatz besteht aus Blättern von sechs verschiedenen Baumarten[15].

Vorliegende Musteranzahl:

- 940 Trainingsmuster
- 309 Validierungsmuster

Versuch 2 Teil 1: Training mit dem JavaSNNS

Verwendete Netztopologie für Referenznetze:

- 625 Input Neuronen
- 417 Hidden Neuronen
- 6 Output Neuronen

[15] Nähere Beschreibung in Kapitel 4.1 auf Seite 57

Standard Backpropagation

Parameterbelegungen für Standard Backpropagation Referenznetz[16]:

- Step Width: 0,2
- max nonpropagated Error: 0,1

Bester erreichter SSE für die Validierungsfunktion: ca. 33[17].

Backpropagation Momentum

Parameterbelegungen für Backpropagation Momentum Referenznetz:

- Step Width: 0,2
- Momentum: 0,5
- Flat spot Elimination: 0,1
- max nonpropagated Error: 0,1

Bester erreichter SSE für die Validierungsfunktion: ca. 26[18].

Quickpropagation

Parameterbelegungen für Quickpropagation Referenznetz:

- Step Width: 0,2
- Maximum groth factor: 2,0
- Weight decay: 1,0E-4
- max nonpropagated Error: 0,1

Bester erreichter SSE für die Validierungsfunktion: ca. 80[19].

[16] Für genauere Beschreibungen der Parameter der Lernfunktionen siehe Zell und andere (1998)
[17] Der entsprechende Fehlergraph befindet sich im Anhang B auf Seite B.5
[18] Der entsprechende Fehlergraph befindet sich im Anhang B auf Seite B.6. Dieser enthält nur 100 Zyklen. Nach dem die Trainingsfunktion einen SSE von 0 erreicht hat, sind keine Änderungen mehr zu erwarten.
[19] Der entsprechende Fehlergraph befindet sich im Anhang B auf Seite B.7

Resilientpropagation

Parameterbelegungen für Resilientpropagation Referenznetz:

- δ min: 0,1
- δ max: 50,0
- Weight decay: 4,0

Bester erreichter SSE für die Validierungsfunktion: ca. 27[20].

Versuch 2 Teil 2: Trainingsläufe mit genetischer Optimierung

Rahmendaten für die Topologie der genetisch erzeugten Individuen:

- Größe des Input Layers: 625 Neuronen
- Ein Hidden Layer.
- Die Schwankungsbreite für die Größe des Hidden Layer liegt zwischen 16 und 125 Neuronen.
- Größe des Output Layers: 6 Neuronen.

Es werden maximal 6 Generationen durchgerechnet.

Abbildung 5.5 und Tabelle 5.6 zeigen die summierten Fehlerquadrate der jeweiligen Gewinnernetze pro Generation. Abbildung 5.7 und Tabelle 5.8 geben die durchschnittlichen, summierten Fehlerquadrate der jeweils selektierten Netze pro Generation an.

Versuch 2 : Auswertung

Gewinnernetze

Die in Abbildung 5.5 und Tabelle 5.6 dokumentierten Läufe verlaufen einander ähnlicher als die vergleichbaren Läufe in Versuch 1. Aufällig ist, dass Lauf 1 und 2 sich von Generation 5 auf 6 leicht verschlechtern, während sich Lauf 3 an dieser Stelle noch verbessern kann.

[20] Der entsprechende Fehlergraph befindet sich im Anhang B auf Seite B.8

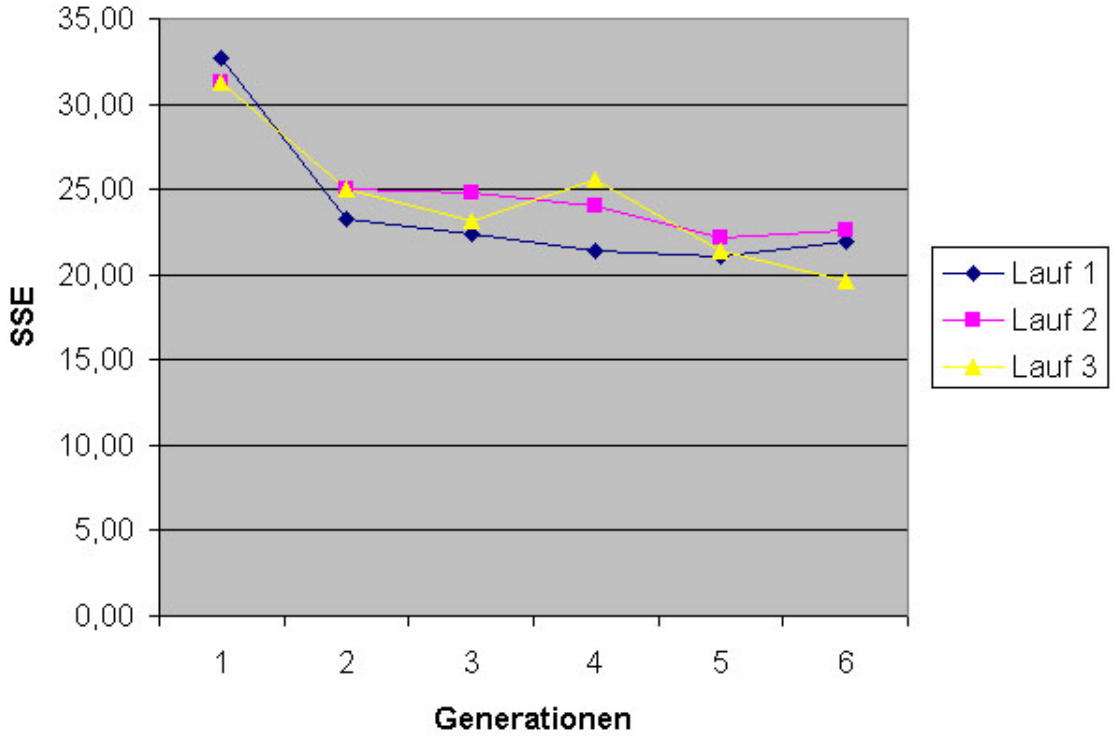

Abbildung 5.5.: Blätter - Gewinnernetze

Generation	SSE Lauf 1	SSE Lauf 2	SSE Lauf 3
1	32,73	31,28	31,24
2	23,29	25,00	25,00
3	22,33	24,85	23,10
4	21,35	24,03	25,60
5	21,03	22,14	21,36
6	21,99	22,65	19,60

Abbildung 5.6.: Blätter - Gewinnernetze - Tabelle

Die Endergebnisse fallen besser aus als das Ergebnis des besten Referenznetzes.

Durch die Verschlechterung der Läufe 1 und 2 zum Ende hin, ist es schwer etwas über die Tendenz zu sagen.

Genau wie in Versuch 1 haben die drei letzten Gewinnernetze ausschließlich die Logistische Funktion als Lernfunktion im Output Layer.

Eine genaue Aufstellung der Konfiguration der Gewinnernetze befindet sich im Anhang A.

