

Studiengang: Elektro- und Informationstechnik

Prüfer: Prof. Dr. P. Levi

Betreuer: Dipl. Inf. T. Buchheim
Dipl. Ing. M. Necker, M. Sc.

Studienarbeit Nr. 1937

Anwendung von Neuro-Fuzzy Methoden für die Robotersteuerung

Dipl. Ing. (BA) Juliane Kanne

begonnen am: 2. Februar 2004

beendet am: 31. Juli 2004

CR-Klassifikation: F.1.1, I.2.3, I.2.4, I.2.6, I.5.1

Institut für Parallele und
Verteilte Höchstleistungsrechner
Universität Stuttgart
Universitätsstraße 38
D-70569 Stuttgart

Inhaltsverzeichnis

1	Motivation und Einleitung	1
1.1	Aufgabenstellung	1
1.2	Aufbau der Arbeit	2
2	Neuro-Fuzzy-Systeme	3
2.1	Neuronale Netze	3
2.1.1	Struktur Neuronaler Netze	4
2.1.2	Lernmechanismen Neuronaler Netze	5
2.2	Fuzzy-Logik	6
2.2.1	Begriffe der Fuzzy-Logik	7
2.2.2	Fuzzy-Regler	7
2.3	Neuro-Fuzzy-Systeme	13
3	Die Software NFident	15
3.1	Neuro-Fuzzy Function Approximation (NEFPROX)	15
3.1.1	NEFPROX - Definition	17
3.2	Adaptive Neuro Fuzzy Inference System (ANFIS)	22
3.3	Die NFident Software	23
4	Modifikation und Test der Software NFident	25
4.1	Anwendungsmöglichkeiten	27
4.2	Funktionsapproximation	27
4.2.1	Funktionsapproximation - zweidimensional	28
4.2.2	Funktionsapproximation - dreidimensional	32
4.2.3	Fazit	36
5	Robotersteuerung mit Neuro-Fuzzy-Systemen	41

5.1	Robotersteuerung durch Vorgabe von Regeln	41
6	Lernstrategie bei der Robotersteuerung	49
6.1	Reinforcement Learning	49
6.2	Einbindung des NFident-Modells ins Reinforcement Learning	51
6.2.1	Reinforcement Learning mit einem einfachen Szenario	52
6.2.2	Anpassung des Netzes zur Robotersteuerung	57
7	Zusammenfassung und Ausblick	59
A	Netz- und Parameterdateien der Software NFident	61
A.1	NFident Parameter-Datei	61
A.2	NFident Netz-Datei für Robotersteuerung (1)	64
A.3	NFident Netz-Datei für Jumper-Szenario	65
A.4	NFident Netz-Datei für Robotersteuerung (2)	67
B	Beschreibung der implementierten Funktionen	71
B.1	NFidentNet	71
	Abbildungsverzeichnis	75
	Tabellenverzeichnis	77
	Abkürzungen und Formelzeichen	79
	Literaturverzeichnis	81

Kurzfassung

Diese Arbeit befasst sich mit der Anwendung der Software NFident zur Steuerung eines Roboters mithilfe von Neuro-Fuzzy-Systemen.

Zunächst werden die zugrunde liegenden Methoden der Fuzzy-Logik und der Neuronalen Netze kurz erläutert. Es werden vor allem die in der Software NFident implementierten Modelle des Sugeno- und Mamdani-Reglers genauer betrachtet. Außerdem werden die verwendeten Neuro-Fuzzy Modelle vorgestellt. Hierbei handelt es sich um das an der Otto-von-Guericke-Universität in Magdeburg entwickelte NEFPROX-Modell sowie um das ANFIS-Modell von J.-S.R. Jang.

Des Weiteren wird die Anpassung der Software an die gegebene Aufgabe erläutert und es werden einige Tests beschrieben, die zeigen, in wie weit sich die Neuro-Fuzzy-Systeme grundsätzlich zur Steuerung eines Roboters eignen.

Schließlich wird gezeigt, wie Neuro-Fuzzy-Systeme zur Robotersteuerung eingesetzt werden können. Die Anwendungsmöglichkeiten werden zunächst in einer Simulationsumgebung entwickelt und schließlich auf einem Roboter getestet.

Kapitel 1

Motivation und Einleitung

Neuronale Netze und Fuzzy Logik sind zwei Methoden aus dem Bereich der künstlichen Intelligenz. Beide Methoden orientieren sich stark an biologischen Vorbildern. Während Neuronale Netze die Lernvorgänge im menschlichen Gehirn imitieren, versucht man mithilfe der Fuzzy-Logik, die menschliche, “unscharfe” Entscheidungsfindung nachzubilden. Seit den 90er Jahren haben Neuronale Netze und Fuzzy Logik zunehmend Anwendung in einer Vielzahl von Bereichen gefunden. Diese erstrecken sich von der Unterhaltungselektronik über Prozesssteuerung in der Industrie bis hin zu Entscheidungsunterstützungssystemen im Finanzsektor.

Neuro-Fuzzy-Systeme kombinieren die Methoden der Neuronalen Netze und der Fuzzy-Logik und bilden daraus lernfähige Netze die in der Lage sind, unscharfe und unpräzise Daten zu verarbeiten, und deren Wissen in Form von (linguistischen) Regeln dargestellt wird. Man findet Neuro-Fuzzy-Systeme bereits in vielen industriellen Anwendungen und sie bieten sich speziell dort an, wo herkömmliche Regler wegen zu hoher Komplexität oder zu unpräzisen Daten versagen.

Ein Beispiel für die Realisierung eines Neuro-Fuzzy Systems ist das Neuro-Fuzzy-Function-Approximation-Modell (NEFPROX). Es wurde 1994 an der Otto-von-Guericke-Universität in Magdeburg entwickelt und ist speziell für die Anwendung im Bereich der Funktionsapproximation optimiert. Ein weiteres Modell für Neuro-Fuzzy-Systeme ist das Adaptive Neuro Fuzzy Inference System (ANFIS) das von J.-S.R. Jang entworfen wurde. Zur Erstellung und Anwendung von Neuro-Fuzzy-Netzen stellt die Universität Magdeburg die Software NFident zur Verfügung, in welcher das NEFPROX Modell sowie teilweise das ANFIS Modell implementiert ist.

1.1 Aufgabenstellung

Im Rahmen dieser Studienarbeit wird die Anwendung von Neuro-Fuzzy-Methoden für die Steuerung eines Roboters betrachtet. Der Roboter soll im Rahmen des RoboCups in die Lage versetzt werden, Fußball zu spielen. Sensorik und Aktorik des Roboters

werden als bereits verwirklicht vorausgesetzt. Das Neuro-Fuzzy-System soll dafür eingesetzt werden, aus Daten wie Position des Roboters, Ballposition, Position weiterer Spieler etc. die Aktionen des Roboters zu steuern, d.h. Vorgaben für Translation und Rotation des Roboters zu machen.

Um diese Funktion zu realisieren soll als Basis die Software NFident verwendet werden. Die Software ermöglicht zunächst, mithilfe einer Menge von Trainingsdaten ein Neuro-Fuzzy-Netz zu generieren, zu testen und zu verwenden. Um die Software für den speziellen Fall der Robotersteuerung anwenden zu können, müssen die Funktionen erweitert und angepasst werden, so dass das Programm anschließend in das Gesamtsystem integriert werden kann. Des Weiteren muss getestet werden, in wie weit Neuro-Fuzzy-Netze grundsätzlich für die Robotersteuerung geeignet sind. Dies geschieht anhand einfacher Beispiele zur Funktionsapproximation. Anschließend soll die Neuro-Fuzzy-Steuerung in Verbindung mit einer Robotersimulation und schließlich am realen Roboter getestet werden.

1.2 Aufbau der Arbeit

Die Gliederung der Arbeit orientiert sich an den genannten Themenschwerpunkten. In Kapitel 2 werden die Grundlagen zum Thema Neuronale Netze, Fuzzy Logik und Neuro-Fuzzy Systeme dargestellt.

Die NFident Software wird in Kapitel 3 vorgestellt und die dort implementierten Modelle werden näher betrachtet.

Kapitel 4 befasst sich mit der an die Robotersteuerung angepassten Software und beschreibt ihre Anwendungsmöglichkeiten. Des Weiteren wird die Funktion der Neuro-Fuzzy-Modelle überprüft.

Schließlich wird in Kapitel 5 gezeigt wie die Software zur Robotersteuerung eingesetzt werden kann.

Das abschließende Kapitel 7 fasst die wichtigsten Ergebnisse der Arbeit zusammen und gibt einen Ausblick auf zukünftige Anwendungen.

Kapitel 2

Neuro-Fuzzy-Systeme

Als Neuro-Fuzzy Systemen bezeichnet man Systeme, welche die Prinzipien von Neuronalen Netzen und Fuzzy Logik kombinieren, um so die Vorteile beider Methoden besser zu nutzen und Nachteile auszugleichen.

In den Abschnitten 2.1 und 2.2 sollen zunächst Neuronale Netze und Fuzzy Logik getrennt beschrieben werden. Abschnitt 2.3 erläutert wie die beiden Methoden zu Neuro-Fuzzy-Systemen kombiniert werden können.

2.1 Neuronale Netze

Die Erforschung künstlicher Neuronaler Netze begann zunächst, um 1940, mit dem Ziel, die Funktionsweise des menschlichen Gehirns näher zu erforschen. Dabei ging man von der Idee aus, dass das Gehirn aus Nervenzellen, den Neuronen, besteht, welche untereinander verbunden sind und sich über elektrische Signale beeinflussen. Die Lernfähigkeit des menschlichen Gehirns kommt dadurch zustande, dass Lernvorgänge die Neuronenverbindungen physikalisch verändern: während eines Lernvorgangs wird die Verbindung zwischen zwei betroffenen Neuronen verstärkt, sie ist in Zukunft effektiver und hat somit ein stärkeres Gewicht in der Funktion des Gesamtnetzes.

Das Ziel bei der Erforschung künstlicher Neuronaler Netze ist es, durch geschickte Vernetzung sehr vieler einfacher Schaltungen, sogenannter Neuronen, eine mit dem menschlichen Gehirn vergleichbare Leistungsfähigkeit zu erreichen. Aufbauend auf diesen Überlegungen gelang es Frank Rosenblatt 1958 erstmals, ein lernfähiges künstliches Neuronales Netz, das Perceptron, zu entwickeln [Rose58].

Nach einer relativ langen Phase während derer Neuronale Netze nur wenig Anerkennung fanden, begann dann um 1980 die Zeit der “modernen” Neuronalen-Netz-Forschung, die bis heute viel Beachtung erfährt. Es wurden mehrschichtige Neuronale Netze entwickelt die deutlich leistungsfähiger waren als die bisherigen Ansätze und schließlich wurde 1986 das als “Backpropagation” bekannte Lernverfahren entwickelt, welches die Geschwindigkeit und Leistungsfähigkeit Neuronaler Netze weiter erhöhte.

Heutzutage werden Neuronale Netze in vielen Bereichen angewendet und erforscht. Wichtige Anwendungsgebiete sind die Mustererkennung (Zeichenerkennung, Spracherkennung, Bilderkennung, Sternklassifikation, etc), Bewertungen (Produktqualität gut/schlecht), Zeitreihenanalyse (Wettervorhersage, Kursprognose an der Börse) sowie Steuerungs- und Regelungsaufgaben (Roboter, Motoren, etc.).

2.1.1 Struktur Neuronaler Netze

Ein Neuronales Netz besteht aus einer Menge von Neuronen und einer Menge von Verbindungen, die zusammen die Struktur eines gerichteten Graphen besitzen. Die Knoten des Graphen heißen Neuronen, die Kanten Verbindungen. Jedes Neuron kann eine beliebige Menge von Verbindungen empfangen, über die es seine Eingabe erhält. Aus der Eingabe erzeugt jedes Neuron genau eine Ausgabe, die es über eine beliebige Menge von Verbindungen aussendet. Das Neuronale Netz erhält aus Verbindungen, die aus der "Außenwelt" kommen seine Eingabe und gibt seine Ausgaben über Verbindungen ab, die in der "Außenwelt" enden.

Im Allgemeinen gliedern sich die Neuronen in Schichten: eine Eingabeschicht, eine Ausgabeschicht und eine oder mehrere verdeckte, also von außen nicht sichtbare Schichten. Man unterscheidet zwischen Netzen ohne Rückkopplung (Feedforward-Netz) und Netzen mit Rückkopplung (rekurrente Netze). In Feedforward-Netzen werden Daten nur in eine Richtung weitergegeben, es existiert kein Pfad, der von einem Neuron wieder zurück zu diesem Neuron führt. Dahingegen gibt es bei rekurrenten Netzen Verbindungen, die direkt oder über andere Neuronen wieder zum selben Neuron zurückführen.

Die Übertragungsfunktion eines Neuronalen Netzes ergibt sich nun folgendermaßen: Jeder Verbindung ist ein Gewicht w zugeordnet. Jedes Neuron u ist gegeben durch seinen Eingabevektor $x = (x_1, x_2, \dots, x_n)$, den Gewichtsvektor $w = (w_1, w_2, \dots, w_n)$, eine Aktivierungsfunktion $A_u(x, w)$ und eine Ausgabefunktion O_u . Durch $O_u(A_u(x, w)) = a$ wird der Ausgabewert a des Neurons bestimmt der über die ausgehenden Verbindungen an andere Neuronen weitergegeben wird.

Eine Eingabe von außen durchläuft nun also die verschiedenen Schichten des Neuronalen Netzes. Anhand der Gewichte der durchlaufenen Verbindungen und der in den einzelnen Neuronen implementierten Funktionen ergeben sich die Ausgabewerte der Neuronen der Ausgabeschicht und damit der Ausgabevektor des Netzes.

Abbildung 2.1 zeigt ein einfaches feedforward Netz mit zwei verdeckten Schichten.

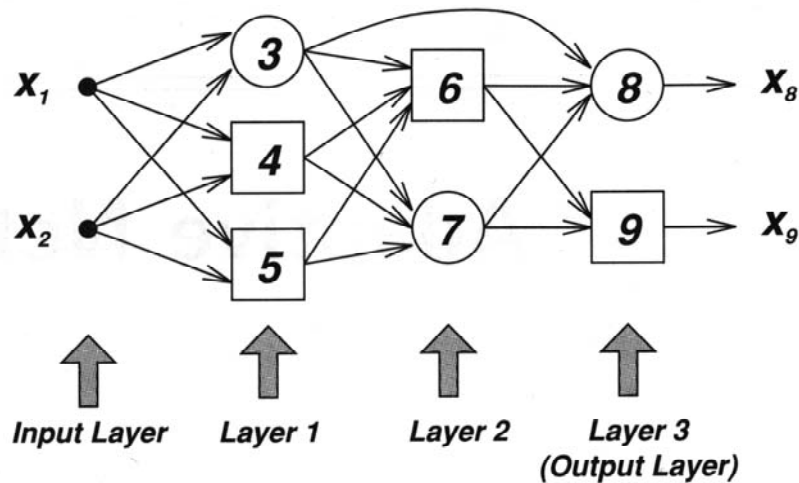


Abbildung 2.1: Neuronales feedforward Netz mit Ein- und Ausgabeschicht sowie zwei verdeckten Schichten, [Jang97]

Mathematisch lassen sich alle wesentlichen Modelle Neuronaler Netze durch Definition 2.1.1 (aus [Nauck94]) beschreiben. Dabei wird das Neuronale Netz als eine formale Struktur aufgefasst, die durch eine Menge und einige Abbildungen beschreibbar ist.

Definition 2.1.1 Ein Neuronales Netz ist ein Tupel (U, W, A, O, NET, ex) wobei gilt:

1. U ist eine endliche Menge von Verarbeitungseinheiten (Neuronen),
2. W , die Netzstruktur, ist eine Abbildung vom kartesischen Produkt $U \times U$ in \mathbb{R} ,
3. A ist eine Abbildung, die jedem $u \in U$ eine Aktivierungsfunktion $A_u: \mathbb{R}^3 \rightarrow \mathbb{R}$ zuordnet,
4. O ist eine Abbildung, die jedem $u \in U$ eine Ausgabefunktion $O_u: \mathbb{R} \rightarrow \mathbb{R}$ zuordnet,
5. NET ist eine Abbildung, die jedem $u \in U$ eine Netzeingabefunktion $Net_u: (\mathbb{R} \times \mathbb{R})^U \rightarrow \mathbb{R}$ zuordnet, und
6. ex ist eine externe Eingabefunktion $ex: U \rightarrow \mathbb{R}$, die jedem $u \in U$ eine externe Eingabe in der Form einer reellen Zahl $ex_u = ex(u) \in \mathbb{R}$ zuordnet.

2.1.2 Lernmechanismen Neuronaler Netze

Ein Neuronales Netz lernt, indem es sich gemäß einer Lernregel selbst modifiziert. Dabei können zum Beispiel neue Verbindungen erstellt oder bestehende Verbindungen gelöscht werden. Außerdem können die Funktionen der Neuronen verändert werden. In der Regel werden jedoch die Gewichte der Verbindungen angepasst.

Das Netz benötigt zum Lernen bzw. als Voraussetzung keinerlei explizites Wissen über die zu lernende Aufgabe, es lernt lediglich aus der Vorgabe von Datensätzen bestehend aus Ein- und Ausgabewerten. Somit können Neuronale Netze überall dort eingesetzt werden, wo kein mathematisches Prozessmodell bekannt ist oder dessen Umsetzung mit herkömmlichen Reglern hohe Kosten verursachen würde. Gleichzeitig bedeutet dies jedoch auch, dass aus dem trainierten Netz kein gelerntes Wissen extrahiert werden kann. Das Netz zeigt ein "Black-Box-Verhalten", es erfüllt seine Funktion, aber es ist nicht erkennbar, wie es dies erreicht.

Es lassen sich drei Arten von Lernalgorithmen unterscheiden: Überwachtes Lernen (Supervised Learning), Verstärkendes Lernen (Reinforcement Learning) und Unüberwachtes Lernen (Unsupervised Learning).