Durchschnitt der selektierten Netze

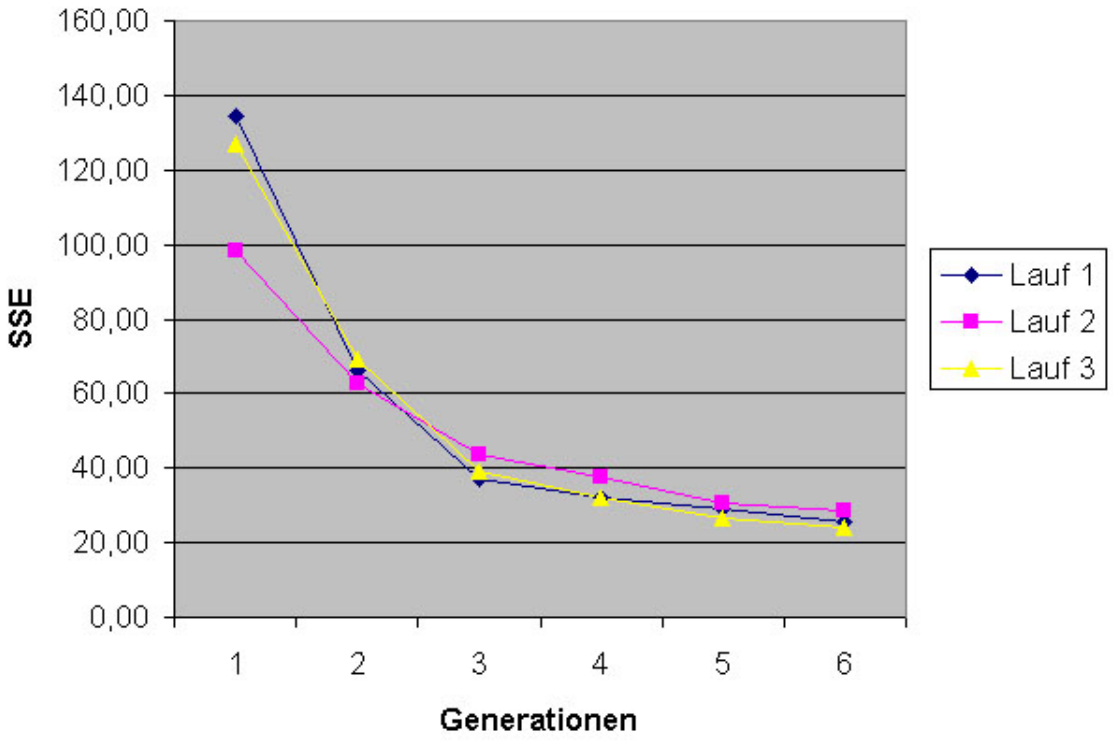

Abbildung 5.7.: Blätter - Durchschnitt der selektierten Netze

Die Verläufe der über die selektierten Netze gemittelten, summierten, quadratischen Fehler in Abbildung 5.7 und Tabelle 5.8 zeigen sich einheitlicher als die Verläufe der summierten quadratischen Fehler der Gewinnernetze. Sie fallen streng monoton.

Generation	SSE Lauf 1	SSE Lauf 2	SSE Lauf 3
1	134,35	98,27	126,72
2	66,02	62,92	69,05
3	37,31	43,82	39,21
4	32,26	37,59	32,33
5	29,12	30,84	26,49
6	25,46	28,71	24,05

Abbildung 5.8.: Blätter - Durchschnitt der selektierten Netze - Tabelle

Das Gefälle der drei Kurven nimmt ab. Zusammen mit dem Umstand, dass es bei den Gewinnernetzen in zwei von drei Fällen zum Ende hin Verschlechterungen gibt, deutet dies auf eine beginnende Stagnation des Verfahrens hin.

Bewertung

In Versuch 2 bleibt keines der Gewinnernetze in der letzten Generation hinter den Ergebnissen der Referenznetze zurück. Anders als in Versuch 1 ist die tatsächliche Bilderkennungsleistung unbefriedigend. Eine Überprüfung des Gewinnernetzes aus Lauf 3 ergab, dass 16 von 276 Mustern falsch erkannt wurden. Dies entspricht einer Fehlerrate von 5,8%, was, verglichen mit dem Ergebnis des dritten Laufs aus Versuch 1, nicht als Erfolg gewertet werden kann.

Die Tatsache, dass die drei gefundenen Gewinnernetze besser sind als das beste Referenznetz, deutet darauf hin, dass die schlechte Fehlerrate nicht auf das vorgestellte Verfahren zurückzuführen ist.

5.4. Zusammenfassung und Interpretation

In den vorangegangenen Versuchen konnte die Funktionsfähigkeit des vorgestellten Verfahrens gezeigt werden.

Die jeweils besten Referenznetze konnten in beiden Versuchen mit nur einer einzigen Ausnahme überboten werden. Dies ist bemerkenswert, weil den Referenznetzen jeweils deutlich mehr Trainingszyklen zur Verfügung standen.

Die erreichte Fehlerrate bei der Klassifikation der Blätter ist für den Autor unbefriedigend. Da die Referenznetze noch schlechter abschnitten, ist die Ursache nicht im vorgestellten Verfahren zu suchen. Die Hauptursache liegt vermutlich in der zu geringen Anzahl an Blattmustern. Zwar ist die Anzahl der Ziffernmuster nicht wesentlich größer, jedoch sind deren Eingangsmuster deutlich weniger komplex. Ein komplexerer Input Layer bedarf tendenziell auch einer größeren Anzahl an Eingangsmustern zur Erzielung der gleichen Fehlerrate[21].

Eine weitere Möglichkeit zur Verbesserung der Ergebnisse wäre gewesen dem Netz Hilfestellungen mit den Mustern zu übergeben. Ein Beispiel für eine solche Hilfestellung sind die in Kapitel 2.3 vorgestellten Hints.

Betrachtung des Zeitaufwandes

Abschließend soll auf den Zeitaspekt eingegangen werden. Das Referenznetz mit der längsten Trainingsdauer aus Versuch 2 brauchte für das Training 13 Stunden und 13 Minuten. Der schnellste Lauf aus Versuch 2 mit genetischer Optimierung wurde nach 37 Stunden und 58 Minuten abgeschlossen. Dies bedeutet unter günstigen Bedingungen etwa eine Verdreifachung der Laufzeit durch die genetische Optimierung.

Auswirkungen bei Änderung von Stellgrößen:

- Bei zweifacher Größe des Hidden Layers verdoppelt sich die Laufzeit[22].
- Eine Verdopplung der Generationen wirkt sich überlinear aus, da die Zahl der pro Generation berechneten Lernzyklen mit jeder Generation mit 1,2 multipliziert wird.
- Wird die Anzahl der Muster verdoppelt, verdoppelt sich die Laufzeit.
- Veränderungen an Input und Output Layer wirken sich linear auf die Rechenzeit aus. Die genauen Auswirkungen hängen von der Gesamttopologie ab.

Legt man die Rechenzeiten des hier verwendeten Computers zu Grunde, kann der Zeitaspekt ein Problem darstellen. Dies gilt insbesondere dann, wenn die Klassifikation deutlich ambitionierter[23] sein soll als in dieser Arbeit.

[21] Siehe Callan (2003)

[22] Dies gilt nur für eine Topologie mit nur einem Hidden Layer, wie sie hier verwendet wurde.

[23] *Ambitionierter* meint eine Erweiterung der Layergrößen, der Anzahl der Generationen und der Anzahl der Muster

6. Zusammenfassung und Ausblick

6.1. Resümee

Zielvorgabe

Gegenstand vorliegender Arbeit war die Untersuchung eines Verfahrens zur Klassifikation anhand von Beispielen. Im Kern sollte eine Kopplung aus Genetischen Algorithmen und Neuronalen Netzen stehen. Das Verfahren sollte ohne vertiefte Kenntnisse über Genetische Algorithmen und Neuronale Netze gute Klassifikatoren liefern.

Für die Versuche wurden Bilder von Blättern und handgeschriebene Ziffern benutzt.

Um einen Qualitätsmaßstab zu erhalten, wurden Vergleichsmessungen mit „handerstellten“ Neuronalen Netzen herangezogen.

Bewertung des Erreichten

Einsatz des Verfahrens von Laien

Ein Laie auf dem Gebiet der Informatik hätte sicherlich Probleme mit dem Verfahren. Die Software muss an die zu klassifizierenden Muster angepasst werden, um gültige Neuronale Netze zu erstellen. Darüber hinaus müssen einige Parameter für den genetischen Optimierer eingestellt werden.

Die Hemmschwelle zum Einsatz von Neuronalen Netzen wird dennoch gesenkt, da die Anzahl der frei wählbaren Parameter, die für die Erstellung eines neuen Klassifikators notwendig sind, auf eine einstellige Zahl reduziert wird.

Qualität des Verfahrens

Grundsätzlich ist festzuhalten, dass das Verfahren funktioniert. Hierzu sprechen die Versuche aus Kapitel 5 eine klare Sprache.

Dennoch ist eine Bewertung der Verfahrensqualität nicht unproblematisch. Zwar gibt es einen Vergleichsmaßstab, in Form der Referenznetze, aber es ist nicht gesichert, wie gut dieser Maßstab für eine Qualitätsbewertung nutzbar ist. Die Ergebnisse der Referenznetze sind Zufallsergebnisse. Sie können insofern nur eine „Idee“ von der Qualität des Verfahrens geben.

Beispielsweise wurde das jeweils neu initialisierte Recilient-Propagation-Netz aus Versuch 1 noch zwei weitere Male trainiert, um zu sehen, wie sehr das gewonnene Ergebnis dem „Normalfall“ entspricht. Das erste Ergebnis, das auch im Rahmen dieser Arbeit verwendet wurde, war außerordentlich gut und konnte in den beiden anderen Trainings nicht annähernd erreicht werden. So ist zu erklären, warum ein Lauf in Versuch 1 das Referenznetz nicht schlagen konnte: Das Referenznetz war auch ohne genetische Optimierung schon sehr gut.