Beim Überwachten Lernen gibt man dem Netz zu jeder Eingabe die korrekte Ausgabe an. Anhand dieser Angabe wird das Netz dahingehend modifiziert, dass die Differenz zwischen tatsächlicher Ausgabe und der korrekten Ausgabe minimal wird. Dies setzt voraus, dass Trainingsdaten existieren, d.h. dass eine Menge von Datensätzen bestehend aus Eingabedaten und den zugehörigen korrekten Ausgabedaten zum Trainieren des Netzes zur Verfügung steht. Ein typisches überwachtes Lernverfahren ist Backpropagation. Hierbei werden die Eingabedaten ins Netz gegeben, das Netz wird durchlaufen und es erzeugt einen Ausgabevektor. Dieser wird mit dem korrekten Ausgabevektor verglichen und die Differenz wird berechnet. Der Fehler wird rückwärts von der Ausgabe- zur Eingabeschicht durch das Netz geschickt und dabei werden die Verbindungsgewichte so verändert, dass der Fehler verringert wird [Nauck94], [Jang97].

Beim Verstärkenden Lernen wird dem Netz lediglich zurückgegeben, ob seine Ausgabe korrekt war oder nicht. Die Differenz zwischen Ausgabe und korrektem Wert bleibt unbekannt. Auf dieses Lernverfahren soll in Kapitel 5 noch näher eingegangen werden [www02].

Bei der dritten Art des Lernens, dem Unüberwachten Lernen, gibt es überhaupt keine Informationen von außen und das Netz versucht ohne Beeinflussung von außen, die gegebenen Daten in Ähnlichkeitsklassen aufzuteilen.

2.2 Fuzzy-Logik

Die Grundlagen der Fuzzy-Logik wurden 1965 von Lotfi Zadeh mit der Theorie der Fuzzy-Mengen begründet. Hintergrund ist die Überlegung, dass die herkömmliche Logik, bei der jeder logischen Aussage ein Wahrheitswert "wahr" oder "falsch" zugeordnet werden kann, nicht der menschlichen (und erfahrungsgemäß in der Regel sehr effektiven) Denkweise entspricht.

Im täglichen Leben verwenden wir ständig Ausdrücke wie "groß/klein" oder "schnell/langsam" die sich nicht in konkrete Zahlen fassen lassen. So könnte man zwar z.B. festlegen, Menschen über 1,85m Körpergröße seien "groß" und alle anderen "klein", doch würden wir auch einen Menschen mit 1,84m Größe noch als groß bezeich-

nen. Die Grenze zwischen den beiden Gruppen ist *unscharf*. Trotzdem können wir mit diesen unscharfen Aussagen umgehen und z.B. Ratschläge wie “Wenn die Raumtemperatur *zu hoch* wird, drehe das Ventil *etwas zu*” problemlos umsetzen. Um eine solche Aufgabe zu automatisieren, müsste man die Begriffe “zu hoch” und “etwas zudrehen” konkretisieren und als Zahlenwerte festlegen.

Aus diesen Überlegungen ergab sich die Idee der Fuzzy-Logik. Der Begriff “*fuzzy*” lässt sich in etwa mit “*unscharf*” übersetzen und bezieht sich auf die verschwommene Abgrenzung von Zuständen in der menschlichen Logik. Die Methode der Fuzzy-Mengen löst dieses Problem, indem sie die zweiwertige Entscheidung, bei der ein Element entweder vollständig oder überhaupt nicht zu einer Menge gehört, auflöst. Stattdessen werden Zugehörigkeitsgrade zwischen 0 und 1 vergeben.

2.2.1 Begriffe der Fuzzy-Logik

Die Theorie der **Fuzzy-Mengen** bilden die Grundlage der Fuzzy-Logik. Wie oben angedeutet beschreiben die Fuzzy-Mengen die Zugehörigkeit zu einem bestimmten linguistischen Term wie “groß” oder “schnell”. Dabei wird jedoch nicht scharf unterschieden zwischen dazugehörig und nicht dazugehörig, sondern es gibt auch eine graduelle Zugehörigkeit und Eingangswerte können zu mehreren Mengen (z.B. “schnell” und “langsam”) gleichzeitig gehören. Die Fuzzy-Mengen sind definiert durch „Zugehörigkeitsfunktionen“ oder englisch “**membership functions**”, welche zumeist Dreiecks-, Trapez- oder Gaussfunktionen sind und den Wertebereich $[0, 1]$ haben. Einige Beispiele sind in Bild 2.2 dargestellt.

2.2.2 Fuzzy-Regler

Für die Anwendung der Fuzzy-Logik in der Regelungstechnik, werden sogenannte Fuzzy-Regler verwendet. Um im Anwendungsfall mit Fuzzy-Mengen arbeiten zu können, muss der gesamte mögliche Wertebereich einer Eingangsvariablen eines Fuzzy-Systems durch Fuzzy-Mengen verschiedener linguistischer Terme abgedeckt sein. Diese überschneiden sich in der Regel, grenzen jedoch mindestens aneinander an. Diese Aufteilung nennt sich “Fuzzy Partitioning”.

Abbildung 2.3 zeigt beispielhaft die Partitionierung für die Eingangsvariable “Alter” mithilfe von dreieckförmigen Zugehörigkeitsfunktionen. In der Abbildung wird deutlich, dass es mithilfe der Fuzzy-Mengen möglich ist, dass ein Eingangswert zu zwei Mengen gleichzeitig gehört. Das Alter “30” z.B. gehört sowohl zur Menge “jung” als auch zur Menge “nicht jung”.

Soll die Fuzzy-Logik in einem technischen System angewendet werden, so erhält das System als Eingangsgrößen zunächst meistens scharfe Eingangswerte. Um diese mithilfe der Fuzzy-Logik weiter zu verarbeiten ist zunächst die „**Fuzzifizierung**“ der

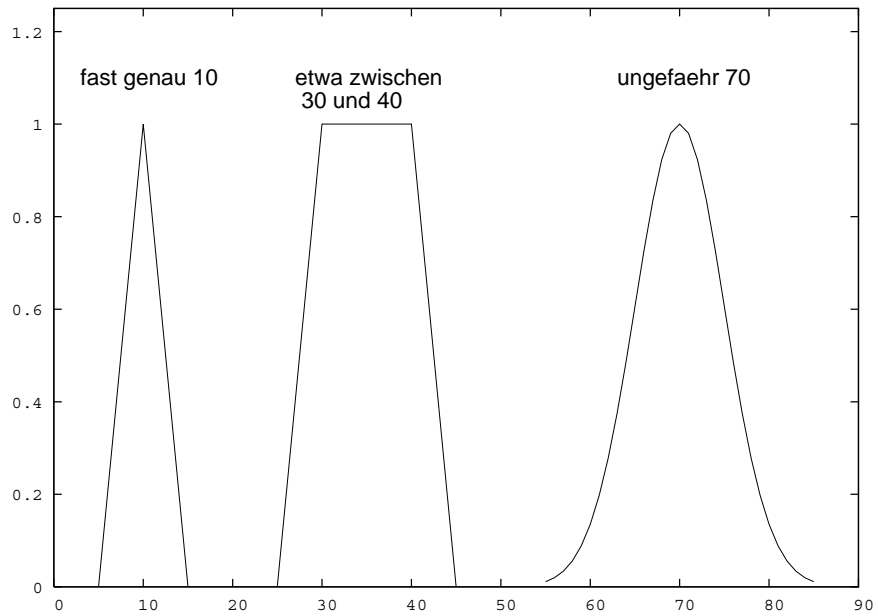


Abbildung 2.2: Darstellung der linguistischen Werte "fast genau 10", "etwa zwischen 30 und 40" und "ungefähr 70" durch Zugehörigkeitsfunktionen

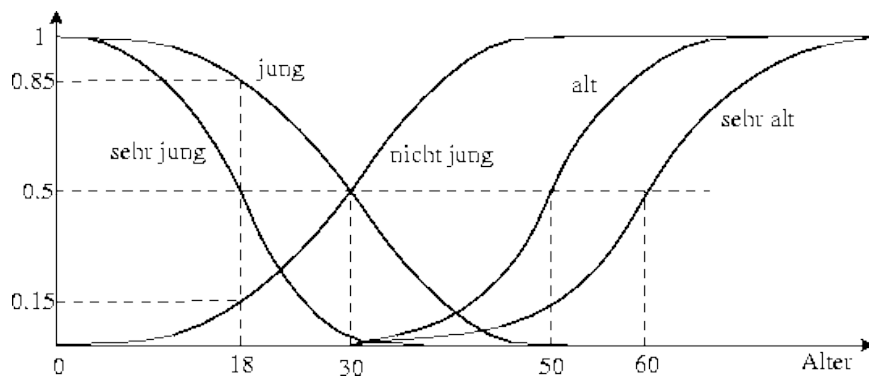


Abbildung 2.3: Fuzzy-Partitionierung der Eingangsvariable „Alter“

Eingangsgrößen nötig. Dabei werden über die Fuzzy-Mengen die Zugehörigkeitswerte der Variable zu den einzelnen linguistischen Termen des Eingangsraumes berechnet. Die Zugehörigkeitswerte bewegen sich zwischen „1“ (gehört ganz dazu) und „0“ (gehört nicht dazu).

Als nächster Schritt müssen die fuzzifizierten Eingangswerte verarbeitet und ein Ausgangswert berechnet werden. Diesen Vorgang nennt man **„Fuzzy-Inferenz“**. Die Fuzzy-Inferenz bildet den Kern des Fuzzy-Systems und hier zeigen sich die Unterschiede zwischen verschiedenen Modellen. Es soll zunächst kurz allgemein beschrieben werden, was in diesem Schritt geschieht:

Das Fuzzy-System beinhaltet einen Satz von linguistischen Regeln in der Form **„Wenn** die Geschwindigkeit *groß* ist **und** der Abstand zum vorausfahrenden Fahrzeug *gering* ist, **dann** *stark* bremsen“. Mithilfe solcher Wenn...dann-Regeln lässt sich ein System vollständig beschreiben. Im Fall mehrere Regelprämissen (wie im oben erwähnten Beispiel) müssen zunächst die linguistischen Werte logisch miteinander verknüpft werden. Die geschieht mithilfe eines t-Norm-Operators. Ein t-Norm-Operator dient zur konjunktiven Verknüpfung zweier Fuzzy-Mengen. Meistens verwendet man als t-Norm-Operator einen MIN-Operator [www09]. Durch die Verknüpfung der Regelprämissen mit Hilfe eines t-norm-Operators erhält man für jede Regel eine Konklusion.

Abbildung 2.4 zeigt ein Beispiel für die Verknüpfung der Prämissen. Hierbei werden die Zugehörigkeitsfunktionen UND-verknüpft (MIN-Operator)

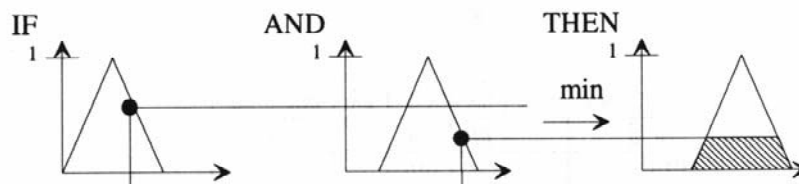


Abbildung 2.4: UND-Verknüpfung (MIN-Operator) von zwei Regelprämissen

Um nun aus den Schlussfolgerungen mehrerer Regeln zu einem endgültigen Ergebnis zu kommen, müssen die einzelnen Ergebnisse noch geeignet zusammengefasst werden. Die geschieht mit einem t-Conorm-Operator. Dieser ist wie der t-Norm-Operator ein Operator zur disjunktiven Verknüpfung zweier Fuzzy-Mengen. Am häufigsten wird als t-Conorm-Operator der MAX-Operator verwendet. Abbildung 2.5 zeigt den gesamten Vorgang zur Auswertung der Regeln. Die Regelkonklusionen werden hier mithilfe des MAX-Operators verknüpft. Es ergibt sich eine neue Zugehörigkeitsfunktion, die das Ausgangssignal beschreibt [Nauck94].

Um schließlich die Entscheidung des Fuzzy-Systems für eine technische Anwendung nutzen zu können, muss das „unscharfe“ Ausgangssignal in einen konkreten Wert umgewandelt werden. Diesen Vorgang nennt man **„Defuzzifizierung“**. Hierbei wird die Fuzzy-Menge des Ausgangs in einen repräsentativen Wert umgewandelt.

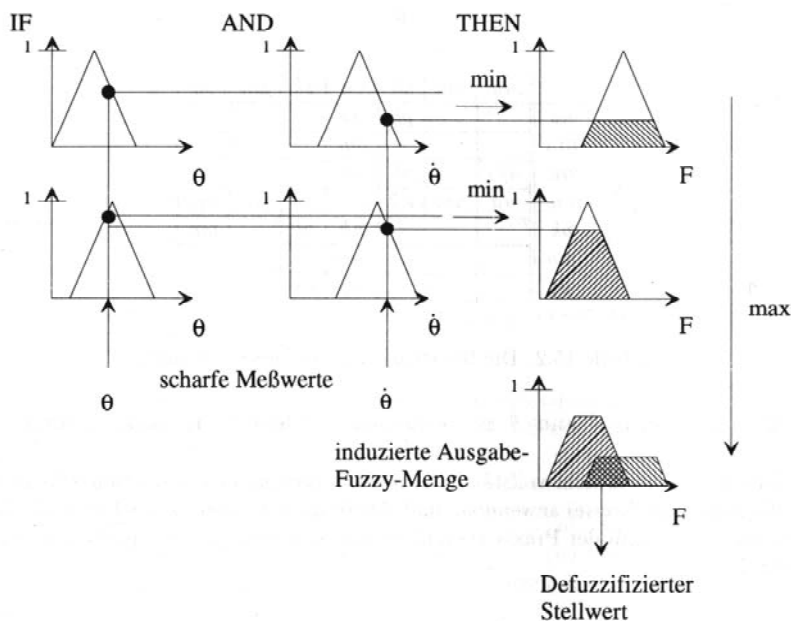


Abbildung 2.5: Verknüpfung von Regelkonklusionen zu einer Ausgabe-Fuzzy-Menge, [Nauck94]

Es gibt verschiedene Defuzzifizierungsmethoden. Häufig wird die “Center of Gravity” (COG) Methode verwendet. Hier wird der Schwerpunkt der Fläche unter der Zugehörigkeitsfunktion des Ausgangs bestimmt und seine x-Koordinate wird als Stellwert ausgegeben. Eine weitere Methoden ist die “Mean of Maximum” (MOM) Methode, welche den Mittelwert aus allen Werten, bei denen die Fuzzy-Menge des Ausgangs einen maximalen Zugehörigkeitsgrad liefert, als Stellwert berechnet [Jang97], [www01], [www03]. In Bild 2.6 sind die Ergebnisse für verschiedene Defuzzifizierungsmethoden eingezeichnet.

Zusammengefasst sieht ein Fuzzy-Regler also wie in Abbildung 2.7 dargestellt aus: Signale von außen werden fuzzifiziert, anschließend findet unter Verwendung der Regelbasis die Fuzzy-Inferenz statt und das hieraus gewonnene Ergebnis wird defuzzifiziert.

Im Folgenden sollen zwei Typen von Fuzzy-Reglern beschrieben werden: der Mamdani-Regler und der Sugeno-Regler. Sie werden in der später vorgestellten Software NFident verwendet. Einen Überblick über weitere Typen von Fuzzy-Reglern findet sich z.B. in [Lee90] und [Dria93].

Bei der Beschreibung der beiden Fuzzy-Regler sollen folgende Konventionen gelten: Wir betrachten ein System mit n Messgrößen $\xi_1 \in X_1, \dots, \xi_n \in X_n$ und einer Stellgröße $\eta \in Y$. Die Regelbasis besteht aus k linguistischen Regeln R_1, \dots, R_k .

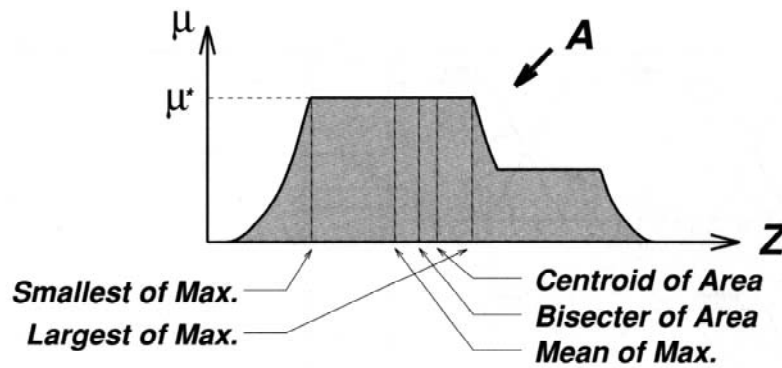


Abbildung 2.6: Aus einer Fuzzy-Menge gewonnener Stellwert bei verschiedenen Defuzzifizierungsmethoden, [Jang97]

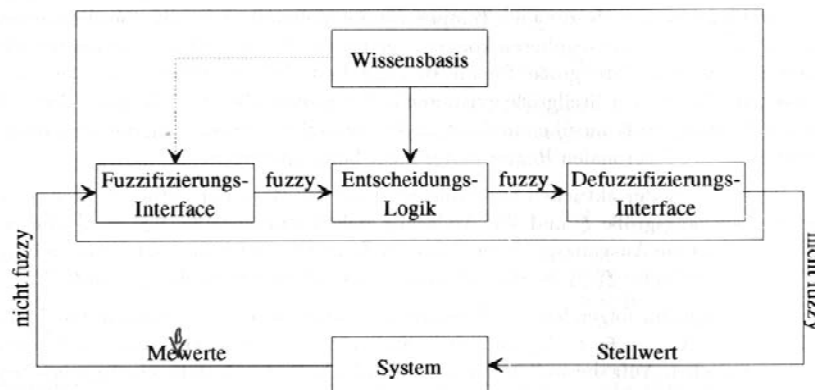


Abbildung 2.7: Architektur eines Fuzzy-Reglers, [Nauck94]

2.2.2.1 Mamdani-Regler

Beim Mamdani-Regler haben die linguistischen Regeln die folgende Form:

$$R_r : \text{If } \xi_1 \text{ is } A_{j_1,r}^{(1)} \text{ and } \dots \text{ and } \xi_n \text{ is } A_{j_n,r}^{(n)} \text{ then } \eta \text{ is } B_{i_r} \quad (r = 1, \dots, k) \quad (2.1)$$

Dabei sind $A_{j_1,r}^{(1)}, \dots, A_{j_n,r}^{(n)}$ und B_{i_r} linguistische Terme der Mess- und Stellgrößen.