Ein härteres Qualitätskriterium ist die Fehlerrate bei der tatsächlichen Mustererkennung. Dass das vorgestellte Verfahren in Versuch 1 einen Klassifikator erzeugen konnte, der alle Validierungsmuster erkannte, ist unbestreitbar ein Erfolg.

Die Fehlerrate in Versuch 2 ist unbefriedigend. Allerdings sind dort im Vergleich zu Versuch 1 nur die Hälfte an Generationen berechnet worden. Dennoch dauerte die Berechnung etwa doppelt so lang. Gleichzeitig hätten für Versuch 2 sehr viel mehr Muster zur Verfügung stehen müssen. Sowohl eine Ausweitung der Generationen als auch eine Vergrößerung der Musterbasis hätten jedoch Auswirkungen auf die Berechnungsdauer gehabt.

Es ist zu bedenken, dass sämtliche Versuche auf einem handelsüblichen Computer durchgeführt wurden. Die CPU dieses Computers war zum Zeitpunkt der Versuche 2 Jahre alt. Aktuelle Computer sind deutlich leistungsfähiger und es ist zu erwarten, dass der Trend hin zu noch leistungsfähigeren Computern weiter geht. Damit verbunden ist die Möglichkeit, den genetischen Optimierer deutlich mehr Berechnungen durchführen zu lassen. Dies umfasst eine mögliche Steigerung von:

- Generationen
- Individuen pro Generation
- Erlaubten Trainingszyklen für die Neuronalen Netze pro Generation
- Anzahl der Hidden Layer
- Größe der Hidden Layer

Es ist also davon auszugehen, dass sich die Leistungsfähigkeit des Verfahrens ohne Veränderung an der Software in Zukunft noch verbessern wird.

Dass das Verfahren grundsätzlich funktioniert, konnte gezeigt werden. Eine abschließende Qualitätsbewertung ist aus gennanten Gründen jedoch nicht möglich.

6.2. Ausblick

Erweiterung des Verfahrens

Zwei wichtige Aspekte im Aufbau Neuronaler Netze werden im hier beschriebenen Verfahren nicht optimiert:

- Netzgewichte bei der Netzinitialisierung
- Bias pro Neuron

Wie sich bei der großen Zahl von Versuchen, die im Rahmen dieser Arbeit gemacht wurden, zeigte, haben beide Aspekte starke Auswirkungen auf die Lernfähigkeit eines Neuronalen Netzes[1]. Daher sollten sie in einer zukünftigen Implementierung Teil des Genetischen Algorithmus sein.

Um das Verfahren zu vereinfachen, wäre ein gewisser Grad an Automatisierung wünschenswert. Nützlich wäre in diesem Zusammenhang die Generierung von gültigen Neuronalen Netzen anhand der übergebenen Musterdateien.

Die Erweiterung der Genetikmaschine um andere Verfahren stellt eine weitere Perspektive für die Zukunft dar. Als Beispiel sei *Simulated Annealing* genannt. Dies könnte das vorgestellte Verfahren an solchen Stellen unterstützen, an denen der Fortschritt stagniert. Hier wäre beispielsweise eine Erhöhung der Mutationsrate denkbar. Eine solche Kombination würde dem Verfahren erlauben, Teile des Lösungsraumes zu durchsuchen, die von den Individuen der jeweils aktuellen Generation „weit entfernt“ sind.

Der Autor geht davon aus, dass Feedforward-Netze, die mit der Error-Backpropagation-Lernregel arbeiten, nicht in jedem Fall das beste Verfahren für alle Klassifikationsaufgaben sind. Andere konnektionistische Verfahren wie zum Beispiel *Bayes-Klassifikatoren* oder *Support-Vector-Machines* werden ebenfalls zur Klassifikation eingesetzt. Es könnte also durchaus interessant sein dem Benutzer der vorgestellten Software weitere Klassifikationsverfahren anzubieten, die dann jeweils genetisch optimiert werden.

[1] Insbesondere fiel dies bei der Abweichungen der summierten quadratischen Fehler zweier Phänotypen auf, die aus dem gleichen Genotyp entstanden. Beide Phänotypen unterscheiden sich in diesen Fällen nur in den zwei genannten Aspekten.

Technische Erweiterung

Der nächste logische Schritt in technischer Hinsicht ist eine Verteilung der Netzberechnung. Bereits heute werden rechenintensive Programme auf Multiprozessorsystemen durchgeführt. Die Verbreitung von Multicore- und Multiprozessorsystemen nimmt zu. Insofern ist zu erwarten, dass sich der Einbau von Multithreadingfähigkeit erheblich auf die Leistungsfähigkeit des vorgestellten Verfahrens auswirken wird. Insbesondere eine Verteilung der Berechnung der einzelnen Neuronalen Netze pro Generation bietet sich an.

Literaturverzeichnis

[Abu-Mostafa 1995] ABU-MOSTAFA, Yaser S.: Lernhilfen für neuronale Netze. In: *Spektrum der Wissenschaft* 11/95 (1995), S. 84–89

[Callan 2003] CALLAN, Robert: *Neuronale Netze im Klartext*. Pearson Studium, 2003. – ISBN 3-8273-7071-x

[Churchland 2001] CHURCHLAND, Paul M.: *Die Seelenmaschine*. Spektrum Akademischer Verlag, 2001. – ISBN 3-8274-1020-7

[Cruse u. a. 2001] CRUSE, Holk ; DEAN, Jeffrey ; RITTER, Helge: *Die Entdeckung der Intelligenz*. dtv, 2001. – ISBN 3-406-44073-8

[Dawson 2003] DAWSON, Christian W.: *Computerprojekte im Klartext*. Pearson Studium, 2003. – ISBN 3-8273-7067-1

[Fahlman 1988] FAHLMAN, S. E.: Faster learning variations on back-propagation: An empirical study. In: *Proceedings of the 1988 Connectionist Models Summer School* (1988), S. 38–51

[Hinton und Williams 1986] HINTON, G. E. ; WILLIAMS, R. J.: Parallel Distributed Processing: Explorations in the Microstructure of Cognition. (1986)

[Holland 1992] HOLLAND, John H.: *Adaptation in Natural and Artificial Systems*. MIT Press, 1992. – ISBN 0-262581-11-6

[Lämmel und Cleve 2001] LÄMMEL, Uwe ; CLEVE, Jürgen: *Lehr- und Übungsbuch Künstliche Intelligenz*. Fachbuchverlag Leipzig im Carl Hanser Verlag, 2001. – ISBN 3-4462-1421-6

[Lenzen 2002] LENZEN, Manuela: *Natürliche und Künstliche Intelligenz*. Campus Verlag, 2002. – ISBN 3-593-37033-6

[Luger 2001] LUGER, George F.: *Künstliche Intelligenz*. Pearson Studium, 2001. – ISBN 3-8273-7002-7

[Meisel 2004] MEISEL, Andreas: *Einführung zum Projektmodul Neuronale Netze*. 2004

[Mitchell 1996] MITCHELL, Melanie: *An Introduction to Genetic Algorithms*. MIT Press, 1996. – ISBN 0-262-13316-4

[Porschke 2002] PORSCHKE, Jan P.: *Genetische Programmierung von verhaltensbasierten Agenten*, Hochschule für Angewandte Wissenschaften Hamburg, Diplomarbeit, 2002. – URL http://users.informatik.haw-hamburg.de/~kvl/porschke/diplom.pdf

[Riedmiller und Braun 1992] RIEDMILLER, M. ; BRAUN, H.: *RPROP - A fast adaptive learning algorithm*. 1992. – URL citeseer.ist.psu.edu/riedmiller92rprop.html

[Rumelhart u. a. 1986] RUMELHART, D. E. ; HINTON, G. E. ; WILLIAMS, R. J.: Learning Representations by BackPropagation Errors. In: *Nature* 323 (1986), S. 533–536

[Sarle 2004] SARLE, Warren: *How many hidden units should I use?* 2004. – URL www.faqs.org/faqs/ai-faq/neural-nets/part3/section-10.html

[Schaffer u. a. 1992] SCHAFFER, J. D. ; WHITLEY, Darrell ; ESHELMAN, Larry J.: *INTERNATIONAL WORKSHOP ON COMBINATIONS OF GENETIC ALGORITHMS AND NEURAL NETWORKS: COGANN-92*. IEEE Comput. Soc. Press, 1992. – ISBN 0-8186-2787-5

[Scherer 1997] SCHERER, Andreas: *Neuronale Netze*. vieweg, 1997. – ISBN 3-936427-45-3

[Spitzer 2000] SPITZER, Manfred: *Geist im Netz*. Spektrum Akademischer Verlag, 2000. – ISBN 3-8274-0572-6

[Spitzer 2002] SPITZER, Manfred: *Lernen*. Spektrum Akademischer Verlag, 2002. – ISBN 3-8274-1396-6

[Storjohann 2005] STORJOHANN, Timo: *Einsatz genetischer Lernverfahren für die Programmierung eines mobilen Roboters*, Hochschule für Angewandte Wissenschaften Hamburg, Diplomarbeit, 2005. – URL http://users.informatik.haw-hamburg.de/~kvl/storjohann/diplom.pdf

[Thompson 2001] THOMPSON, Richard F.: *Das Gehirn*. Spektrum Akademischer Verlag, 2001. – ISBN 3-8274-1080-0

[Trepel 2003] TREPEL, Martin: *Neuroanatomie*. Urban & Fischer, 2003. – ISBN 3-437-41297-3

[Zell und andere 1998] ZELL ; ANDERE: *SNNS User Manual, Version 4.2*. 1998. – URL http://www-ra.informatik.uni-tuebingen.de/downloads/SNNS/SNNSv4.2.Manual.pdf

[Zell 1994] ZELL, Andreas: *Simulation Neuronaler Netze*. Oldenbourg Verlag, 1994. – ISBN 3-4862-4350-0

A. Anhang - Gewinnernetz - Basisdaten

A.1. Gewinnernetze aus Versuch 1 Teil 2

A.1.1. Lauf 1

- Zeitrahmen
  - Generationen: 12
  - Stunden: 18
  - Minuten: 57
- Topologie und Aktivierungsfunktionen
  - Input Layer
    - Neuronen: 160
    - Aktivierungsfunktion: Act_Product
  - Hidden Layer
    - Neuronen: 37
    - Aktivierungsfunktion: Act_Logistic
  - Output Layer
    - Neuronen: 10
    - Aktivierungsfunktion: Act_Logistic
- Lernfunktion
  - Funktionsname: Rprop
  - δ min: 0,06
  - δ max: 50,0
  - Weight decay: 4,21
- Trainingsergebnis
  - Lernzyklen: 381
  - SSE: 1,53

A.1.2. Lauf 2

- Zeitrahmen
  - Generationen: 12
  - Stunden: 19
  - Minuten: 24
- Topologie und Aktivierungsfunktionen
  - Input Layer
    - Neuronen: 160
    - Aktivierungsfunktion: Act_Signum
  - Hidden Layer
    - Neuronen: 51
    - Aktivierungsfunktion: Act_Logistic
  - Output Layer
    - Neuronen: 10
    - Aktivierungsfunktion: Act_Logistic
- Lernfunktion
  - Funktionsname: Rprob
  - δ min: 0,15
  - δ max: 50,4
  - Weight decay: 4,11
- Trainingsergebnis
  - Lernzyklen: 381
  - SSE: 1,46

A.1.3. Lauf 3

- Zeitrahmen
  - Generationen: 12
  - Stunden: 15
  - Minuten: 32
- Topologie und Aktivierungsfunktionen
  - Input Layer
    - Neuronen: 160
    - Aktivierungsfunktion: Act_Signum
  - Hidden Layer
    - Neuronen: 79
    - Aktivierungsfunktion: Act_TanH
  - Output Layer
    - Neuronen: 10
    - Aktivierungsfunktion: Act_Logistic
- Lernfunktion
  - Funktionsname: Rprob
  - δ min: 1,0E-5
  - δ max: 50,0
  - Weight decay: 4,23
- Trainingsergebnis
  - Lernzyklen: 381
  - SSE: 1,83

A.2. Gewinnernetze aus Versuch 2 Teil 2

A.2.1. Lauf 1

- Zeitrahmen
  - Generationen: 6
  - Stunden: 43
  - Minuten: 27
- Topologie und Aktivierungsfunktionen
  - Input Layer
    - Neuronen: 625
    - Aktivierungsfunktion: Act_Exponential
  - Hidden Layer
    - Neuronen: 53
    - Aktivierungsfunktion: Act_Exponential
  - Output Layer
    - Neuronen: 6
    - Aktivierungsfunktion: Act_Logistic
- Lernfunktion
  - Funktionsname: Std_Backpropagation
  - Step Width: 0,2
  - max nonpropagated Error: 0,1
- Trainingsergebnis
  - Lernzyklen: 126
  - SSE: 21,99

A.2.2. Lauf 2

- Zeitrahmen
  - Generationen: 6
  - Stunden: 37
  - Minuten: 58
- Topologie und Aktivierungsfunktionen
  - Input Layer
    - Neuronen: 625
    - Aktivierungsfunktion: Act_TanH
  - Hidden Layer
    - Neuronen: 76
    - Aktivierungsfunktion: Act_TanH
  - Output Layer
    - Neuronen: 6
    - Aktivierungsfunktion: Act_Logistic
- Lernfunktion
  - Funktionsname: Std_Backpropagation
  - Step Width: 0,26
  - max nonpropagated Error: 0,23
- Trainingsergebnis
  - Lernzyklen: 126
  - SSE: 22,65

A.2.3. Lauf 3

- Zeitrahmen
  - Generationen: 6
  - Stunden: 38
  - Minuten: 23
- Topologie und Aktivierungsfunktionen
  - Input Layer
    - Neuronen: 625
    - Aktivierungsfunktion: Act_Logistic
  - Hidden Layer
    - Neuronen: 53
    - Aktivierungsfunktion: Act_TanH
  - Output Layer
    - Neuronen: 6
    - Aktivierungsfunktion: Act_Logistic
- Lernfunktion
  - Funktionsname: BackpropMomentum
  - Step Width: 0,21
  - Momentum: 0,38
  - Flat spot Elimination: 1,0E-5
  - max nonpropagated Error: 0,37
- Trainingsergebnis
  - Lernzyklen: 126
  - SSE: 19,6

B. Anhang - Referenznetze

Die folgenden Abbildungen zeigen die Trainingsläufe der Referenznetze aus Kapitel 5.

- Auf der x-Achse wird die Anzahl der Lernzyklen abgetragen.
- Auf der y-Achse wird der aufsummierte quadratische Fehler (SSE) abgetragen.
- Die schwarzen Kurven zeigen die Ergebnisse für die Trainingssmuster.
- Die roten Kurven zeigen die Ergebnisse für die Validierungsmuster.

Die zwei jeweils eingetragenen Fehlerkurven resultieren aus den Einzelmessungen des JavaNNS nach jedem Trainingszyklus.