Die Mengen X_1, \dots, X_n und Y werden durch die Fuzzy-Mengen $\mu_1^{(1)}, \dots, \mu_{p_1}^{(1)}, \dots, \mu_n^{(1)}, \dots, \mu_{p_n}^{(n)}$ und ν_1, \dots, ν_q partitioniert. (p ist die Anzahl der Fuzzy-Mengen je Messgröße, q die Anzahl der Fuzzy-Mengen der Stellgröße).

Jeder Fuzzy-Menge wird ein linguistischer Term zugeordnet, so dass die Mengen X_1, \dots, X_n und Y sinnvoll unterteilt sind.

Ein gegebenes Messwert-Tupel (x_1, \dots, x_n) wird nun mit den Prämissen der einzelnen Regeln verknüpft, indem für jeden Messwert der Zugehörigkeitsgrad zur entsprechenden Fuzzy-Menge ermittelt wird.

Die Werte müssen nun geeignet konjunktiv verknüpft werden um den Erfüllungsgrad τ_r der Prämisse der Regel R_r zu ermitteln. Beim Mamdani-Regler wird für diese Verknüpfung ein MIN-Operator verwendet:

$$\tau_r = \min \left\{ \mu_{i_1,r}^{(1)}(x_1), \dots, \mu_{i_n,r}^{(n)}(x_n) \right\} \quad (2.2)$$

Die Regelkonklusion lässt sich somit in Form einer Fuzzy-Menge darstellen als

$$\nu_{x_1, \dots, x_n}^{output(R_r)} : Y \rightarrow [0, 1], \quad y \mapsto \min \left\{ \mu_{i_1,r}^{(1)}(x_1), \dots, \mu_{i_n,r}^{(n)}(x_n), \nu_{i_r}(y) \right\} \quad (2.3)$$

Schließlich muss noch die Gesamtausgabe des Reglers berechnet werden. Dies geschieht durch eine MAX-Verknüpfung der Konklusionen der einzelnen Regeln:

$$\nu_{x_1, \dots, x_n}^{output} : Y \rightarrow [0, 1], \quad y \mapsto \max_{r \in \{1, \dots, k\}} \min \left\{ \mu_{i_1,r}^{(1)}(x_1), \dots, \mu_{i_n,r}^{(n)}(x_n), \nu_{i_r}(y) \right\} \quad (2.4)$$

Aufgrund der Verwendung des MAX- und MIN-Operationen wird der Mamdani-Regler auch Max-Min-Regler genannt. Es könnten stattdessen auch beliebige t-Normen und t-Conormen verwendet werden.

Schließlich muss noch die Fuzzy-Menge $\nu_{x_1, \dots, x_n}^{output}$ in einen scharfen Ausgabewert η in Y umgewandelt werden. Dazu wird ein beliebiges Defuzzifizierungsverfahren verwendet.

2.2.2.2 Sugeno-Regler

Der Sugeno-Regler (auch TSK-Regler genannt) ist eine Variante des Mamdani-Reglers. Im Unterschied zu diesem haben die linguistischen Regeln beim Sugeno-Regler jedoch die Form

$$R_r : \text{If } \xi_1 \text{ is } A_{j_1,r}^{(1)} \text{ and } \dots \text{ and } A_{j_n,r}^{(n)} \text{ Then } \eta = f_r(\xi_1, \dots, \xi_n) \quad (r = 1, \dots, k) \quad (2.5)$$

Das heißt, hier werden nur X_1, \dots, X_n durch Fuzzy-Mengen partitioniert. Zur Bestimmung des Stellwertes wird eine Abbildung f_r von $X_1 \times \dots \times X_n$ nach Y ($r = 1, \dots, k$) verwendet. In der Regel hat f_r die Form

$$f_r(x_1, \dots, x_n) = a_1^{(r)} \cdot x_1 + \dots + a_n^{(r)} \cdot x_n + a^{(r)} \quad (2.6)$$

Die Vorgehensweise ist ansonsten die gleiche wie beim Mamdani-Regler. Bei der Berechnung des Erfülltheitsgrades τ_r einer Regel wird hier zur Verknüpfung der Prämissen der Regel eine MIN- oder Produktoperation verwendet.

Im Gegensatz zum Mamdani-Regler erhält man beim Sugeno-Regler aus der Verknüpfung der Ausgaben der Regeln direkt einen scharfen Wert η . Die Defuzzifizierung entfällt also. Der Stellwert berechnet sich als eine mit dem Erfülltheitsgrad der Regeln gewichtete Summe:

$$\eta = \frac{\sum_{r=1}^k \tau_r \cdot f_r(x_1, \dots, x_n)}{\sum_{r=1}^k \tau_r} \quad (2.7)$$

2.3 Neuro-Fuzzy-Systeme

Neuro-Fuzzy-Systeme sind Systeme, die die Methoden der Fuzzy-Logik und der Neuronalen Netze miteinander kombinieren. Ziel ist es dabei, ein effektiveres und anpassungsfähigeres System zu modellieren, als es mit nur einer der beiden Methoden möglich wäre. So ist es ein großer Nachteil von Fuzzy-Systemen, dass man in ihnen zwar bereits vorhandenes Expertenwissen modellieren kann, sie jedoch im Weiteren nicht lernfähig sind. Neuronale Netze hingegen sind lernfähig, haben aber den Nachteil, dass weder bereits vorhandenes Wissen in das Modell integriert werden kann, noch das erlernte Wissen aus dem Modell extrahiert werden kann. Durch die Kombination der beiden Methoden soll also ein lernfähiges System entstehen, in dem man bei der Erstellung Expertenwissen integrieren kann und dessen erlernte Funktion ohne größeren Aufwand extrahierbar ist.

Zur Kombination von Neuronalen Netzen und Fuzzy-Systemen lassen sich allgemein zwei Ansätze unterscheiden. Bei den als **“Kooperative Neuro-Fuzzy-Systeme”** bezeichneten Modellen arbeiten ein Neuronales Netz und ein Fuzzy-System grundsätzlich unabhängig voneinander. Sie werden dadurch gekoppelt, dass einige Parameter des Fuzzy-Systems vom Neuronalen Netz erzeugt oder während des Einsatzes optimiert werden. Beim den als **“Hybride Neuro-Fuzzy-Systeme”** bezeichneten Modellen bilden Fuzzy-Systeme und Neuronales Netz eine Einheit, die sich meistens an der Struktur des Neuronalen Netzes anlehnt. So kann z.B. das Fuzzy-System als spezielles

Neuronales Netz interpretiert werden oder mit Hilfe eines Neuronalen Netzes implementiert werden.

Es gibt vielfältige Ansätze Neuro-Fuzzy-Systeme zu modellieren. Im Folgenden sollen jedoch nur zwei Modelle aus der Gruppe der "Hybriden Neuro-Fuzzy-Modelle" betrachtet werden, welche in der Software NFident implementiert sind. Weitere Ansätze für Neuro-Fuzzy-Netze werden in [Nauck94] beschrieben.

Kapitel 3

Die Software NFident

NFident ist eine Software zur Funktionsapproximation mit Fuzzy-Systemen, basierend auf überwachtem Lernen (supervised learning). NFident implementiert den an der Universität Magdeburg entwickelten NEFPROX-Ansatz und den ANFIS-Ansatz von J.-S. R. Jang. Mit NFident können mithilfe des NEFPROX-Modells Neuro-Fuzzy-Systeme nach dem Prinzip des Mamdani-Reglers und mithilfe des ANFIS-Modells solche nach dem Prinzip des Sugeno-Reglers realisiert werden. Sowohl die Regelbasis als auch die Zugehörigkeitsfunktionen können gelernt werden.

Die beiden implementierten Modelle NEFPROX und ANFIS sollen hier zunächst in Abschnitt 3.2 und 3.1 vorgestellt werden. In Abschnitt 3.3 wird dann näher auf die verwendete Software eingegangen.

3.1 Neuro-Fuzzy Function Approximation (NEFPROX)

Das NEFPROX-Modell wurde 1997 an der Universität Magdeburg entwickelt. Es basiert auf dem 1994 ebenfalls an der Universität Magdeburg entwickelten Modell Neuro-Fuzzy-Controller (NEFCON), ist jedoch allgemeiner verwendbar. Die Funktion eines NEFPROX-Systems basiert auf dem in Abschnitt 2.2.2.1 beschriebenen Fuzzy-Regler vom Typ Mamadani und kann als ein Spezialfall eines dreischichtigen Perceptrons aufgefasst werden. Abbildung 3.1 zeigt die Struktur eines solchen NEFPROX-Systems, hier mit zwei Eingangsgrößen und fünf Regeln.

Ein NEFPROX-System besteht aus drei Schichten:

- Die Einheiten der **Eingabeschicht** nehmen im Normalfall keine weitere Verarbeitung vor.
- Die **innere Schicht** des Systems repräsentiert die Regeln des Systems. Ihre Aktivierungen entsprechen dem Erfüllungsgrad ihrer Prämissen.

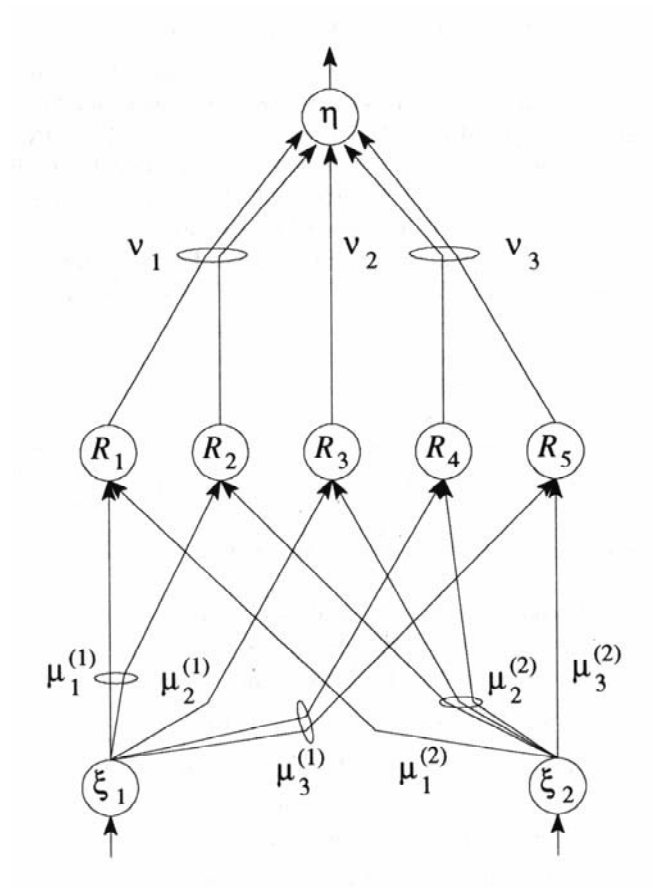


Abbildung 3.1: Struktur eines NEFPROX-Systems mit zwei Eingangsgrößen und fünf Regeln, [Nauck94]

- Die **Ausgabeschicht** besteht aus einem Neuron, welches einen scharfen Ausgabewert liefert.

Das NEFPROX-Modell zeigt vor allem in der Verbindung zwischen den einzelnen Schichten einen deutlichen Unterschied zu herkömmlichen Neuronalen Netzen: zunächst gibt es Verbindungen, denen ein gemeinsames Gewicht zugeordnet ist (in Abbildung 3.1 z.B. die Verbindungen vom Eingabeneuron ξ_1 zu den inneren Neuronen R_1 und R_2) Diese gemeinsame Gewichtung muss vom Lernalgorithmus berücksichtigt werden. Des Weiteren sind die Verbindungsgewichte im NEFPROX-Modell keine Zahlenwerte, sondern Fuzzy-Mengen. Die Gewichte der Verbindungen zwischen Eingabeschicht und innerer Schicht repräsentieren die Fuzzy-Mengen der Regelprämissen, die Verbindungen zwischen innerer Schicht und Ausgabeneuron stehen für die Fuzzy-Mengen der Regelkonklusionen. Somit wird also die Regelbasis durch die Struktur des Netzes umgesetzt. In Abbildung 3.1 z.B. lautet die Regel R_3

$$R_1 : \text{If } \xi_1 \text{ is } A_2^{(1)} \text{ and } \xi_2 \text{ is } A_2^{(2)} \text{ Then } \eta \text{ is } B_2 \quad (3.1)$$

Dabei sind $A_2^{(1)}$ und $A_2^{(2)}$ die linguistischen Terme, die durch $\mu_2^{(1)}$ und $\mu_2^{(2)}$ repräsentiert werden, B_2 ist der durch ν_2 repräsentierte linguistische Term.

Das NEFPROX-Modell verwendet für den Lernvorgang einen Algorithmus zum überwachten Lernen. D.h. es wird eine Menge von Trainingsdatensätzen vorgegeben, die zum Lernen verwendet werden können. Des Weiteren können, wenn entsprechendes Wissen vorhanden ist, Regeln für das System vorgegeben werden. Das Netz wird dann schrittweise mit den Trainingsdatensätzen trainiert und es werden sowohl die Fuzzy-Mengen als auch die Regelmenge angepasst.

Da die NFident-Software hauptsächlich den NEFPROX-Ansatz implementiert, werden im folgenden Abschnitt die Definitionen des NEFPROX-Modells und der verwendeten Lernalgorithmen angegeben. Die Definitionen sind aus [www05] übernommen, wo das NEFPROX-Modell detailliert vorgestellt wird.

3.1.1 NEFPROX - Definition

Das NEFPROX-Modell basiert auf dem Modell des dreischichtigen Fuzzy-Perceptrons. Dieses ist wie folgt definiert:

Definition 3.1.1 *Ein dreischichtiges Fuzzy-Perceptron ist ein dreischichtiges, vorwärtsbetriebenes Neuronales Netz (U, W, NET, A, O, ex) mit den folgenden Spezifikationen:*

1. $U = \cup_{i \in M} U_i$ ist eine nicht-leere Menge von Verarbeitungseinheiten (Neuronen), und $M = 1, 2, 3$ heißt Indexmenge von U .
Es gilt für alle $i, j \in M$, $U_i \neq \emptyset$ und $U_i \cap U_j = \emptyset$ für $i \neq j$.
 U_1 heißt Eingabeschicht, U_2 Regelschicht und U_3 Ausgabeschicht.

2. Die Netzwerkstruktur ist $W : U \times U \rightarrow F(\mathbb{R})$, wobei ausschließlich Verbindungen $W(u, v)$ mit $u \in U_i, v \in U_{i+1}$ ($i \in 1, 2$) existieren.
3. A ordnet jedem $u \in U$ eine Aktivierungsfunktion A_u zur Berechnung der Aktivierung a_u zu. Es gilt

(a) für Eingabeeinheiten und Regeleinheiten $u \in U_1 \cup U_2$:

$$A_u : \mathbb{R} \rightarrow \mathbb{R}, a_u = A_u(\text{net}_u) = \text{net}_u,$$

(b) für Ausgabeeinheiten $u \in U_3$:

$$A_u : F(\mathbb{R}) \rightarrow F(\mathbb{R}), a_u = A_u(\text{net}_u) = \text{net}_u,$$

4. O ordnet jeder Einheit $u \in U$ eine Ausgabefunktion O_u zur Berechnung der Ausgabe o_u zu. Es gilt

(a) für Eingabeeinheiten und Regeleinheiten $u \in U_1 \cup U_2$:

$$O_u : \mathbb{R} \rightarrow \mathbb{R}, o_u = O_u(a_u) = a_u,$$

(b) für Ausgabeeinheiten $u \in U_3$:

$$O_u : F(\mathbb{R}) \rightarrow \mathbb{R}, o_u = O_u(\text{net}_u) = \text{DEFUZZ}_u(\text{net}_u),$$

dabei ist DEFUZZ_u ein geeignetes Defuzzifizierungsverfahren.

5. NET ordnet jeder Einheit $u \in U$ eine Netzeingabefunktion NET_u zur Berechnung der Netzeingabe net_u zu. Es gilt:

(a) für Eingabeeinheiten $u \in U_1$:

$$NET_u : \mathbb{R} \rightarrow \mathbb{R}, \text{net}_u = ex_u,$$

(b) für innere Einheiten $u \in U_2$:

$$NET_u : (\mathbb{R} \times F(\mathbb{R}))^{U_1} \rightarrow [0, 1], \text{net}_u = \top_{u' \in U_1} (W(u', u)(a_{u'})),$$

dabei ist \top eine t -Norm,

(c) für Ausgabeeinheiten $u \in U_3$:

$$NET_u : ([0, 1] \times F(\mathbb{R}))^{U_2} \rightarrow F(\mathbb{R}), \text{net}_u = \mathbb{R} \rightarrow [0, 1], \text{net}_u(x) = \perp_{u' \in U_2} (\top(o_{u'}, u)(x)),$$

dabei ist \perp eine t -Conorm.

Falls die Fuzzy-Mengen $W(u', u)$, $u' \in U_2$, $u \in U_3$ über ihren Trägern monoton sind, und $W^{-1}(u', u)(\tau)$ das $x \in \mathbb{R}$ bestimmt, für das $W(u', u)(x) = \tau$ gilt, so kann alternativ für eine Ausgabeeinheit $u \in U_3$ gelten:

$$\text{net}_u(x) = \begin{cases} 1 & \text{falls } x = \frac{\sum_{u' \in U_2} a_{u'} \cdot W^{-1}(u', u)(u_{u'})}{\sum_{u' \in U_2} o_{u'}} \\ 0 & \text{sonst} \end{cases}$$

Für die Ausgabe o_u gilt dann abweichend von (4(b))

$$o_u = x, \text{ mit } \text{net}_u(x) = 1.$$

6. ex ist eine externe Eingabefunktion $ex: U_1 \rightarrow \mathbb{R}$ die jedem $u \in U$ eine externe Eingabe in Form einer reellen Zahl $ex_u = ex(u) \in \mathbb{R}$ zuordnet.

Ein NEFPROX-System kann als ein Spezialfall eines dreischichtigen Fuzzy-Perceptrons aufgefasst werden. Es gilt folgende Definition:

Definition 3.1.1 *Ein NEFPROX-System ist ein dreischichtiges Fuzzy-Perceptron mit folgenden Spezifikationen:*

1. Die Eingabeeinheiten werden mit x_1, \dots, x_n , die inneren Regeleinheiten mit R_1, \dots, R_k , und die Ausgabeeinheiten mit y_1, \dots, y_m bezeichnet.
2. Jede Verbindung zwischen Einheiten x_i und R_r ist mit einem linguistischen Term $A_{k_r}^{(i)}$ ($k_r \in \{1, \dots, p_i\}$) benannt.
3. Jede Verbindung zwischen Einheiten R_r und einer Ausgabeeinheit y_j ist mit einem linguistischen Term $B_{k_r}^{(j)}$ ($k_r \in \{1, \dots, q_j\}$) benannt.
4. Verbindungen, die von derselben Eingabeeinheit x_i wegführen und identische Bezeichnungen tragen, besitzen zu jeder Zeit dasselbe Fuzzy-Gewicht. Diese Verbindungen heißen gekoppelt und ihre Gewichte heißen gemeinsame Gewichte. Eine analoge Beziehung gilt für Verbindungen, die zu derselben Ausgabeeinheit y_j führen.
5. Sei $L_{x,R}$ die Bezeichnung der Verbindung zwischen einer Einheit x und einer Regeleinheit R . Für alle Regeleinheiten R, R' gilt $(\forall x L_{x,R} = L_{x,R'}) \Rightarrow R = R'$.

Die Fuzzy-Mengen des NEFPROX-Systems können verschiedene Formen haben. Gängig sind trapez- oder dreiecksförmige Kurven sowie gaußsche Glockenkurven. Die jeweiligen Kurven werden durch die folgenden Gleichungen beschrieben:

$$\text{Dreieck}(x; a, b, c) = \begin{cases} 0, & x \leq a \\ \frac{x-a}{b-a}, & a \leq x \leq b \\ \frac{c-x}{c-b}, & b \leq x \leq c \\ 0, & c \leq x \end{cases} \quad (3.2)$$

Dabei stehen die Parameter a, b und c für die x -Werte der linken Grenze des Dreiecks (a), des Hochpunktes (b) und der rechten Grenze (c).

Die Trapezfunktion wird beschrieben durch

$$\text{Trapez}(x; a, b, c, d) = \begin{cases} 0, & x \leq a \\ \frac{x-a}{b-a}, & a \leq x \leq b \\ 1, & b \leq x \leq c \\ \frac{c-x}{c-b}, & c \leq x \leq d \\ 0, & d \leq x \end{cases} \quad (3.3)$$

Hierbei ist a die x -Koordinate der linken Grenze des Trapezes. b und c sind der linke und rechte x -Wert des Maximums und d ist die x -Koordinate der rechten Grenze.

Schließlich ist die Gauss-Funktion gegeben durch

$$\text{Gauss}(x; a, b) = e^{\frac{1}{2}(\frac{x-a}{b})^2} \quad (3.4)$$

Dabei ist a das Zentrum der Gauss-Funktion und b bestimmt die Breite der Glockenkurve.

Die Parameter a, b, c und d werden im Verlauf des Lernens verändert, wodurch die Fuzzy-Mengen angepasst werden.

Es soll nun das Lernverfahren des NEFPROX-Modells definiert werden. Es teilt sich in zwei Abschnitte: der Algorithmus zum Lernen der Regelbasis (Strukturlernalgorithmus) und der Algorithmus zum Anpassen der Fuzzy-Mengen (Parameterlernalgorithmus).

Definition 3.1.1 *Betrachtet wird ein NEFPROX-System mit n Eingabeeinheiten x_1, \dots, x_n , k Regeleinheiten R_1, \dots, R_k und m Ausgabeeinheiten y_1, \dots, y_m . Weiterhin ist eine Lernaufgabe $L = \{(s_1, t_1), \dots, (s_r, t_r)\}$ mit r Mustern gegeben, wobei jedes Muster aus einem Eingabemuster $s \in \mathbb{R}^n$, und einem Zielmuster $t \in \mathbb{R}^m$ (erwünschte Ausgabe) besteht. Der Lernalgorithmus der k Regeleinheiten des NEFPROX-Systems (Strukturlernalgorithmus) besteht aus den folgenden Schritten:*

1. Wähle das nächste Muster (s, t) aus L
2. Finde für jede Eingabeeinheit $x_i \in U_1$ die Zugehörigkeitsfunktion $\mu_{j_i}^{(i)}$ mit

$$\mu_{j_i}^{(i)}(s_i) = \max_{j \in \{1, \dots, p_i\}} \left\{ \mu_j^{(i)}(s_i) \right\}$$

3. Falls keine Regeleinheit R mit

$$W(x_1, R) = \mu_{j_1}^{(1)}, \dots, W(x_n, R) = \mu_{j_n}^{(n)}$$

existiert, dann erzeuge eine solche Einheit und verbinde sie mit allen Ausgabeeinheiten.

4. Finde für jede Verbindung von der neuen Regeleinheit zu den Ausgabeeinheiten ein geeignetes Fuzzy-Gewicht durch das folgende Verfahren:

Aus allen Zugehörigkeitsfunktionen, die zur Ausgabeeinheit y_i gehören wähle $\nu_{j_i}^{(i)}$, so dass

$$\nu_{j_i}^{(i)}(t_i) = \max_{j \in \{nu_j^{(i)}(t_i)\}}, \text{ und } \nu_{j_i}^{(i)}(t_y) \geq 0.5$$

gilt. Gibt es keine derartige Fuzzy-Menge, dann erzeuge $\nu_{neu}^{(i)}$ so dass $\nu_{neu}^{(i)}(t_i) = 1$ gilt, füge sie zu den y_i zugeordneten Zugehörigkeitsfunktionen hinzu, und setze $W(R, y_i) = \nu_{neu}^{(i)}$.

5. Falls es noch nicht verarbeitete Muster in L gibt, dann fahre mit Schritt (1) fort, ansonsten beende die Regelerzeugung.
6. Evaluieren die Regelbasis abschließend: Ermittle für jede Regel einen Mittelwert der Ausgaben aller Muster, die einen Treffergrad größer Null zu der jeweiligen Regel aufweisen. Ändere gegebenenfalls die entsprechende Ausgabe-Fuzzy-Menge in der Konklusion der jeweiligen Regel, falls der Mittelwert einen höheren Zugehörigkeitsgrad zu einer anderen als der aktuellen Ausgabe-Fuzzy-Menge aufweist.

Der überwachte Lernalgorithmus zur Adaption der Fuzzy-Mengen des NEFPROX-Systems (Parameterlernalgorithmus) läuft zyklisch durch die Lernaufgabe L und wiederholt die folgenden Schritte, bis ein gegebenes Abbruchkriterium erfüllt ist.

1. Wähle das nächste Muster (s, t) aus L , propagiere es durch das NEFPROX-System und bestimme den Ausgabevektor.
2. Für jede Ausgabeeinheit $y_i \in U_3$: Bestimme den Deltawert $\delta_{y_i} = t_i - o_{y_i}$
3. Für jede Regeleinheit R mit $o_r > 0$:
 - (a) Bestimme für alle $y_i \in U_3$ die Deltawerte für die Parameter a, b, c, d der Fuzzy-Menge $W(R, y_i)$ unter Verwendung einer Lernrate $\sigma > 0$:

$$\begin{aligned} \delta_{b_i} &= \sigma \cdot \delta_{y_i} \cdot (c - a) \cdot o_R \cdot (1 - W(R, y_i)(t_i)), \\ \delta_{a_i} &= \sigma \cdot (c - a) \cdot o_R + \delta_{b_i}, \\ \delta_{c_i} &= -\sigma \cdot (c - a) \cdot o_R + \delta_{b_i}, \end{aligned}$$

und wende die Änderungen auf $W(R, y_i)$ an, falls dies nicht gegen eine gegebene Menge Φ von Einschränkungen verstößt. (Beachte: Das Gewicht $W(R, y_i)$ kann mit anderen Verbindungen geteilt und in diesem Fall mehr als einmal verändert werden.)

(b) Bestimme den Deltawert

$$\delta_R = o_R(1 - o_R) \cdot \sum_{y \in U_3} (2W(R, y)(t_i) - 1) \cdot |\delta_y|$$

(c) Bestimme für alle $x \in U_1$ die Deltawerte für die Parameter a, b, c, d der entsprechenden Fuzzy-Menge $W(x, R)$, falls $W(x, R)(o_x) > 0$ gilt:

$$\begin{aligned} \delta_b &= \sigma \cdot \delta_R \cdot (c - a) \cdot (1 - W(x, R)(o_x)) \cdot \text{sgn}(o_x - b) \\ \delta_a &= -\sigma \cdot \delta_R \cdot (c - a) \cdot (1 - W(x, R)(o_x)) + \delta_b, \\ \delta_c &= \sigma \cdot \delta_R \cdot (c - a) \cdot (1 - W(x, R)(o_x)) + \delta_b, \end{aligned}$$

Dabei ist $\sigma > 0$ eine Lernrate. Wende diese Änderung auf $W(x, r)$ an, sofern dies nicht gegen gegebene Einschränkungen Φ verstößt. (Beachte: Das Gewicht $W(x, R)$ kann mit anderen Verbindungen geteilt und in diesem Fall mehr als einmal verändert werde.)

4. Prüfe am Ende einer Epoche, ob das Abbruchkriterium erfüllt ist und halte in diesem Fall an. Andernfalls fahre mit Schritt (1) fort.

3.2 Adaptive Neuro Fuzzy Inference System (ANFIS)

Das ANFIS-Modell beschreibt ein feedforward Netz mit fünf Schichten, wobei die Verbindungen nicht gewichtet werden. Die Logik des Netzes ist somit ausschließlich in den Neuronen und der Netztopologie enthalten. Das Netz kann die Funktionalität eines Sugeno-Reglers, wie in Abschnitt 2.2.2.2 beschrieben, bereitstellen. Abbildung 3.2 zeigt die zugrunde liegende Struktur eines ANFIS-Netzes.

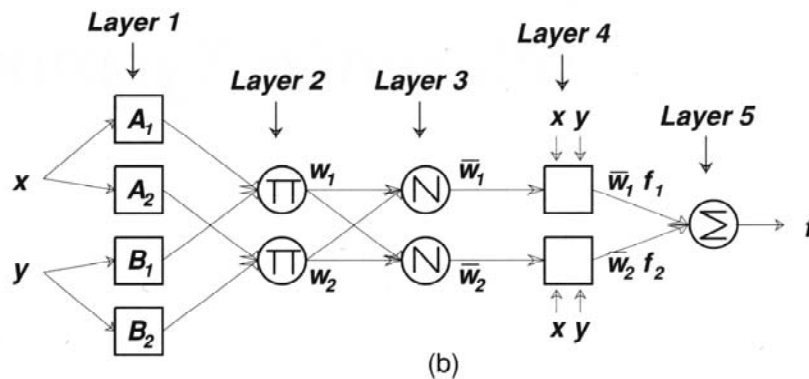


Abbildung 3.2: Struktur eines ANFIS-Systems, [Jang97]

Die in 3.2 als Vierecke dargestellten Neuronen der ersten Schicht ordnen den Eingabewerten die Zugehörigkeit zu einem linguistischen Ausdruck zu. Diese Schicht übernimmt dementsprechend also die Fuzzifizierung der Eingabewerte. In der zweiten Schicht gibt es für jede Regel des Fuzzy-Reglers ein Neuron, welches die Regelprämisse bilden. In diesen Neuronen werden alle eingehenden Zugehörigkeitsfunktionen multipliziert. Die dritte Schicht ist vollständig mit der zweiten verbunden und hat ebenso viele Neuronen wie die Vorgängerschicht. Die Neuronen der dritten Schicht berechnen den relativen Erfüllungsgrad der Regel bezogen auf alle anderen Regeln. Die Neuronen der vierten Schicht (in Abbildung 3.2 viereckig dargestellt) haben eine Verbindung zu einem Neuron der vorhergehenden Schicht und Verbindungen zu allen Eingabeneuronen. Diese Schicht berechnet die gewichtete Ausgabe einer Regel. Schicht 5 besteht aus einem einzigen Neuron, welches den Ausgangswert des Reglers als Summe aller Ausgaben von Schicht 4 berechnet. Die Schichten 1 und 4 können während des Lernvorganges angepasst werden, Schicht 2, 3 und 5 sind unveränderlich [Jang97], [www04].

Das ANFIS-Modell wird in [Jang97] ausführlich dargestellt.

3.3 Die NFident Software

Die Software NFident implementiert parallel sowohl das NEFPROX- als auch das ANFIS-Modell. Ausgelegt ist sie jedoch vor allem für NEFPROX-Modelle. Der in Abschnitt 3.1.1 vorgestellte Lernalgorithmus ist implementiert.

Die Software ermöglicht es dem Nutzer ein Netz zu trainieren. Dabei kann entweder anhand von Trainingsdaten ein neues Netz erzeugt werden oder aber ein bereits bestehendes Netz geladen und angepasst werden. Die Netze werden in Form von Textdateien gespeichert, die alle Informationen über Netztyp, Regelbasis und Fuzzy-Mengen enthalten. Des Weiteren kann ein bestehendes Netz getestet werden und die Ergebnisse in einer Textdatei abgespeichert werden. Letztendlich kann ein bestehendes Netz in Form von GNU-PLOT-Graphiken visualisiert werden. Dabei werden die Fuzzy-Mengen der Ein- und Ausgangsvariablen graphisch dargestellt.

Kapitel 4

Modifikation und Test der Software NFident

Um die vorhandene NFident-Software für die Robotersteuerung im Rahmen der bestehenden Projekte des Instituts für Verteilte und Parallele System an der Universität Stuttgart nutzen zu können, mussten einige Änderungen vorgenommen werden. Für die Software wurde ein C++-Interface implementiert, über das es möglich ist, sie in objektorientierte Software-Projekte einzubinden. Außerdem sollten die Vorgaben bei der Erstellung von Netzen und beim Training, die ursprünglich über Testdateien gemacht werden, ebenso softwaremäßig gemacht werden können. Die Funktionalität des Programms wurde außerdem dadurch erhöht, dass weitere Funktionen, die ursprünglich nur intern für das Training verwendet wurden, für den Anwender nutzbar gemacht wurden.

Als Basisfunktionen ist es zunächst möglich, ein bestehendes Netz zu laden, zu speichern oder als GNU-PLOT-Graphiken auszugeben. Das Netz muss dazu in Form einer sogenannten Netz-Datei gegeben sein und wird auch als solche abgespeichert. Eine solche Datei wird beispielhaft im Anhang A.2 und dargestellt. Sie enthält unter anderem die in Tabelle 4.1 aufgelisteten Parameter.

Des Weiteren können mit der modifizierten Software neue Neuro-Fuzzy-Systeme erstellt werden. Dazu können zum einen, so wie in der ursprünglichen Software, die Netzeigenschaften aus einer Parameterdatei geladen werden (Beispiel im Anhang A.1). Die Netzeigenschaften können jedoch auch über entsprechende Funktionen vorgegeben werden und anschließend kann daraus ein Netz erzeugt werden. Tabelle 4.2 listet die Parameter auf, welche vom Benutzer ohne die Verwendung einer Parameterdatei vorgegeben werden können.

Das Netz kann wahlweise über die Funktion `newNet()` sofort mit einer in einer Datei abgelegten Menge von Trainingsdaten trainiert werden, oder aber mit der Funktion `createFuzzySystem()` als "leeres" Netz erzeugt werden. Dabei ist jedoch zu beachten, dass es nicht möglich ist, ein Netz ohne Fuzzy-Mengen zu erzeugen. Um jedoch die Fuzzy-Mengen zu erzeugen, benötigt die Software Trainingsdaten. Deswegen muss auch für die Funktion `createFuzzySystem()` mindestens ein Datensatz vorgegeben werden.

Parameter	Schlüsselwort	mögliche Werte
Netztyp	TYPE	0 $\hat{=}$ Netz vom Typ NEFPROX/Mamdani 1 $\hat{=}$ Netz vom Typ ANFIS/Sugeno
Anzahl der Eingangsgrößen Anzahl Regeln Anzahl der Ausgangsgrößen	DIMENSIONS	Anzahl Eingangsvariablen Anzahl Regeln Anzahl Ausgangsvariablen
Operator zur Verknüpfung der Regelprämissen	T-NORM	0 $\hat{=}$ MIN 1 $\hat{=}$ PROD
Defuzzifizierungsmethode	DEFUZZIFICATION	0 $\hat{=}$ Center of Gravity 1 $\hat{=}$ Mean of Maximum
Definition der Fuzzy-Mengen Form der Fuzzy-Mengen: Parameter a, b, c, d:	VARIABLES	0 $\hat{=}$ Dreieck 1 $\hat{=}$ Trapez 2 $\hat{=}$ Gaußsche Glockenkurve entsprechend der Form der Fuzzy-Mengen (Gleichung (3.2) bis (3.4))
Regelbasis	RULES	Regeln entsprechend der Definition der Fuzzy-Mengen

Tabelle 4.1: Parameter einer Netzdatei

Eigenschaft	Funktion	Bemerkung
Netztyp	<code>setNetType()</code>	0 $\hat{=}$ Netz vom Typ NEFPROX/Mamdani 1 $\hat{=}$ Netz vom Typ ANFIS/Sugeno
Defuzzifizierungsmethode	<code>setDefuzzification()</code>	0 $\hat{=}$ Center of Gravity 1 $\hat{=}$ Mean of Maximum
Anzahl der Variablen	<code>setDimensions</code>	
Anzahl und Form der Fuzzy-Mengen der Eingangsvariablen	<code>setInputFuzzySets()</code>	
Anzahl und Form der Fuzzy-Mengen der Ausgangsvariablen	<code>setOutputFuzzySets()</code>	Form: 0 $\hat{=}$ Dreieck 1 $\hat{=}$ Trapez 2 $\hat{=}$ Gaußsche Glockenkurve
Operator zur Verknüpfung der Regelprämissen	<code>setTNorm()</code>	0 $\hat{=}$ MIN 1 $\hat{=}$ PROD
Lernrate σ	<code>setLearningRate()</code>	
Epochendefinition	<code>setEpochDef()</code>	maxepoch: max. Epochenanzahl minepoch: min. Epochenanzahl chgepoch: Abbruchkriterium
Lernart	<code>setLearningRestrictions()</code>	batch/online Lernen assymetrische Fuzzy-Mengen erlauben
maximale Regelanzahl	<code>setMaxRules()</code>	

Tabelle 4.2: Netzeigenschaften die vom Benutzer über Funktionen vorgegeben werden können

Im günstigsten Fall sollte dieser (willkürliche) Werte im oberen Bereich des Definitionsbereichs der Variablen enthalten.