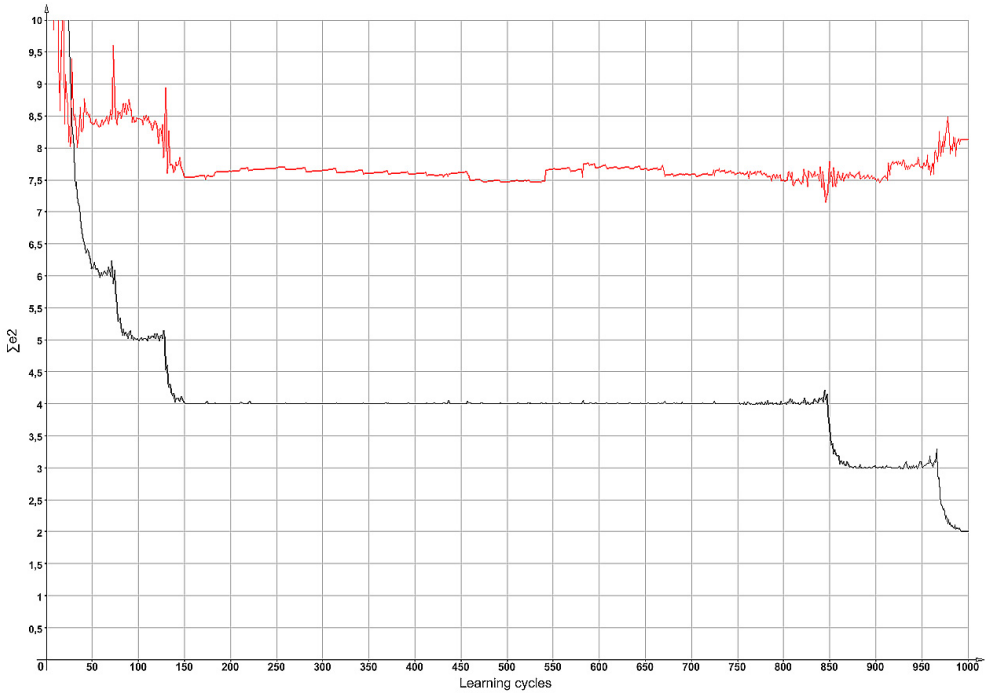

Abbildung B.1.: Versuch 1 Teil 1 Ziffern - Standard Backpropagation 1000 Zyklen

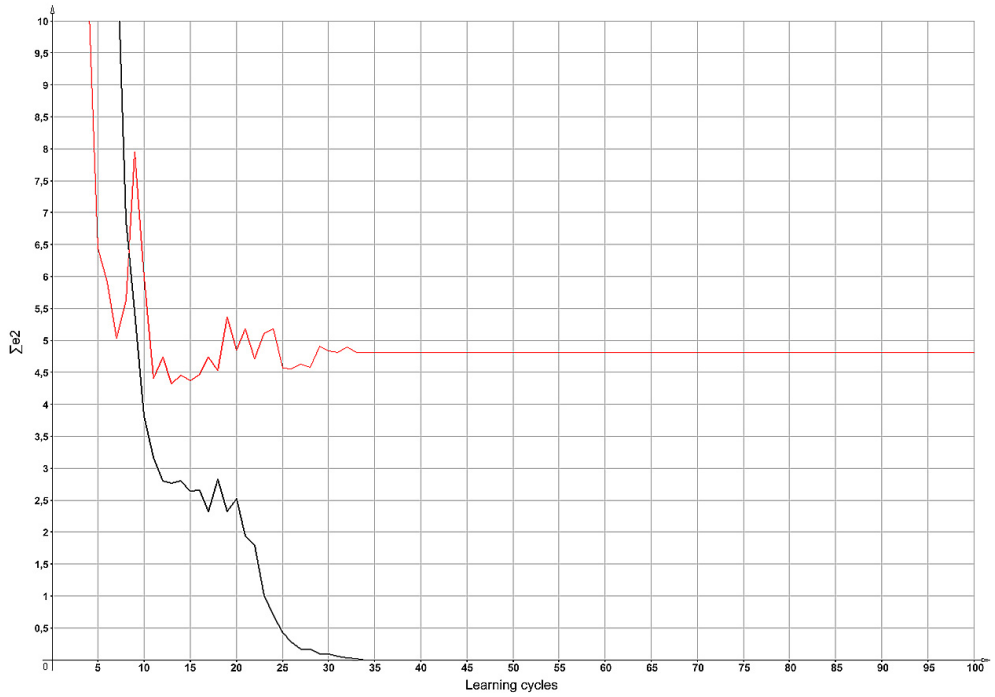

Abbildung B.2.: Versuch 1 Teil 1 Ziffern - Backpropagation Momentum 100 Zyklen

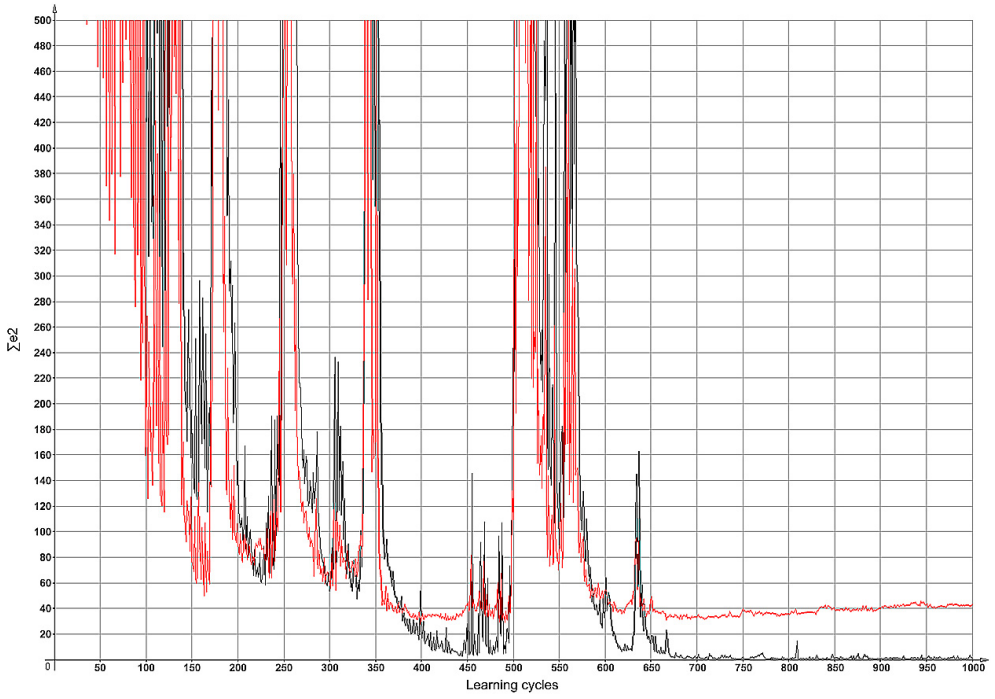

Abbildung B.3.: Versuch 1 Teil 1 Ziffern - Quickpropagation 1000 Zyklen

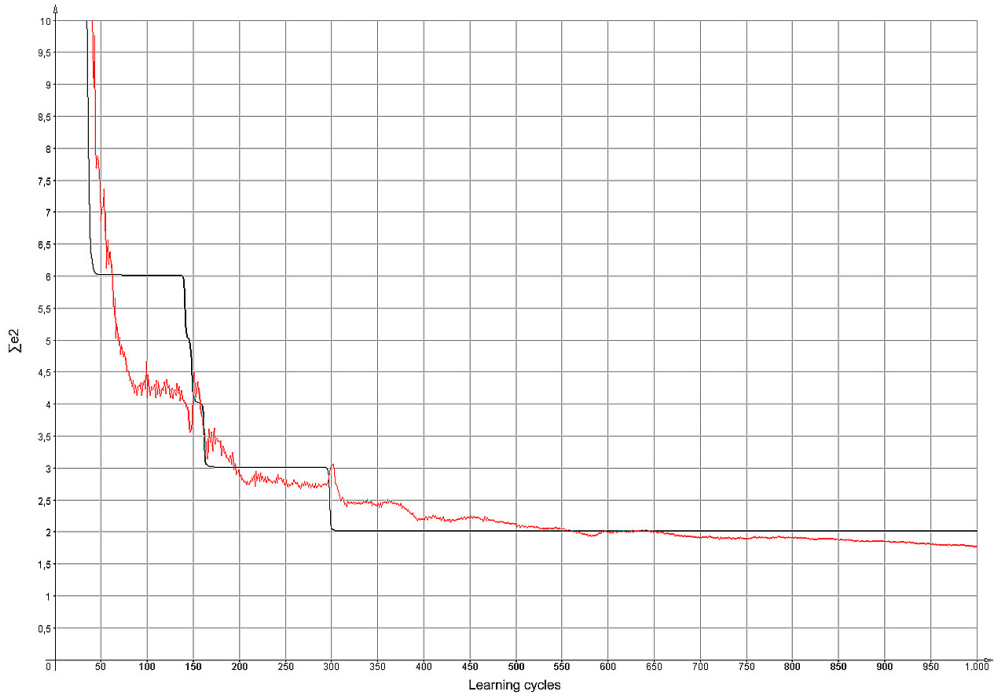

Abbildung B.4.: Versuch 1 Teil 1 Ziffern - Resilient Propagation 1000 Zyklen

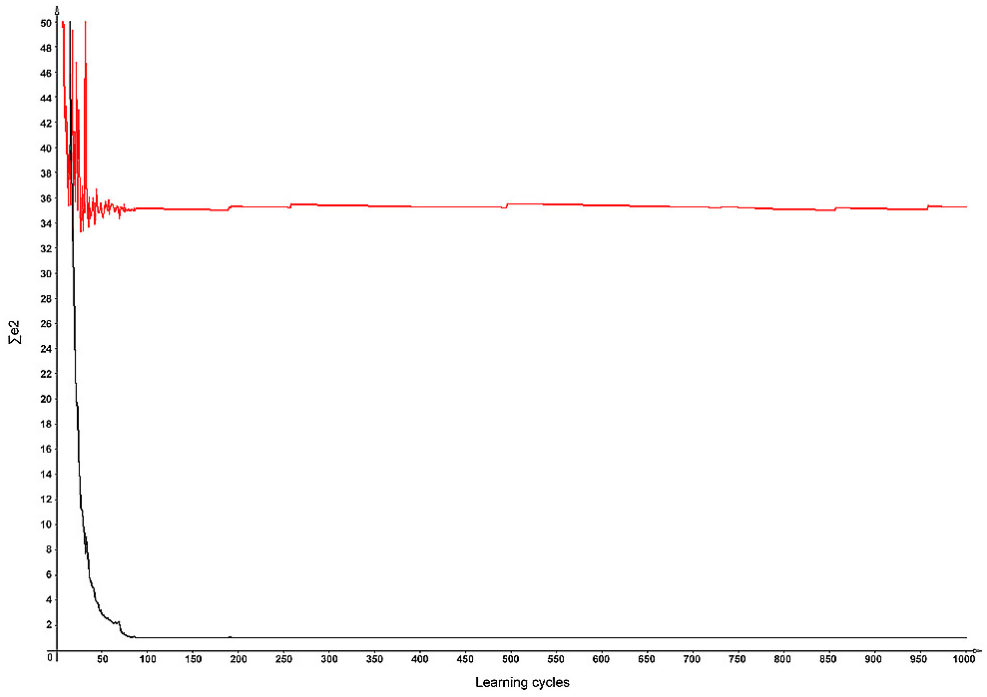

Abbildung B.5.: Versuch 2 Teil 1 Blätter - Standard Backpropagation 1000 Zyklen

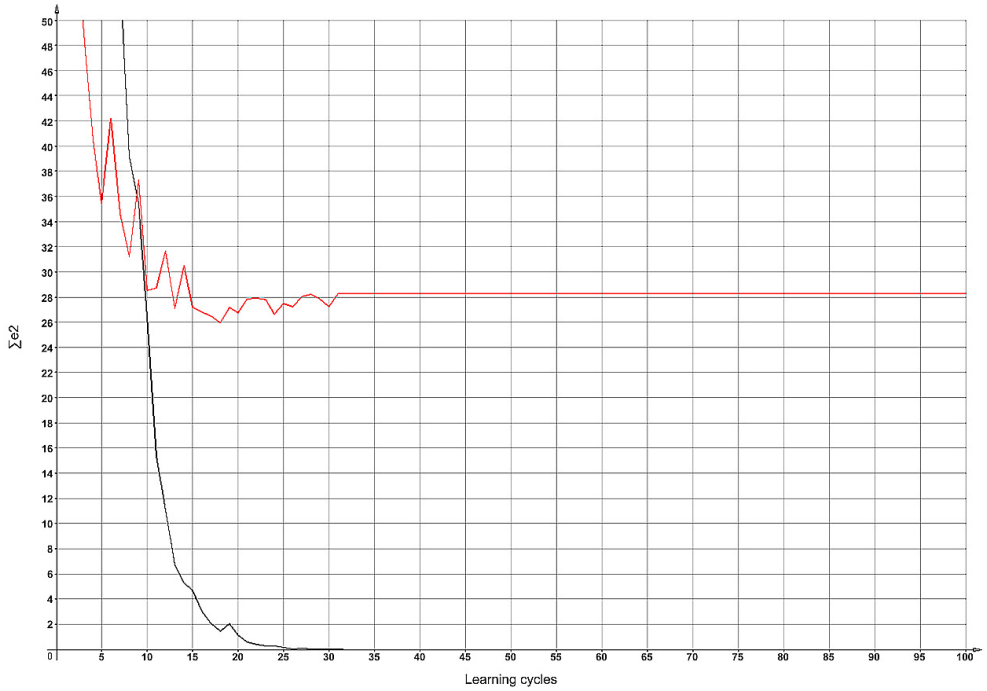

Abbildung B.6.: Versuch 2 Teil 1 Blätter - Backpropagation Momentum 100 Zyklen

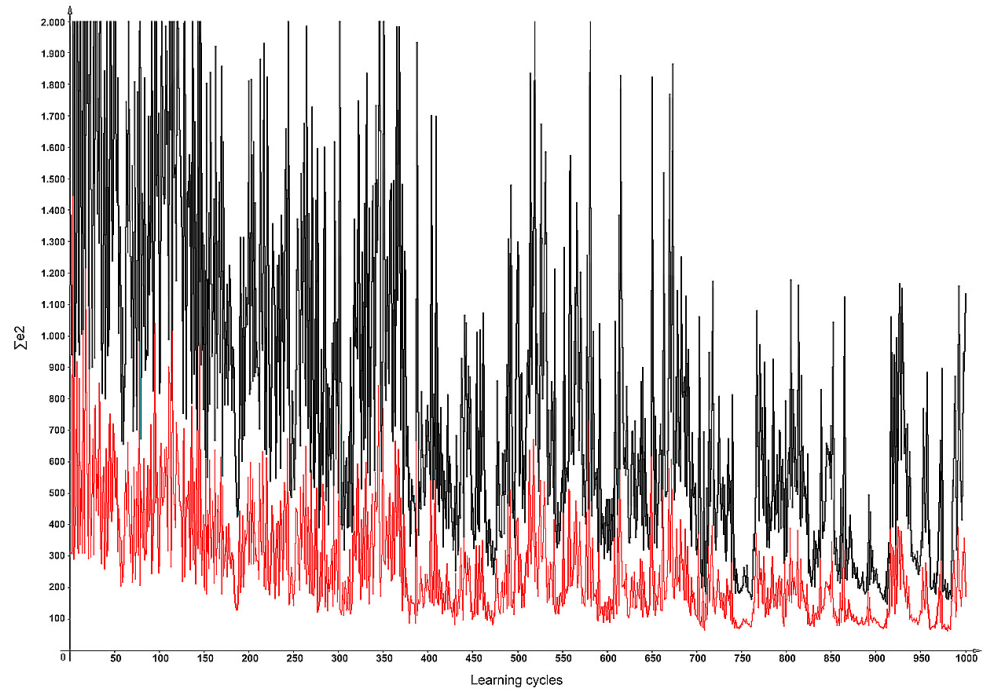

Abbildung B.7.: Versuch 2 Teil 1 Blätter - Quickpropagation 1000 Zyklen

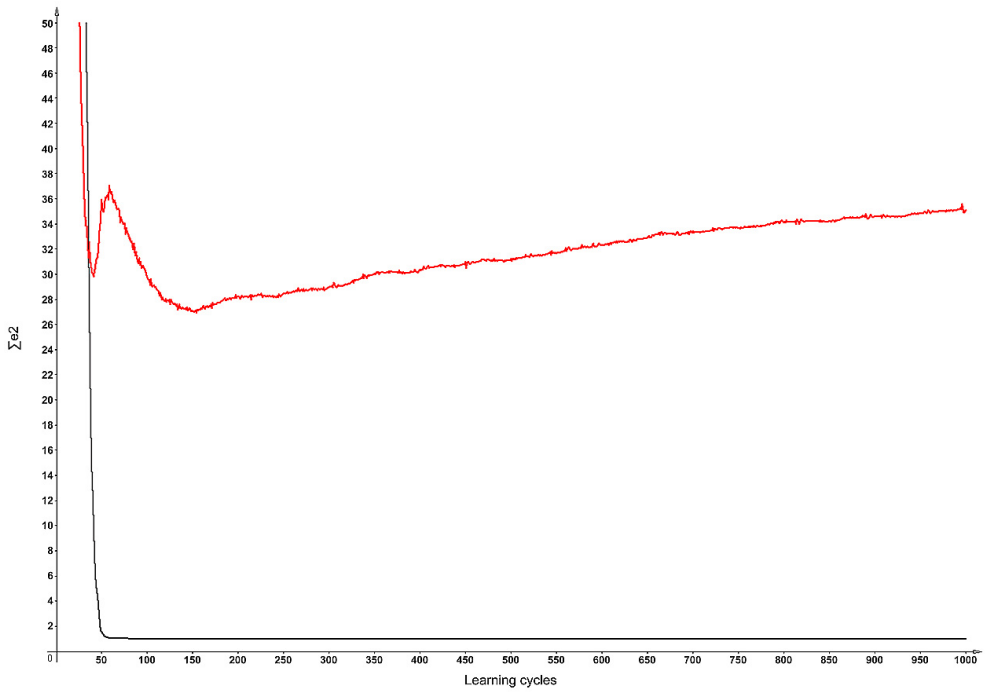

Abbildung B.8.: Versuch 2 Teil 1 Blätter - Resilient Propagation 1000 Zyklen

C. Anhang - Grafische Benutzeroberfläche

Abbildung C.1.: Programmstart

Abbildung C.2.: Kontrollfenster

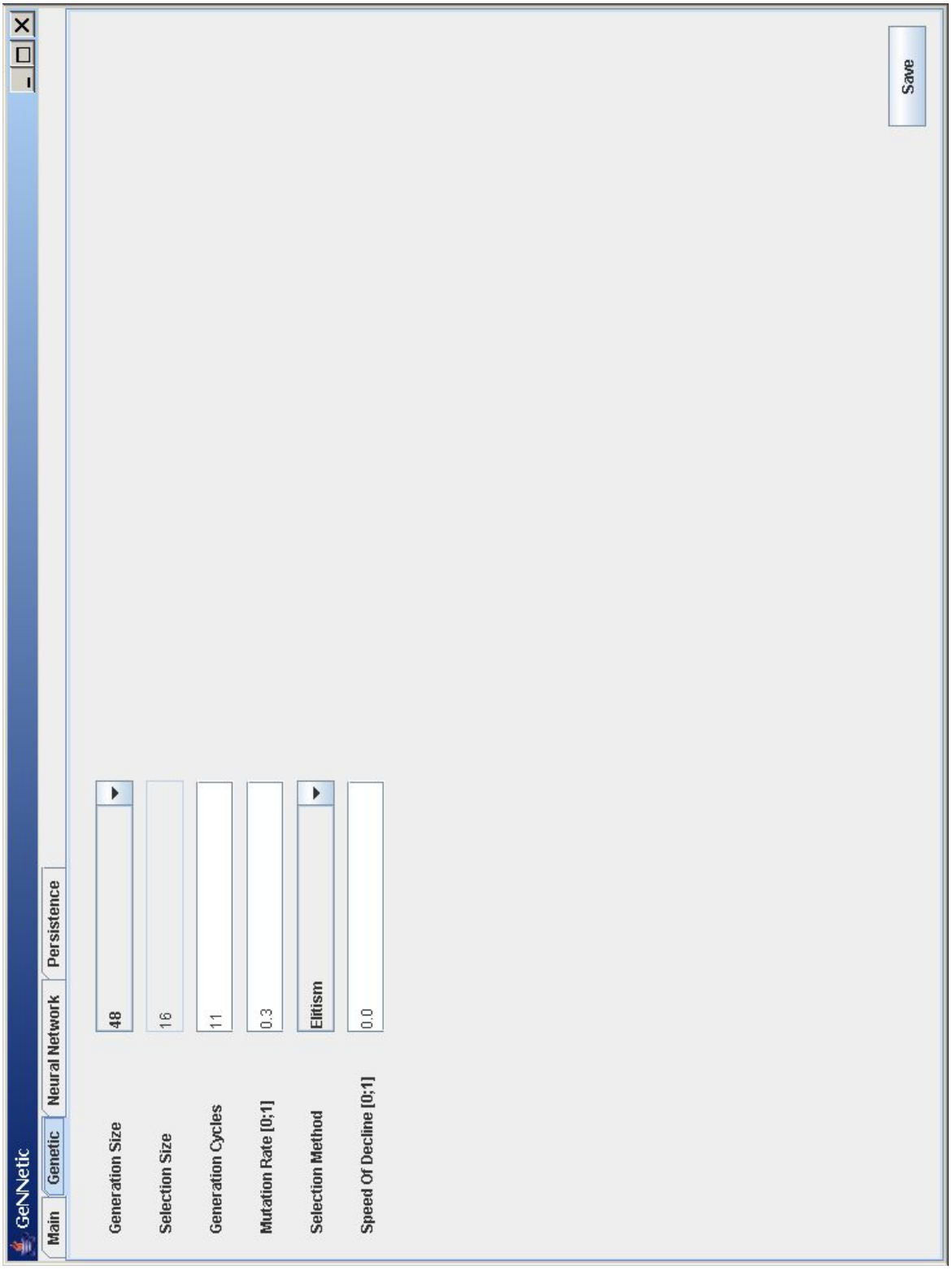

Abbildung C.3.: Genetikfenster

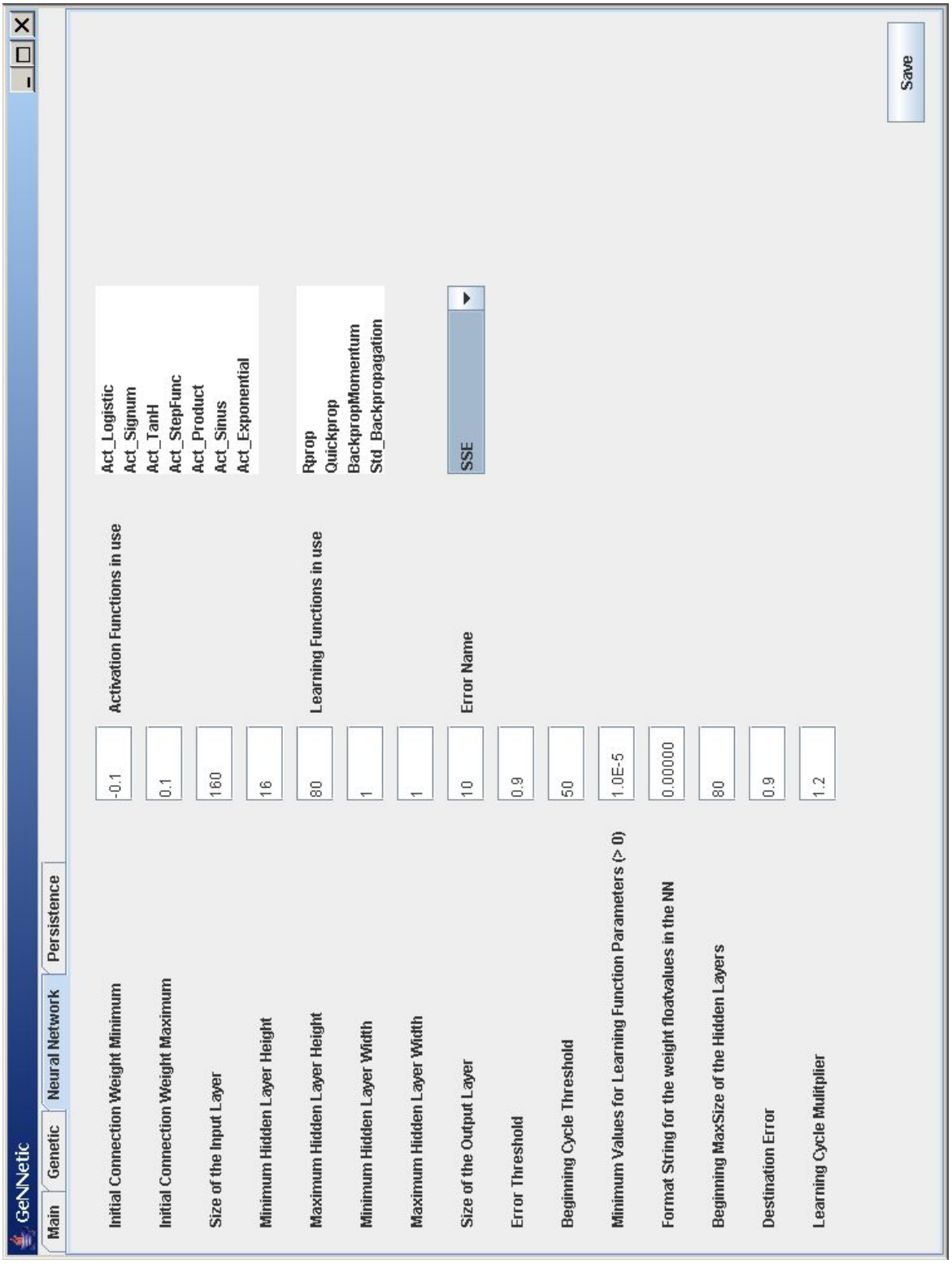

Abbildung C.4.: Netzwerkfenster

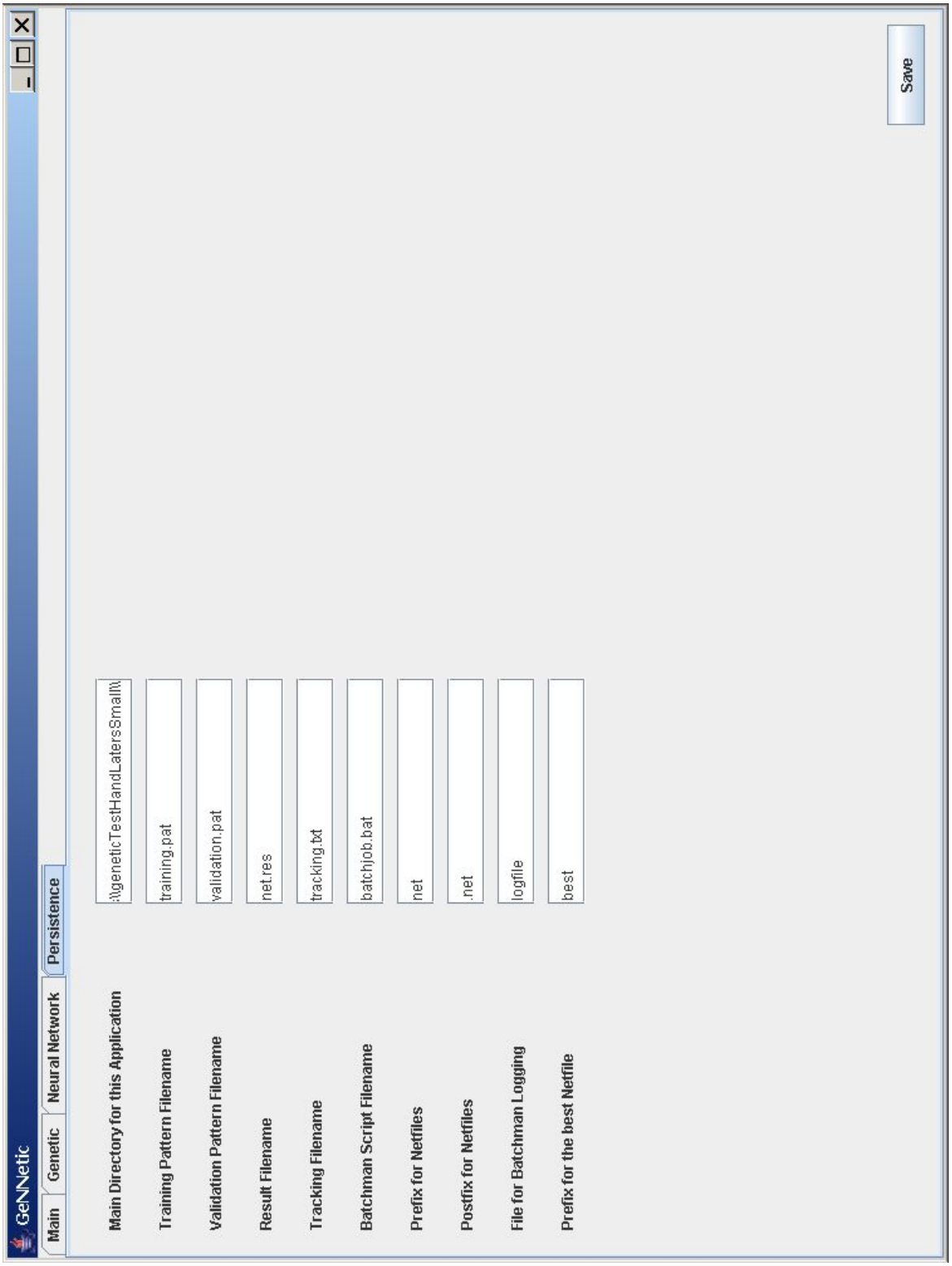

Abbildung C.5.: Persistenzfenster

Glossar

Aktivierungsfunktion Gibt den Ausgabewert eines Neurons abhängig vom Ergebnis der Propagierungsfunktion an.