Die angepasste Software ermöglicht es einzelne oder mehrere Datensätze in Form eines arrays einzulesen. Hiermit kann ein beliebiges Netz, egal ob "leer" oder bereits vor-trainiert, trainiert werden.

Dasselbe gilt für das Einlesen von Generalisierungsdaten, die bei Bedarf für das Trainieren verwendet werden können.

Zwei wichtige neue Funktionen sind diejenigen, welche das propagieren und backpropagieren einzelner Datensätze erlauben. Die Funktion `propagatePattern()` ermöglicht es, die Netzausgabe für eine bestimmte Kombination von Eingangswerten abzufragen. Mithilfe der Funktion `backpropagatePattern()` können einzelne Datensätze bestehend aus Eingangswerten und zugehörigen Ausgangswerten rückwärts in das Netz gegeben werden, woraufhin ein Lernvorgang stattfindet. Im Unterschied zum Trainieren des Netzes wird hierbei nur ein Datensatz verwendet. Die Anzahl der Epochen während der maximal trainiert werden soll, kann angegeben werden.

4.1 Anwendungsmöglichkeiten

Mithilfe der modifizierten Software bietet sich nun ein breites Anwendungsspektrum. Für eine beliebige Lernaufgabe kann ein Netz erstellt und trainiert werden. Die Anzahl der Ein- und Ausgangsvariablen kann dabei frei gewählt werden. Die Netz- und Lernparameter können für die jeweilige Aufgabe optimiert werden. Des Weiteren kann das NFident-System in andere objektorientierte Softwaresysteme eingebunden werden.

Im Rahmen dieser Arbeit wurden im wesentlich drei Anwendungsfälle näher betrachtet: zunächst wurde die Software anhand einiger einfacher Probleme der Funktionsapproximation getestet. Im nächsten Schritt wurde mithilfe der Software ein Netz erstellt, das direkt zu Steuerung eines Roboters verwendet wurden. Schließlich wurde versucht, das Netz in ein bestehendes System einzubinden, welches dazu verwendet wird, mithilfe von Reinforcement Learning die Steuerung des Roboters zu erlernen.

Kapitel 5 befasst sich ausführlicher mit der Anwendung der Software für die Robotersteuerung. Die als Voruntersuchung dienende Funktionsapproximation soll im folgenden Abschnitt 4.2 vorgestellt werden.

4.2 Funktionsapproximation

Um die Funktionalität der NFident-Software zu überprüfen wurden vor der eigentlichen Anwendung für die Robotersteuerung einige Funktionsapproximationen durchgeführt. Sie sollen zeigen, wie das Lernverfahren funktioniert und wie bereits bestehende Netze auf das Nachtrainieren mit weiteren Datensätzen reagieren. Dabei geht man davon aus, dass die Aufgabe der Robotersteuerung in etwa der Approximation einer Funktion entspricht. Das Neuro-Fuzzy-Netz erhält als Vorgabe einige Zustandswerte

wie z.B. Position und Ausrichtung des Roboters (x, y) und muss aus diesen Daten einen Ausgabewert, z.B. eine Aktion, $z = f(x, y)$ bestimmen.

Da das mathematische Modell der Robotersteuerung nicht bekannt ist, ist auch die Topologie der zu approximierenden Funktion im diesem Fall nicht bekannt. Um jedoch zunächst die Funktionalität der Software für ein solches Szenario zu überprüfen, wird im folgenden Abschnitt versucht, eine (bekannte) Funktion mithilfe unvollständiger Datensätze zu approximieren und es wird das Verhalten der Neuro-Fuzzy-Netze dabei untersucht. Im ersten Teil wird eine nichtlineare zweidimensionale Funktion $f(x)$ untersucht, im Anschluss wird eine dreidimensionale Funktion $f(x, y)$ betrachtet. Die folgenden Abschnitte dokumentieren die Ergebnisse dieser Untersuchung.

4.2.1 Funktionsapproximation - zweidimensional

Betrachtet wird ein Neuro-Fuzzy-Netzwerk mit einer Eingangsvariable x und einer Ausgangsvariable y . Es soll ein Neuro-Fuzzy-Netz erstellt und trainiert werden, welches eine Funktion $f(x, y)$ möglichst genau approximieren kann.

Betrachtet wird die nichtlineare Funktion

$$y = f_1(x) = \sin(2x) + \cos(x^2) \quad \text{mit } x \in [0, 5] \quad (4.1)$$

Besonders interessant beim Trainieren von Neuronalen Netzen ist es, zu beobachten, wie sich ein bereits bestehendes Netz durch das Nachtrainieren mit Datensätzen, die nur einen Teilbereich des Netzes betreffen, verändert. Es soll daher betrachtet werden, in wie weit und wie gut sich ein bereits grob vortrainiertes Neuro-Fuzzy-Netzwerk durch ein Nachtrainieren mit Teil-Datensätzen verbessern lässt.

Um ein vortrainiertes Netz zu simulieren, welches die Zielfunktion noch nicht vollständig korrekt approximieren kann, werden bei der Erstellung des Netzes Trainingsdatensätze entsprechend einer der Zielfunktion ähnlichen, jedoch nicht identischen Funktion eintrainiert. So erhält man ein Netz, welches einen ähnlichen Funktionsverlauf wie den der Zielfunktion aus Gleichung (4.1) approximiert, an einigen Stellen jedoch erheblich vom gewünschten Funktionsverlauf abweicht.

Zum Vortrainieren wurde die Funktion

$$y = f_2(x) = \sin(2x) - \sin(2x)^2 + \cos(0.5x^2) \quad \text{mit } x \in [0, 5] \quad (4.2)$$

gewählt und es wurden 63 Wertepaare vorgegeben mit denen das Netz initialisiert wurde.

Abbildung 4.1 zeigt die zu approximierende Funktion $f_1(x)$ aus Gleichung (4.1) und die zum Vortrainieren verwendete Funktion $f_2(x)$ aus Gleichung (4.2).

Wird ein Neuro-Fuzzy-Netz N_1 vom Typ Mamdani mit den in Tabelle 4.3 zusammengefassten Eigenschaften erstellt und wie beschrieben mit 63 Werten der Funktion $f_2(x)$ trainiert, so approximiert das Netz N_1 anschließend den in Abbildung 4.2 dargestellten Funktionsverlauf. Es ist deutlich erkennbar, dass die Funktion $f_2(x)$ relativ gut

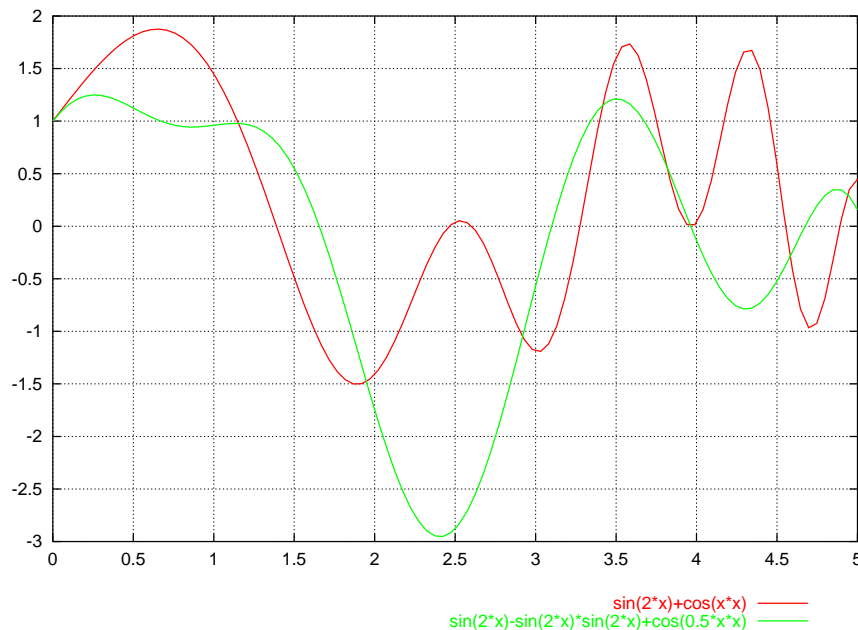


Abbildung 4.1: Die zu approximierende Funktion $f_1(x) = \sin(2x) + \cos(x^2)$ (rot) und die zum Vortrainieren verwendete Funktion $f_2 = \sin(2x) - \sin(2x)^2 + \cos(0.5x^2)$ (grün)

approximiert wird. Wenn überhaupt, weicht der Ausgabewert des Netzes N_1 nur sehr geringfügig vom Funktionswert ab.

Nimmt man jedoch an, dass das Netz N_1 letztendlich die Funktion $f_1(x)$ approximieren soll (wie es ja hier der Fall ist), so ist die Approximation zunächst noch relativ schlecht.

Nun werden nach und nach Abschnitte der Funktion $f_1(x)$ in das vortrainierte Netz N_1 eingegeben und es wird beobachtet, wie gut sich die Approximation dem gewünschten Funktionsverlauf $f_1(x)$ annähert.

Zunächst wird das vortrainierte Netz N_1 mit Einem Satz W_1 von 20 Wertepaaren (x, y) entsprechend der Funktion $f_1(x)$ mit $x \in [2, 2.5]$ trainiert. Abbildung 4.3 zeigt, wie sich die Approximation im entsprechenden Bereich anpasst, ansonsten aber fast unverändert bleibt.

Die Abbildungen 4.4 und 4.5 zeigen, wie sich die Approximation ändert, wenn man das Netz N_1 mit anderen Sätzen von Wertepaaren W_2 und W_3 trainiert. Dabei enthält W_2 40 Wertepaare (x, y) für die gilt: $y = f_1(x)$ mit $x \in [3.5, 4.5]$

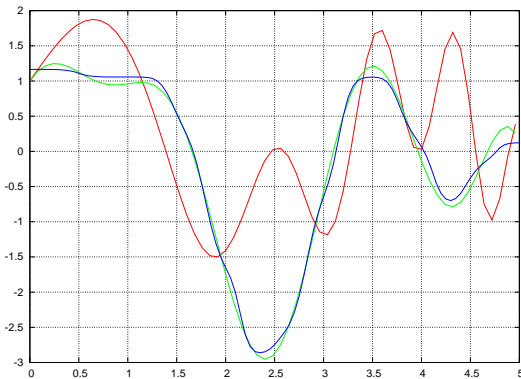
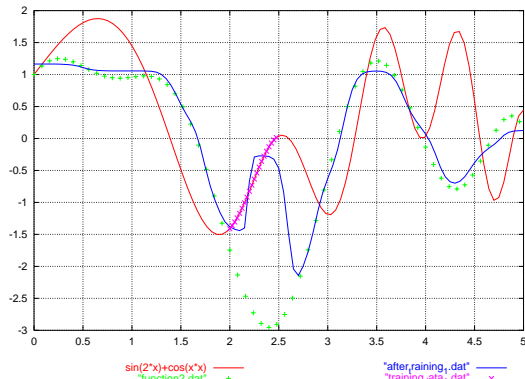
W_3 besteht aus 20 Wertepaaren mit $y = f_1(x)$ mit $x \in [3, 3.5]$

Im nächsten Schritt das Netz N_1 nun nacheinander die Datensätze W_1 , W_2 und W_3 trainiert. Das heißt, das durch das Trainieren mit W_1 entstandene Netz N_{W_1} wird mit W_2 trainiert. Es entsteht das Netz $N_{W_1,2}$. Dieses wiederum wird mit W_3 trainiert. Das resultierende Netz heißt $N_{W_1,2,3}$. Abbildung 4.6 zeigt die Funktionsapproximation durch das Netz $N_{W_1,2}$. In Abbildung 4.7 ist die Approximation durch $N_{W_1,2,3}$ dargestellt.

Zum Vergleich wird das Netz N_{W_1} außerdem in einem Schritt mit den Datensätzen W_2

Eigenschaftsparameter	Wert	Bedeutung
Netztyp	0	Netz vom Typ NEFPROX/Mamdani
Anzahl der Eingangsgrößen	1	
Anzahl der Ausgangsgrößen	1	
Operator zur Verknüpfung der Regelprämissen	0	MIN
Defuzzifizierungsmethode	0	Center of Gravity
Anzahl von Fuzzy-Mengen pro Variable	16	
Form der Fuzzy-Mengen	2	Gaußsche Glockenkurve
Lernrate σ	0.01	
maximale Regelanzahl	-1	beliebig
minimaler Fehler (Abbruchkrit.)	0.0	Lernen beenden wenn Fehler = 0.0
Abbruchkrit. $chepoch$	100	Lernen beenden wenn Fehler 100 Epochen lang nicht kleiner wird
min. Anzahl Epochen	0	
max. Anzahl Epochen	1000	
Lernrhythmus	1	batch-learning
Eigensch. der Fuzzy-Mengen: $nopass$	1	Fuzzy-Mengen dürfen während des Trainings nicht aneinander vorbeilaufen
$assym$	1	Fuzzy-Mengen dürfen während des Trainings assymetrisch werden

Tabelle 4.3: Eigenschaften des für die 2D-Funktionsapproximation verwendeten Netzes

Abbildung 4.2: Funktionsapproximation durch das Netz N_1 (blau) nach dem Training mit 63 Wertepaaren (x, y) der Funktion $f_2(x)$ mit $x \in [0, 5]$ (grün)Abbildung 4.3: Funktionsapproximation durch das Netz N_{W_1} (blau) nach dem Training mit 20 Wertepaaren (x, y) der Funktion $f_1(x)$ mit $x \in [2, 2.5]$ (rosa)

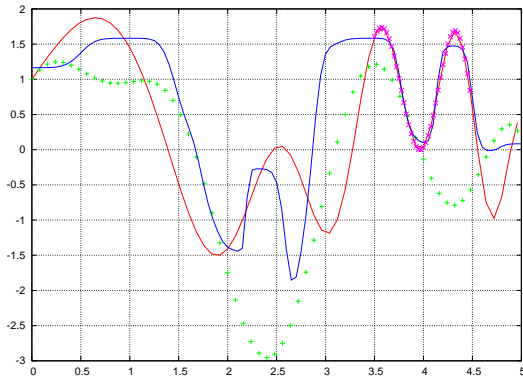


Abbildung 4.4: Funktionsapproximation durch das Netz N_{W_2} (blau) nach dem Training mit 40 Wertepaaren (x, y) der Funktion $f_1(x)$ mit $x \in [3.5, 4.5]$ (rosa)

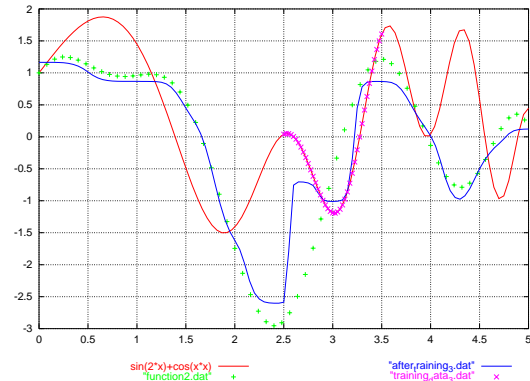


Abbildung 4.5: Funktionsapproximation durch das Netz N_{W_3} (blau) nach dem Training mit 20 Wertepaaren (x, y) der Funktion $f_1(x)$ mit $x \in [3, 3.5]$ (rosa)

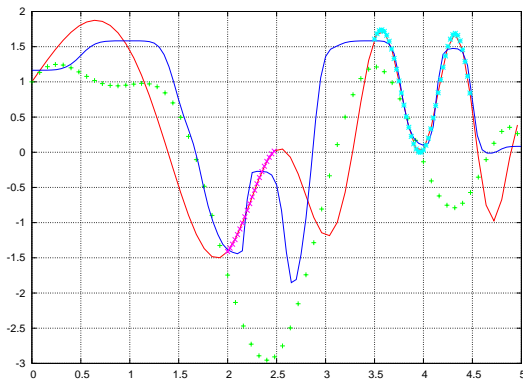


Abbildung 4.6: Funktionsapproximation durch das Netz $N_{W_{1,2}}$ (blau) nach dem Training mit W_1 (rosa) und W_2 (türkis)

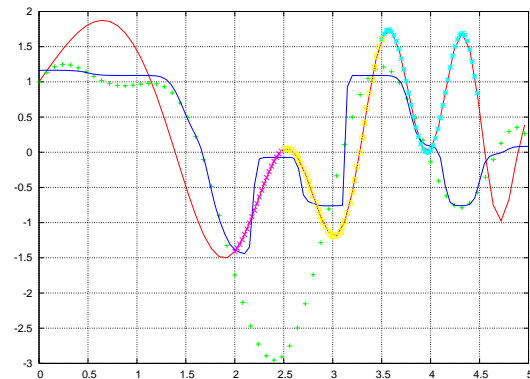


Abbildung 4.7: Funktionsapproximation durch das Netz $N_{W_{1,2,3}}$ (blau) nach dem Training mit W_1 (rosa), W_2 (türkis) und W_3 (gelb)

und W_3 trainiert. Wie in Abbildung 4.8 erkennbar ist, ist die Approximation durch das resultierende Netz $N_{W'_{1,2,3}}$ im Bereich $x \in [4, 4.5]$ besser als $N_{W_{1,2,3}}$ (Abbildung 4.7). Um $x = 3$ zeigt sich hingegen eine leichte Verschlechterung der approximierten Werte. Insgesamt wird jedoch der Funktionsverlauf von $N_{W'_{1,2,3}}$ etwas besser approximiert als von $N_{W_{1,2,3}}$.

Bei einer genaueren Betrachtung der Bilder erkennt man, dass das Nachtrainieren eines bestehenden Netzes im Allgemeinen eine positive Auswirkung auf die Funktionsapproximation hat. Der Bereich dem die Trainingsdatensätze entnommen wurden wird vom modifizierten Netz deutlich besser approximiert. Vergleicht man jedoch die Abbildungen 4.6 und 4.7, so erkennt man, dass das Trainieren mit einem weiteren Datensatz die Verbesserung (hier im Bereich $x \in [3.5, 4.5]$) teilweise rückgängig macht. Dieser Effekt kommt dadurch zustande, dass das Netz beim Nachtrainieren dadurch angepasst wird, dass die Fuzzy-Mengen der Ein- und Ausgangsvariablen verschoben werden. So werden sie im ersten Trainingsschritt in den Bereich $x \in [3.5, 4.5]$ geschoben, so dass dieser optimal approximiert wird (Abbildung 4.6). Im zweiten Trainingsschritt werden die Fuzzy-Mengen jedoch im Bereich $x \in [3.0, 3.5]$ "benötigt" um dort die Approximation zu verbessern. Die Fuzzy-Mengen werden also vom Bereich des ersten Trainings wieder abgezogen (Abbildung 4.7). Trainiert man das Netz N_{W_1} jedoch sofort mit beiden Trainingsmengen W_2 und W_3 ergibt sich im Mittel eine bessere Approximation (Abbildung 4.8).

Schließlich wird noch das Netz $N_{W_{1,2,3}}$ mit 200 Datensätzen (x, y) trainiert, die den gesamten betrachteten Bereich abdecken und für die gilt $y = f(x)$ mit $x \in [0, 5]$. Abbildung 4.9 zeigt, dass die Approximation sich nicht mehr wesentlich verändert.

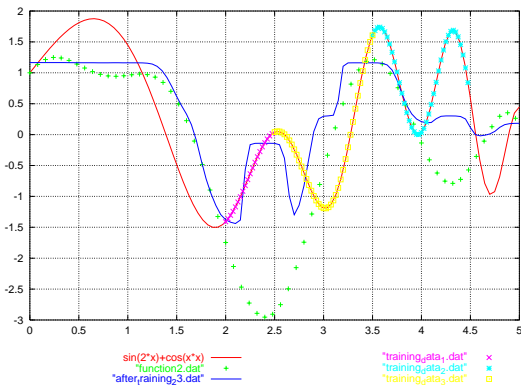


Abbildung 4.8: Funktionsapproximation durch das Netz $N_{W'_{1,2,3}}$ (blau) nach dem Training mit W_1 (rosa) und $W_2 \cup W_3$ (türkis und gelb)

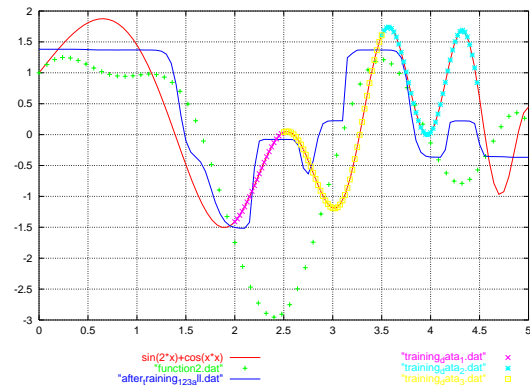


Abbildung 4.9: Funktionsapproximation nach dem Training mit W_1 (rosa) und W_2 (türkis) und W_3 (gelb) sowie der gesamten Funktion $f_1(x)$ (rot)

4.2.2 Funktionsapproximation - dreidimensional

Ähnlich dem Vorgehen in Abschnitt 4.2.1 soll nun ein Neuro-Fuzzy-Netzwerk mit zwei Eingangsvariablen x und y und einer Ausgangsvariable z betrachtet werden. Die zu

approximierende Funktion $f_1(x, y)$ ist also eine dreidimensionale Funktion.

Gewählt wurde die Funktion

$$z = f(x, y) = \sin(x) - \cos(y) \quad x \in [0, 5], y \in [0, 5] \quad (4.3)$$

Der Funktionsverlauf ist in Abbildung 4.10 dargestellt. Tabelle 4.4 fasst die Eigenschaften des verwendeten Netzes zusammen.

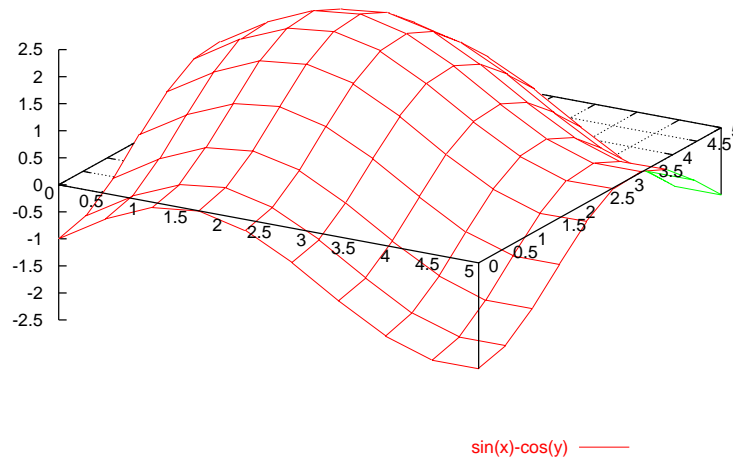


Abbildung 4.10: Die zu approximierende Funktion $z = f(x, y) = \sin(x) - \cos(y)$

Zum Trainieren des Fuzzy-Netzes wurden verschiedene Pfade durch die Fläche gewählt und als Wertemengen mit Tupeln (x, y, z) gewählt. Die vier Trainingsmengen W_1 , W_2 , W_3 und W_4 sind in Abbildung 4.11 dargestellt. Sie lassen sich wie folgt berechnen:

$$W_1 : f(x, y) = \sin(x) - \cos(y) \quad \text{mit } x \in [0, 5] \text{ und } y = 5 - x^2 \quad (4.4)$$

$$W_2 : f(x, y) = \sin(x) - \cos(y) \quad \text{mit } y \in [0, 5] \text{ und } x = (y + 1) * y/4 \quad (4.5)$$

$$W_3 : f(x, y) = \sin(x) - \cos(y) \quad \text{mit } x \in [0, 5] \text{ und } y = 5 - x \quad (4.6)$$

$$W_4 : f(x, y) = \sin(x) - \cos(y) \quad \text{mit } x \in [3.5, 5] \text{ und } y \in [0, 1.5] \quad (4.7)$$

Im Gegensatz zum vorherigen Abschnitt wurde das Netz nicht mit einer anderen Funktion grob vortrainiert, sondern bei der Erstellung des Netzes $N_2 = N_{W_1}$ wurde die Trainingsmenge W_1 verwendet. Da diese Wertemenge nur einen sehr kleinen Bereich der Funktion abdeckt ist die Approximation, wie in Abbildung 4.12 zu erkennen ist, zunächst noch relativ schlecht.

Trainiert man nun das Netz N_{W_1} zusätzlich mit W_2 , verbessert sich die Approximation, vor allem im Bereich des Pfades W_2 . Die Funktionsapproximation durch das resultierende Netz $N_{W_1,2}$ ist in Abbildung 4.13 dargestellt.

Eigenschaftsparameter	Wert	Bedeutung
Netztyp	0	Netz vom Typ NEFPROX/Mamdani
Anzahl der Eingangsgrößen	2	
Anzahl der Ausgangsgrößen	1	
Operator zur Verknüpfung der Regelprämissen	0	MIN
Defuzzifizierungsmethode	0	Center of Gravity
Anzahl von Fuzzy-Mengen pro Variable	16	
Form der Fuzzy-Mengen	2	Gaußsche Glockenkurve
Lernrate σ	0.01	
maximale Regelanzahl	-1	beliebig
minimaler Fehler (Abbruchkrit.)	0.0	Lernen beenden wenn Fehler = 0.0
Abbruchkrit. $chepoch$	100	Lernen beenden wenn Fehler 100 Epochen lang nicht kleiner wird
min. Anzahl Epochen	0	
max. Anzahl Epochen	1000	
Lernrhythmus	1	batch-learning
Eigensch. der Fuzzy-Mengen: $nopass$	1	Fuzzy-Mengen dürfen während des Trainings nicht aneinander vorbeilaufen
$assym$	1	Fuzzy-Mengen dürfen während des Trainings assymetrisch werden

Tabelle 4.4: Eigenschaften des für die 3D-Funktionsapproximation verwendeten Netzes

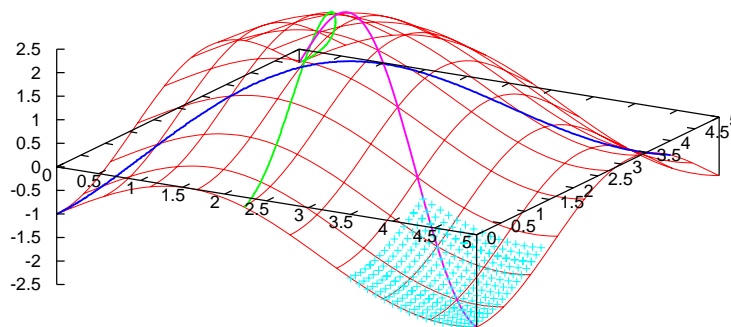


Abbildung 4.11: Die zu approximierende Funktion $f(x,y) = \sin(x) - \cos(y)$ (rot) und die daraus gewonnenen Trainingsmengen W_1 (grün), W_2 (blau), W_3 (rosa) und W_4 (türkis)

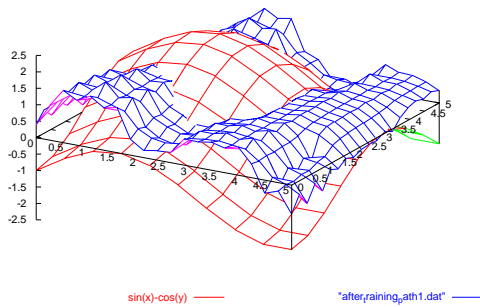


Abbildung 4.12: Die zu approximierende Funktion $f(x, y) = \sin(x) - \cos(y)$ (rot) und die Approximation durch das Neuro-Fuzzy-Netz N_{W_1} (blau) nach dem Training mit Trainingsmenge W_1

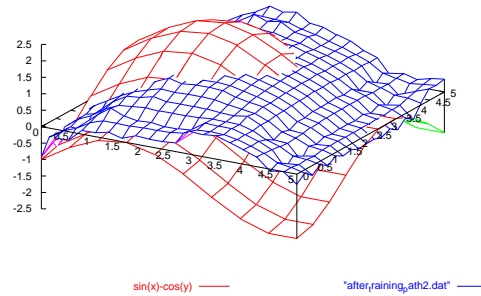


Abbildung 4.13: Die zu approximierende Funktion $f(x, y) = \sin(x) - \cos(y)$ (rot) und die Approximation durch das Neuro-Fuzzy-Netz $N_{W_{1,2}}$ (blau) nach dem Training mit den Trainingsmengen W_1 und W_2

Nach einem weiteren Training mit Trainingsmenge W_3 approximiert das Netz $N_{W_{1,2,3}}$ die Funktion wie in Abbildung 4.14 dargestellt.

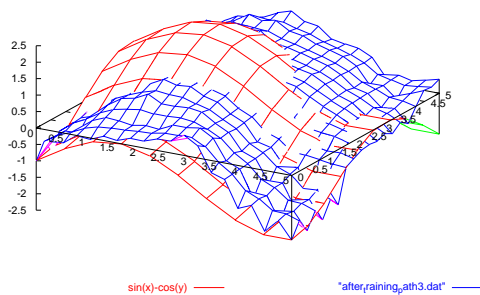


Abbildung 4.14: Die zu approximierende Funktion $f(x, y) = \sin(x) - \cos(y)$ (rot) und die Approximation durch das Neuro-Fuzzy-Netz $N_{W_{1,2,3}}$ (blau) nach dem Training mit den Trainingsmengen W_1 , W_2 und W_3

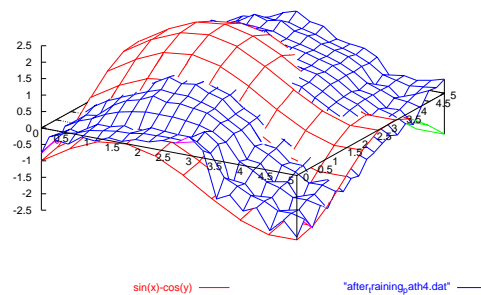


Abbildung 4.15: Die zu approximierende Funktion $f(x, y) = \sin(x) - \cos(y)$ (rot) und die Approximation durch das Neuro-Fuzzy-Netz $N_{W_{1,2,3,4}}$ (blau) nach dem Training mit den Trainingsmengen W_1 , W_2 , W_3 und W_4

Das Netz $N_{W_{1,2,3}}$ approximiert damit die gewünschte Funktion nun schon recht gut. Vor allem die Funktionswerte, die in der Umgebung der bereits eintrainierten Trainingsmengen liegen werden nahezu richtig approximiert.

In Abbildung 4.16 ist zur Verdeutlichung nochmals die Differenz zwischen Sollwert ($z = f(x, y)$) und Istwert (Ausgabe des Netzes $N_{W_{1,2,3}}$) aufgetragen.

Wird das Netz $N_{W_{1,2,3}}$ nun noch mit Trainingsmenge W_4 trainiert, verbessert sich die Approximation nochmals, wie in Abbildung 4.15 sichtbar ist.

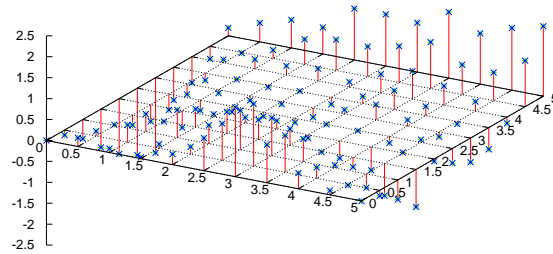


Abbildung 4.16: *Differenz zwischen Sollwert $z = f(x, y)$ und Approximation durch das Netz $N_{W_{1,2,3}}$ nach dem Training mit den Trainingsmengen W_1 , W_2 und W_3*

Um die Auswirkungen des beschriebenen Trainings zu verdeutlichen und sichtbar zu machen, auf welche Bereiche des Netzes sich das Training mit einer Teilmenge der Zielfunktion auswirkt, zeigt Abbildung 4.17 die Differenz zwischen den approximierten Werten nach dem zweiten und nach dem dritten Training ($N_{W_{1,2}}$ und $N_{W_{1,2,3}}$). Es ist deutlich erkennbar, dass die größten Änderungen in den Bereichen vorkommen, die nahe der Trainingsgeraden liegen und in denen der approximierte Wert noch stark vom Sollwert abwich (hier vor allem im Bereich $x > 3.5$ und $y < 1.5$)

4.2.2.1 Online Lernen

Bei den bisherigen Tests wurde jeweils ein ganzer Datensatz auf einmal in das Netz eintrainiert. Soll das Netz online, also während der Verwendung, lernen, so wird jeweils nach einem einzelnen Datensatz ein Trainingsschritt durchgeführt.

Die Lernergebnisse sind hierbei geringfügig schlechter. Grund hierfür ist die Tatsache, dass vor jedem Training die Regelbasis überprüft und eventuell erweitert wird. Werden alle Trainingsdatensätze auf einmal eintrainiert, so wird die Regelbasis dementsprechend erweitert und für alle Trainingsdatensätze verwendet. Werden die Datensätze einzeln eintrainiert, so wird die Regelbasis erst nach und nach erweitert, so dass für die ersten Datensätze eine kleinere Regelbasis verwendet wird als für die später eintrainierten. Die Lernergebnisse beim online Lernen sind in den drei Abbildungen 4.18, 4.19 und 4.20 dargestellt.

4.2.3 Fazit

Die in den Abschnitten 4.2.1 und 4.2.2 beschriebenen Test der NFident-Software zeigen, dass die Software sehr gut zur Funktionsapproximation geeignet ist.