Allel Allele sind mögliche Ausprägungen eines Gens.

Bayes-Klassifikator Bayes-Klassifikatoren basieren auf der statistischen Entscheidungstheorie. Die Zugehörigkeit zu einer Klasse im Merkmalsraum lässt sich als Dichtefunktion einer Verteilung angeben, so die Annahme. Ein Objekt wird als der Klasse mit der höchsten Wahrscheinlichkeit zugehörig betrachtet.

Bias Schwellwert, der jedem Neuron zugeordnet wird. Das Bias fließt in die Propagierungsfunktion ein.

Cross-Over Siehe Rekombination.

Ersetzungsschema Vorgabe zur Auswahl von Individuen während der Selektion abhängig von deren Fitness.

Fitness Ergebnis der Fitnessfunktion, das die Eignung des Individuums zum Erreichen einer gegebenen Aufgabe anzeigen soll.

Fitnessfunktion Funktion zur Ermittlung der Fitness eines Individuum.

Generation Gruppe von Individuen zu einem bestimmten Zeitpunkt im Genetischen Algorithmus.

Genetischer Algorithmus Verfahren, das die biologische Evolution, basierend auf Konkurrenz, Selektion und Reproduktion, imitiert.

Genotyp Bauplan eines Phänotypen.

Gewicht Andere Bezeichnung für eine Verbindung zwischen zwei Neuronen.

Globales Minimum Globales Optimum, bei dem die Lösung einen möglichst kleinen Wert darstellen soll.

Globales Optimum Punkt in einem Suchraum, der für die insgesamt beste Lösung steht.

Heuristik Erfahrungsgerichtetes Suchen im Suchraum.

Hidden Layer Innere Schichten in einem FeedForward Netz

Hints Hinweise, die einem Neuronalen Netz die Wichtigkeit oder Irrelavanz bestimmter Merkmale vermitteln.

Individuum Kandidat für eine Lösung im Suchraum. Siehe Phänotyp und Genotyp.

Input Layer Dateneingangsschicht von Neuronalen Netzen.

Konnektionistisch Lernverfahren, deren Wissensrepräsentation und Lernverhalten auf Netzwerken beruht.

Layer Aus mehreren strukturell zusammengehörigen Neuronen bestehende Menge von Neuronen.

Lernen Siehe Training.

Lernfunktion Stellt eine Berechnungsvorschrift für die Veränderung der Verbindungsgewichte zwischen den Neuronen dar.

Lokales Minimum Nach oben offene Extremstelle im Suchraum.

Lokales Optimum Extremstelle im Suchraum. Die Gestalt ist abhänging davon, ob das Optimum ein Maximum oder ein Minimum ist.

Mutation Zufällige Änderung am Genotyp.

Neuron Betrachtet man das Neuronale Netz als Graph, sind die Neuronen die Knoten. Es handelt sich um einfache Einheiten, in denen Berechnungen durchgeführt werden. Die geschieht auf Basis der ihnen von anderen Neuronen oder dem Dateneingang übergebenen Daten. Das Ergebnis dieser Berechnungen wird an andere Neuronen oder den Datenausgang übergeben.

Neuronale Netze Konnektionistisches Verfahren, das von Tiergehirnen inspiriert ist.

Output Layer Datenausgangsschicht von Neuronalen Netzen.

Phänotyp Ausprägung eines Genotypen.

Population Gruppe von Individuen.

Propagierungsfunktion Summe der Neuroneneingänge plus Bias.