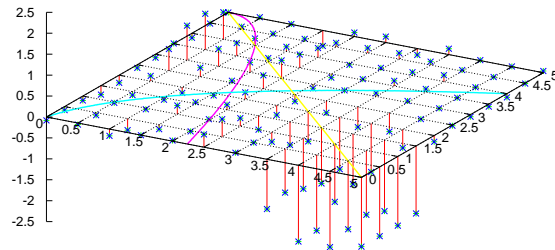


Abbildung 4.17: Differenz der approximierten Werte nach dem zweiten Training (Netz $N_{W_{1,2}}$) und dritten Training (Netz $N_{W_{1,2,3}}$), gelb dargestellt die neu eintrainierte Trainingsgerade sowie die bereits vorher eintrainierten Trainingsmengen 1 und 2 in blau und rosa

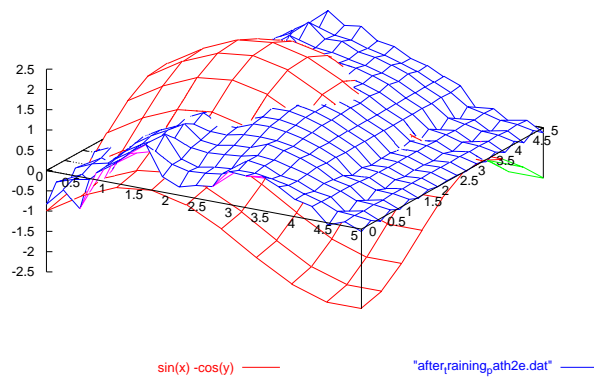


Abbildung 4.18: Die zu approximierende Funktion $f(x, y) = \sin(x) - \cos(y)$ (rot) und die Approximation durch das Neuro-Fuzzy-Netz (blau) nach dem Training mit den einzelnen Trainingswerten aus Menge W_1 und W_2

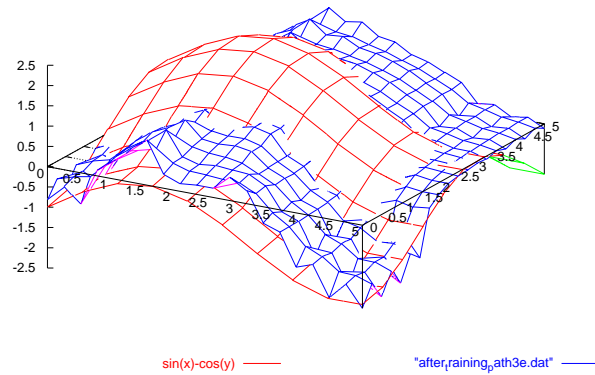


Abbildung 4.19: Die zu approximierende Funktion $f(x, y) = \sin(x) - \cos(y)$ (rot) und die Approximation durch das Neuro-Fuzzy-Netz (blau) nach dem Training mit den einzelnen Trainingswerten aus Menge W_1, W_2 und W_3

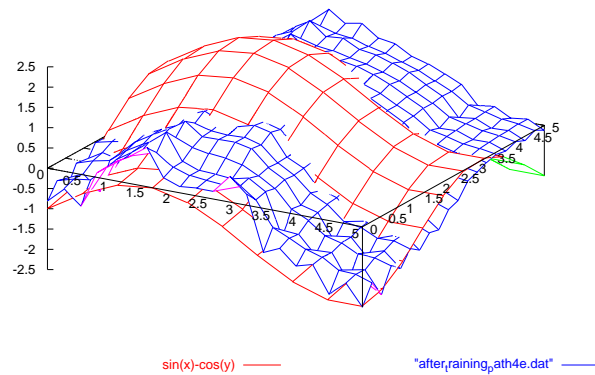


Abbildung 4.20: Die zu approximierende Funktion $f(x, y) = \sin(x) - \cos(y)$ (rot) und die Approximation durch das Neuro-Fuzzy-Netz (blau) nach dem Training mit den einzelnen Trainingswerten aus Menge W_1, W_2, W_3 und W_4

Betrachtet wurde vor allem das stückweise Trainieren eines Neuro-Fuzzy-Netzes, welches durch die Modifikation der Software ermöglicht wurde und welches später für die Roboter-Steuerung verwendet wird. Auch dieses Training erzielt verhältnismäßig gute Approximationsergebnisse. Es hat sich jedoch gezeigt, dass das Ergebnis beim stückweisen Training etwas schlechter ist und sich bereits gut eintrainierte Bereiche wieder verschlechtern, wenn vermehrt neue Bereiche trainiert werden. Dieser Effekt sollte bei der Anwendung beachtet werden. Wenn möglich sollten große Sätze von Trainingsdaten am Stück eintrainiert werden, so dass sie das Netz gleichmäßig anpasst.

Des Weiteren wurde gezeigt, dass ein vergleichbarer Lerneffekt erzielt wird, wenn nicht ein ganzer Datensatz am Stück eintrainiert wird (sog. Batch Lernen), sondern jeder Datensatz einzeln rückpropagiert wird (Online Lernen). Auch hier ist die Approximation nach dem Training etwas schlechter als wenn alle Datensätze am Stück eintrainiert werden. In der Anwendung sollte deswegen das Batch Lernen dem Online Lernen vorgezogen werden.

Kapitel 5

Robotersteuerung mit Neuro-Fuzzy-Systemen

Die modifizierte NFident-Software soll dazu verwendet werden einen Roboter zu steuern. Dabei bieten sich verschiedene Möglichkeiten, ein Neuro-Fuzzy-Netz in das Roboter-System zu integrieren. Die verschiedenen Ansätze sollen in diesem Kapitel beschrieben werden.

Es wird vorausgesetzt, dass das Neuro-Fuzzy-Netz ausschließlich die Bewegungen des Roboters steuern soll. Es findet keine Situationserkennung statt und es werden keinerlei Sensorinformationen verarbeitet. Das Neuro-Fuzzy-System erhält von anderen Systemkomponenten die notwendigen Vorgaben, wie z.B. die eigene Position, die eigene Orientierung und die Zielposition. Daraus bestimmt das Neuro-Fuzzy-Netz die optimale Aktion für ein vorgegebenes Verhaltensmuster wie z.B. das Anfahren eines Balls.

5.1 Robotersteuerung durch Vorgabe von Regeln

Die naheliegendste Methode, mithilfe eines Neuro-Fuzzy-Netzes einen Roboter zu steuern, ist es direkt die Regeln für das Fahrverhalten zu formulieren. Da es eine wichtige Eigenschaft von Neuro-Fuzzy-Netzen ist, dass Expertenwissen in Form von linguistischen Regeln integriert werden kann, ist dies relativ einfach zu realisieren. So kann allein mithilfe von externem Wissen ein Netz erzeugt werden, welches den Roboter entsprechend bestimmter Verhaltensmuster steuert.

Es sei das folgende Szenario gegeben:

Bekannt sind die Position des Roboters ($robX$, $robY$) sowie seine Orientierung $robTh$ im Fußballfeld. Außerdem ist die Position des Balls im Fußballfeld gegeben (x, y). Aus den gegebenen Informationen lässt sich außerdem der Abstand des Roboters zum Ball ($Distance$) sowie der Winkel zwischen aktueller Greifer-Ausrichtung und optimaler Greiferausrichtung in Richtung Ball, ($Angle$) berechnen. Der Roboter soll den Ball möglichst direkt anfahren und sich zum Ball hin orientieren, so dass dieser schließlich

im "Greifer" des Roboters liegt.

Als Eingangsgrößen für das Neuro-Fuzzy-Netz sollen nun zunächst die Parameter *Distance* und *Angle* verwendet werden. Das Netz soll daraus die Sollgeschwindigkeiten für Rotation (*rotVel*) und Translation (*transVel*) bestimmen. Der Wertebereich für die Rotation beträgt -280 bis +280 deg/s. Die Translationsgeschwindigkeit kann zwischen 0 und 1600mm/s liegen.

Bevor die linguistischen Regeln für das Verhalten des Roboters formuliert werden können, müssen die Wertebereiche der Ein- und Ausgangsgrößen durch Fuzzy-Mengen partitioniert werden. Für dieses Beispiel wurden die Wertebereiche der Variablen durch jeweils fünf linguistische Terme unterteilt:

- Die Abstand zum Ball (*Distance*) ist *zero, very close, close, far* oder *very far*
- Der Orientierung zum Ball (*deltaTheta*) ist *great negative, negative, zero, positive* oder *great positive*
- Die Translationsgeschwindigkeit (*outTrans*) unterteilt sich in *zero, very slow, slow, fast* und *very fast*
- Die Rotationsgeschwindigkeit (*outRot*) wird gegliedert durch die Fuzzy-Mengen *fast negative, slow negative, zero, slow positive* und *fast positive*.

Nun können mithilfe der definierten linguistischen Terme die Regeln formuliert werden. So gilt zum Beispiel:

Wenn *Distance = very far* **und** *deltaTheta = zero*
dann *outTrans = very fast* **und** *outRot = zero*

Die Regeln werden in NFident in der Form angegeben, dass die linguistischen Terme jeder Variable von 0 bis 4 durchnummeriert werden und die Regeln dann mithilfe der Indizes formuliert werden. So lautet zum Beispiel die oben formulierte Regel in der Netzdatei 4 2 4 2.

Da das Netz in diesem Fall nur zwei Eingangsvariablen mit je fünf Fuzzy-Mengen besitzt, können sämtliche 5^2 Eingangskombinationen durch Regeln abgedeckt werden. Dies ermöglicht eine relativ gute Formulierung des gewünschten Verhaltens. Sobald ein Neuro-Fuzzy-System mehr Eingangsvariablen hat, oder die Eingangsvariablen durch mehr Fuzzy-Mengen partitioniert werden, wird die Regelmenge schnell so groß, dass bei der Formulierung von Regeln eine Auswahl getroffen werden muss und nicht mehr der gesamte Eingaberaum abgedeckt werden kann. Des Weiteren wird eine zu große Regelmenge sehr schnell unübersichtlich, und Fehler im Verhalten des Netzes können

nur schwer zurückverfolgt werden. Folglich ergibt sich bei der Erstellung eines Fuzzy-Netzes das Problem, dass eine höhere Genauigkeit des Netzes, welche nur durch eine größere Anzahl von Fuzzy-Mengen pro Eingangsvariable erreicht werden kann, nur mit großem Aufwand und in der Regel nicht zufriedenstellend realisiert werden kann.

Wichtig ist es in einem solchen Fall, bei der Netz-Modellierung darauf zu achten, dass die vorgegebenen Regeln einen möglichst großen Bereich des Ein- und Ausgaberaumes abdecken müssen. Das Neuro-Fuzzy-Netz interpoliert in den Bereichen, in denen keine Regel greift, einen Ausgabewert. Werden zu wenig Regeln vorgegeben, führt dies dazu, dass das Netz in großen Teilen unbrauchbar ist. Decken die Regeln jedoch weite Teile des Eingaberaumes ab, erzeugt das Netz im allgemeinen sehr gute Ausgabewerte.

Im betrachteten Beispiel ist jedoch die Partitionierung der Eingangsvariablen durch fünf Fuzzy-Mengen ausreichend, so dass es möglich ist, eine vollständige Regelbasis von Hand zu erstellen. Dazu müssen zunächst die Fuzzy-Mengen der Ein- und Ausgänge genau definiert werden. Hier wurden Fuzzy-Mengen gewählt, welche die Form gaußscher Glockenkurven haben. Sie werden, wie in Gleichung (3.4) beschrieben, durch jeweils 2 Parameter definiert: das Zentrum a der Glockenkurve und ihre Breite b . Zunächst liegt es nahe, die Fuzzy-Mengen so zu definieren, dass sie den Eingaberaum einer Variablen gleichmäßig abdecken. Dabei ist jedoch zu beachten, dass häufig vor allem die Randwerte möglichst genau beschrieben werden müssen, hier zum Beispiel die Geschwindigkeit "zero". Deswegen sollten die äußeren Kurven ihr Zentrum am Rand des Wertebereichs haben.

Außerdem sollten spezielle Werte wie der Abstand "0" oder die Geschwindigkeit "0" von einer sehr schmalen Fuzzy-Menge beschrieben werden, damit sie im Verhalten des Netzes deutlich berücksichtigt werden. Ansonsten wird das Netz eine Eingabe "0" in der Regeln nicht als "0" fuzzifizieren, sondern als "nahe 0" und einen Ausgabewert der zur Zugehörigkeitsfunktion "zero" gehört, nicht als exakt "0" defuzzifizieren. Im beschriebenen Szenario soll jedoch der Roboter, wenn er den Ball erreicht hat stehen bleiben (Ausgabe "0") und nicht langsam weiter fahren. Dazu ist es notwendig, dass die Geschwindigkeit 0 ausgegeben wird. Ebenso soll der Roboter wenn er den Ball erreicht hat (Eingabe "0") dies auch eindeutig erkennen und verarbeiten. Eher unwichtige Zwischenbereiche, wie hier zum Beispiel eine mittlere Geschwindigkeit sollten von breiten Fuzzy-Mengen abgedeckt werden.

Sind die Fuzzy-Mengen so definiert, können die Regeln formuliert werden. Dazu gibt man am Besten alle möglichen Eingangskombinationen vor. Hier kommt die aus der Fuzzy-Logik übernommene Eigenschaft der Neuro-Fuzzy-Systeme, dass ihre Funktion in Form von linguistischen Regeln formuliert werden kann, zum Tragen. Sie ermöglicht es dem Nutzer, sein Expertenwissen zu formulieren, ohne dass ein mathematisches Modell notwendig ist. Um zum Beispiel das Geschwindigkeitsverhalten des Roboters zu definieren macht man zum Beispiel die folgenden Aussagen: wenn die Distanz zum Ball klein ist soll die Geschwindigkeit groß sein, je kleiner die Distanz wird, umso kleiner muss die Geschwindigkeit sein. Wenn der Winkel $\Delta\theta$ groß ist und der Abstand schon relativ klein, dann muss die Geschwindigkeit sehr klein sein (bis der Winkel durch Rotation verkleinert wurde). Diese Aussage lässt sich leicht durch die

definierten linguistischen Terme ausdrücken und so als Regelbasis formulieren.

Für den betrachteten Fall des Fußballspiels steht eine Simulation des Szenarios zur Verfügung. Das erstellte Netz kann in die Simulation eingebunden werden und die Funktion des Neuro-Fuzzy-Netzes kann getestet werden. Abbildung 5.1 zeigt das simulierte Fußballszenario.

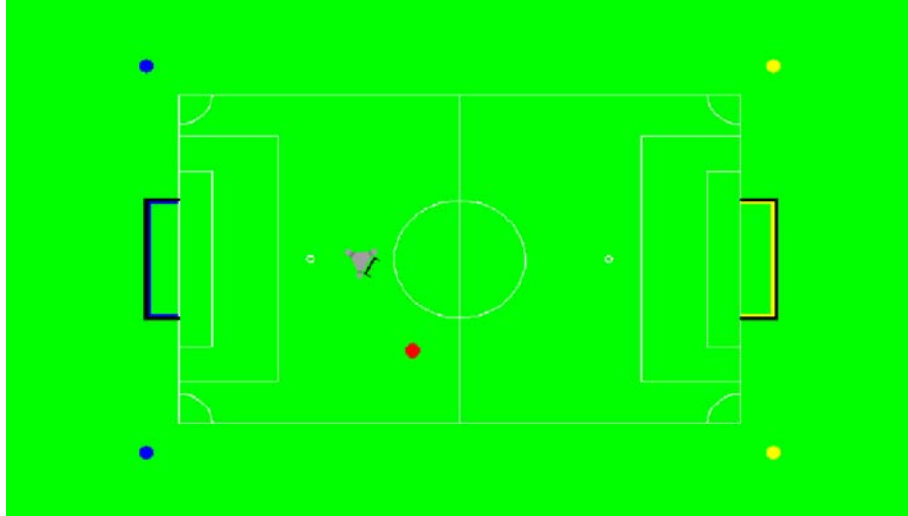


Abbildung 5.1: *Simulation des Fußball-Szenarios*

Auf Grund des beobachteten Verhaltens des Roboters in der Simulation ist es möglich, die Fuzzy-Mengen und die Regeln des Netzes anzupassen, und so zu optimieren, dass das Verhalten des Roboters (subjektiv) besser wird. Im Anhang A.2 ist die Netzdatei angegeben, die für das beschriebene Szenario erstellt wurde. Sie enthält 25 Regeln, mit deren Hilfe sich der Roboter zufriedenstellend steuern lässt. Die Abbildungen 5.2 bis 5.5 zeigen die gewählte Partitionierung der Ein- und Ausgangsvariablen durch jeweils fünf Fuzzy-Mengen.

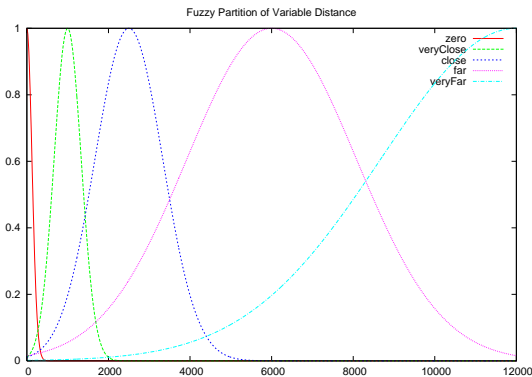


Abbildung 5.2: *Partitionierung der Eingangsvariable Distance durch fünf Fuzzy-Mengen*

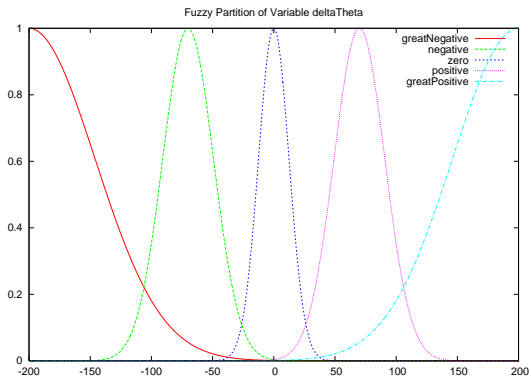


Abbildung 5.3: *Partitionierung der Eingangsvariable deltaTheta durch fünf Fuzzy-Mengen*

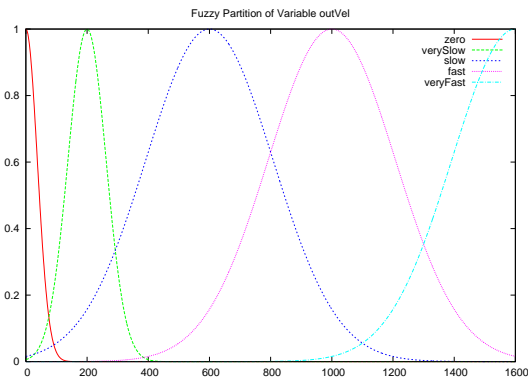


Abbildung 5.4: Partitionierung der Eingangsvariable $outVel$ durch fünf Fuzzy-Mengen

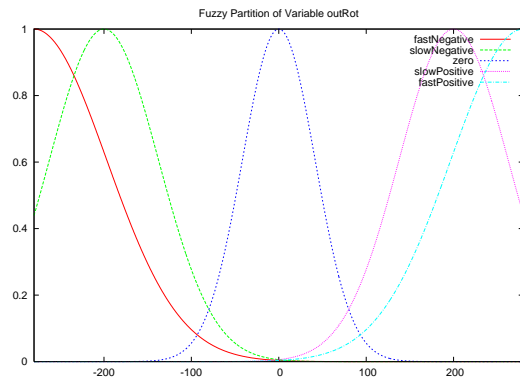


Abbildung 5.5: Partitionierung der Eingangsvariable $outRot$ durch fünf Fuzzy-Mengen

Mithilfe der Simulationsumgebung kann man nun im Nachhinein überprüfen, welche Verläufe sich für die Geschwindigkeit $transVel$ in Abhängigkeit vom Abstand zum Ball oder für die Drehgeschwindigkeit $rotVel$ in Abhängigkeit zu $deltaTheta$ ergeben. Diese Betrachtungen dienen hier nur zur Veranschaulichung anhand eines Beispiels. Für eine genauere Untersuchung müsste beachtet werden, dass im betrachteten Neuro-Fuzzy-Netz die Geschwindigkeit nicht nur vom Abstand, sondern auch von $deltaTheta$ abhängt. Ebenso ist die Drehgeschwindigkeit auch vom Abstand $Distance$ abhängig. Da jedoch die letztendlich vom Neuro-Fuzzy-Netz modellierte Funktion nur bedingt von Interesse ist, soll eine solche Untersuchung hier nicht durchgeführt werden. Abbildung 5.6 zeigt die Geschwindigkeit in Abhängigkeit vom Abstand zum Ball. Da die Strecke bis zum Ball sehr lang ist, und der Roboter damit die Gelegenheit hat, sich in Richtung Ball zu orientieren, kann angenommen werden, dass die Winkeldifferenz $deltaTheta$ keinen Einfluss auf die Geschwindigkeit hat.