Rekombination Individuen werden aus Teilen mehrerer Individuen der Vorgängergeneration zusammengesetzt.

Selektion Auswahl bestimmter Individuen anhand des Ersetzungsschemas.

Simulated Annealing Verfahren zur Vermeidung lokaler Minima.

SSE Summe der Fehlerquadrate aller Output Neuronen.

Support-Vector-Machines Ausgehend von der Grundannahme, dass mit einer passenden nichtlinearen Abbildung ausreichend hoher Dimension Daten aus zwei Kategorien stets mit Hilfe einer Hyperebene getrennt werden können, beruht die Klassifikation mit Support-Vector-Machines in einem hochdimensionalen Merkmalsraum auf dem Finden dieser Hyperebene, welche den entsprechenden Merkmalsraum optimal trennt.

Topologie Struktur des Neuronalen Netzes. Bestehend aus Neuronen, deren Verbindungen zu anderen Neuronen sowie der Rolle, die die Neuronen im Netz spielen.

Training Phase in der das Lernverfahren eine Wissensbasis aufbaut.

Verbindung Betrachtet man das Neuronale Netz als Graph, sind die Verbindungen die Kanten. Die Gewichtung der Verbindung beeinflusst die Information, die über die Verbindung fließt.

Überwachtes Lernen Verfahrensfamilie deren Hauptmerkmal darin besteht, dass im Training Beispieldaten mit einer dazu gehörigen Kategorie (Lösung) gelernt werden.

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §22(4) ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 16. März 2006	
Ort, Datum	Unterschrift