Die Unebenheiten in dieser Kurve kommen dadurch zustande, dass versucht wurde, das Heranfahren an den Ball zu beschleunigen. Da bei der Formulierung und Optimierung der Regeln der subjektive Eindruck des Nutzers eine Rolle spielt und nicht eindeutig erkennbar ist, welchen Einfluss das Verändern einer einzelnen Regel auf das Verhalten des Gesamtsystems haben wird, kann es zu solchen Verhaltensweisen des Netzes kommen.

Bild 5.7 zeigt die Rotationsgeschwindigkeit in Abhängigkeit von der Winkeldifferenz. Da die Startposition bei dieser Messung weit vom Ball entfernt lag ist anzunehmen, dass der Abstand zum Ball keinen wesentlichen Einfluss auf die Rotationsgeschwindigkeit hatte. In Bild 5.8 hingegen lag der Ball zu Beginn direkt hinter dem Roboter, d.h. es bestand eine Abhängigkeit zwischen allen Ein- und Ausgangsgrößen. Der Roboter musste zunächst möglichst schnell gedreht werden, bevor er sich weiter dem Ball nähern konnte.

Insgesamt lässt sich sagen, dass durch die manuelle Erstellung eines Neuro-Fuzzy-Netzes für dieses Szenario eine zufriedenstellende Steuerung erreicht werden konnte. Diese Methode eignet sich jedoch nicht für die Anwendung auf komplexere Szenarien

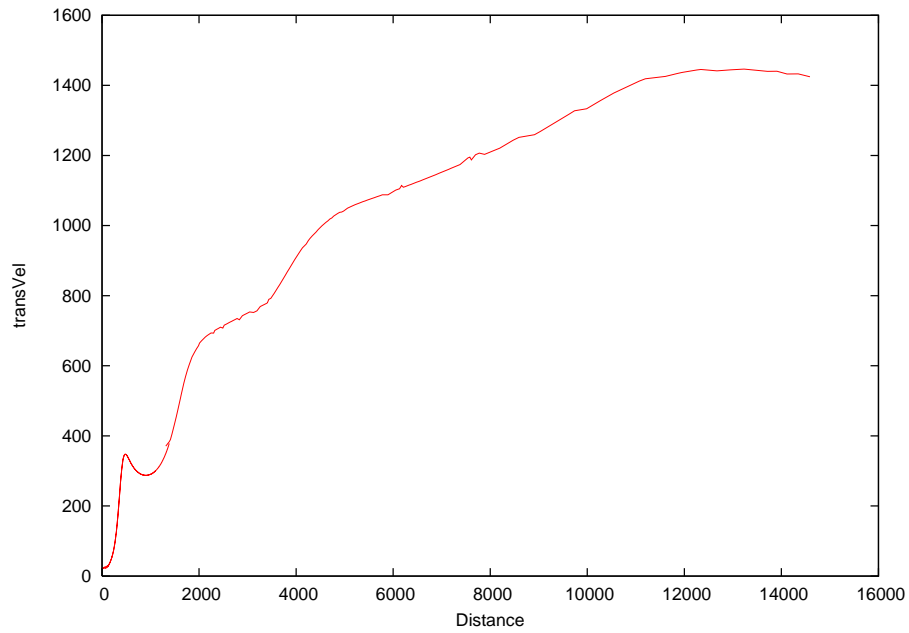


Abbildung 5.6: Vom Neuro-Fuzzy-Netz vorgegebene Sollgeschwindigkeit in Abhängigkeit von der Distanz zum Ball

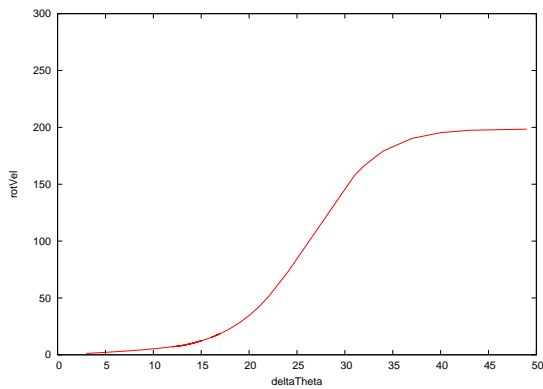


Abbildung 5.7: Vom Neuro-Fuzzy-Netz vorgegebene Sollrotationsgeschwindigkeit in Abhängigkeit von $\delta\theta$ bei großem Anfangsabstand zum Ball

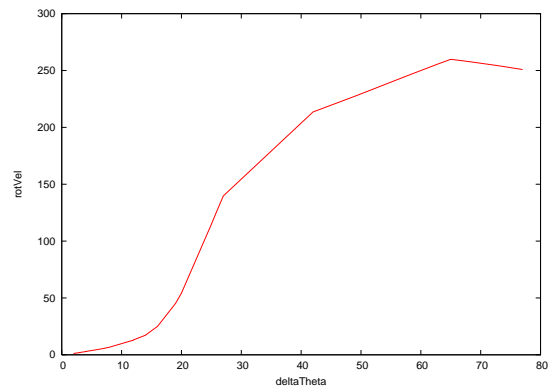


Abbildung 5.8: Vom Neuro-Fuzzy-Netz vorgegebene Sollrotationsgeschwindigkeit in Abhängigkeit von $\delta\theta$ bei kleinem Anfangsabstand zum Ball

mit mehr Eingangsvariablen und einer feineren Partitionierung der Ein- und Ausgabebereiche, da es hier nicht mehr möglich ist, eine (annähernd) vollständige Regelbasis zu erstellen. Das folgende Kapitel untersucht einen Lösungsansatz, der es ermöglichen soll, Neuro-Fuzzy-Systeme auch bei komplexeren Problemen der Robotersteuerung anzuwenden.

Kapitel 6

Lernstrategie bei der Robotersteuerung

Mit dem in Abschnitt 5.1 erstellten Netz wird die betrachtete Steuerungsaufgabe zufriedenstellend ausgeführt. Allerdings wurde hierbei nur der Fuzzy-Anteil des Neuro-Fuzzy-Systems verwendet: das System wurde mit Expertenwissen initialisiert und dann direkt eingesetzt. Um die Fähigkeiten des Neuro-Fuzzy-Systems voll auszunutzen sollte das initialisierte Netz nun weiter trainiert und angepasst werden. Dies ist vor allem notwendig, wenn Neuro-Fuzzy-Netze auch für komplexere Systeme mit mehr Eingangsvariablen oder mit einer feineren Partitionierung der Ein- und Ausgangsräume verwendet werden soll.

Die Software NFident implementiert einen Lernalgorithmus, der auf dem in Abschnitt 2.1.2 eingeführten überwachten Lernen basiert. Bei der betrachteten Aufgabe, der Robotersteuerung, stehen jedoch keine Trainingsdatensätze zur Verfügung, mit denen das beschriebene Backpropagating durchgeführt werden könnte. Deswegen soll das in diesem Abschnitt beschriebene Neuro-Fuzzy-System in ein Softwaresystem integriert werden, welches das Reinforcement Learning (verstärkendes Lernen) implementiert. Zunächst wird im folgenden Abschnitt der Ansatz des Reinforcement Learning genauer beschrieben.

6.1 Reinforcement Learning

Reinforcement Learning bietet die Möglichkeit, ohne das Vorhandensein von Lösungswissen (z.B. in Form von Trainingsdatensätzen) ein Verhalten zu erlernen. Hierzu wird jede ausgeführte Aktion mit bewertet. Durch Rückkopplung werden die Bewertungen der Aktionen im Laufe des Lernprozesses angepasst.

Das Prinzip des Reinforcement Learning lässt sich in die folgenden Schritte gliedern [Volk03], [www08]:

- Jedem Zustand s werden mögliche Aktionen a zugeordnet (*Zustands-Aktions-*

Paar)

- Für jede ausgeführte Aktion wird eine festgelegte Belohnung (*reward*) vergeben.
- Jedem Zustands-Aktions-Paar wird eine Bewertung (*value*) zugeordnet. Diese Bewertung ist der (approximierte) Gesamtbeitrag der Belohnungen, die in Zukunft durch Akkumulation erwartet werden können, wenn von diesem Zustand aus gestartet wird.
- Für eine eintretende Situation wird mithilfe der *Policy* aus den Zustands-Aktions-Paaren das mit der höchsten Bewertung ausgewählt und die Aktion wird ausgeführt.
Um sicherzustellen, dass auch neue Zustands-Aktions-Paare ausprobiert werden, wählt die *Policy* mit einer bestimmten Wahrscheinlichkeit Zustands-Aktions-Paare aus, deren bisherige Bewertung nicht maximal ist.
- Die Bewertungen der Folgezustände werden verwendet, um die Bewertung des gewählten State-Action-Paares anzupassen.
In der Regel wird eine hohe Belohnung (*reward*) vergeben, wenn das Ziel erreicht wird, welche dann eine hohe Bewertung der in diesen Zustand führenden Zustands-Aktions-Paare bewirkt.

Wichtige Elemente sind hierbei die *Belohnungsfunktion* (reward-function), die *Bewertungsfunktion* (value-function) und die *Policy*.

Die Belohnungsfunktion r definiert das Ziel des Reinforcement-Learning-Problems. Für jeden ausgeführten Schritt wird eine Belohnung vergeben. Ziel des Lernens ist in der Regel, die Summe der Belohnungen bis zum Ziel zu maximieren. Eine Möglichkeit ist es, für jede Aktion die nicht zum Ziel führt eine negative Belohnung und nur für das Erreichen des Ziels eine positive Belohnung zu vergeben.

Die Belohnungsfunktion steht während des gesamten Lernvorgangs fest und kann nicht verändert werden.

Die Bewertungsfunktion $V(s, a)$ approximiert für jeden Zustand s und eine dazugehörige ausgeführte Aktion a die Summe der Belohnungen bis zum Erreichen des Zieles. Da das Ziel des Lernens die Maximierung der Belohnungen ist, wird entsprechend der Belohnungsfunktion die Aktion ausgewählt, die in den am höchsten bewerteten Folgezustand führt.

Die *Policy* π ist eine Handlungsstrategie, die in jedem Zustand eine auszuführende Aktion vorschlägt. Während des Lernvorgangs wird versucht, die *Policy* zu optimieren [www08].

Eine Art des Reinforcement Learning ist das Q-Learning. Hier wird anstelle der Bewertungsfunktion die Q-Funktion $Q(s, a)$ definiert. Diese gibt den Wert einer Aktion in einem bestimmten Zustand wieder. Die Q-Funktion wird nach jedem Lernschritt aktualisiert. Die Gleichung für diese Aktualisierung lautet

$$Q(s_t, a_t) := Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right] \quad (6.1)$$

Dabei ist t ein diskreter Zeitpunkt im Lernprozess, α ein Lernparameter und γ ein Diskontierungsfaktor (wird hier nicht benötigt)

Die Policy hat in diesem Fall vor allem die Aufgabe dafür zu sorgen, dass alle Zustands-Aktions-Paare immer wieder besucht werden. Hierzu verwendet man z.B. die sogenannte ϵ -greedy Strategie: Es wird mit der Wahrscheinlichkeit $Pr = (1 - \epsilon)$ die Aktion mit maximalem Erwartungswert Q ausgewählt. Mit der Wahrscheinlichkeit $Pr = \epsilon$ wird zufällig eine Aktion aus der Menge der möglichen Aktionen gewählt [Volk03].

Für das Q-Learning wird ein Funktionsapproximator benötigt, der die Q-Funktion approximiert und somit jeder Aktion a im Zustand s einen skalaren Wert zuordnet. Im folgenden Abschnitt soll erläutert werden, wie ein Neuro-Fuzzy-System als Funktionsapproximator für das Reinforcement Learning verwendet werden kann [www06].

6.2 Einbindung des NFident-Modells ins Reinforcement Learning

Ein Neuro-Fuzzy-Modell ähnlich dem in Abschnitt 5.1 beschriebenen System soll mithilfe des Reinforcement Learning lernen, einen Roboter möglichst optimal zu steuern. Das Szenario sei identisch mit der in Abschnitt 5.1 beschriebenen Aufgabe. Das Neuro-Fuzzy-Modell soll als Funktionsapproximator beim Q-Learning verwendet werden.

Zunächst ein paar grundsätzliche Überlegungen:

In Verbindung mit dem Reinforcement Learning ist es die Aufgabe des Neuro-Fuzzy-Netzes, für jede Aktion in Abhängigkeit vom aktuellen Zustand eine Bewertung abzugeben. Das in Abschnitt 5.1 beschriebene System S_1 schlägt zu jedem Zustand eine Aktion vor. Es muss nun so modifiziert werden, dass es zu jeder Kombination von Zustands- und Aktionsgrößen eine Bewertung ausgibt. Somit werden die Ausgangsgrößen von S_1 nun ebenfalls zu Eingangsgrößen. Das neue System S_2 hat also vier Eingangsgrößen *Distance*, *deltaTheta*, *outTrans* und *outRot* sowie eine Ausgangsgröße, die Bewertung (*value*).

Das Netz wird in das Reinforcement-Learning-System eingebunden und berechnet jeweils die Bewertungen $Q(s_t, a_t)$ und $Q(s_{t+1}, a_{t+1})$. So kann in jedem Lernschritt mithilfe des Neuro-Fuzzy-Netzes die optimale (also am höchsten bewertete) Aktion ausgewählt werden. Außerdem kann über Gleichung (6.1) der aktualisierte Wert $Q'(s_t, a_t)$ berechnet werden. Dieser Wert kann nun verwendet werden, um das Neuro-Fuzzy-Netz zu trainieren bzw. anzupassen: Zusammen mit dem ausgewählten Zustands-Aktions-Paar wird er als Trainingsdatensatz durch das Netz rückpropagiert. Das Netz passt sich mithilfe der in Definition 3.1.1 angegebenen Lernalgorithmen an und wird im nächsten Lernschritt weiterverwendet.

Wie in Abschnitt 4.2 gezeigt wurde, sollte das Neuro-Fuzzy-Netz wenn möglich nicht mit einzelnen Trainingsdatensätzen trainiert werden. Es sollte mit möglichst großen Trainingsmengen trainiert werden, da damit bessere Lernergebnisse erzielt werden können. Der Algorithmus des Reinforcement Learning kann dazu so modifiziert werden,

dass alle Trainingsdatensätze einer Episode gesammelt werden und nach Abschluss der Episode auf einmal eintrainiert werden.

Es stellt sich nun zunächst die Frage, ob sich die oben gemachten Überlegungen verwirklichen lassen und ein System entsteht, das eine Aufgabe wie die Steuerung des Roboters lernen kann. Um dies zu überprüfen wurde ein einfaches Testszenario entworfen und zum testen verwendet.

6.2.1 Reinforcement Learning mit einem einfachen Szenario

Um in einem ersten Schritt die Kombination des Neuro-Fuzzy-System mit dem Reinforcement Learning grundsätzlich zu testen wurde ein einfaches Szenario mit nur zwei Eingangsgrößen entworfen. Der Lernvorgang lässt sich bei einem solchen System in Form einer 3D-Graphik noch einfach graphisch darstellen. Folgende Aufgabe wird dem System gestellt: Auf einer Skala von 0 bis 100 soll die Position "50" erreicht werden indem von einem zufälligen Ausgangspunkt aus Schritte der Länge 4, 3, 2, 1, 0, -1, -2, -3 und -4 gemacht werden. Für jeden Schritt mit dem das Ziel nicht direkt erreicht wird, wird als *reward* eine "Bestrafung" von -5 vergeben. Für Schritte, die dem vorigen Schritt entgegengesetzt sind werden mit einem *reward* von -10 bestraft. Führt ein Schritt direkt zur Position "50" wird der *reward* 0 vergeben.

Mithilfe eines als Lookup-Tabel implementierten Funktionsapproximators im Reinforcement Learning lässt sich zeigen, dass die aus dieser Aufgabe resultierende Funktionstopologie etwa wie in Abbildung 6.1 aussehen muss. In der Abbildung sind die x-Achse, auf der die Position aufgetragen ist und die y-Achse, auf der die gewählte Schrittlänge darstellt wird jeweils auf 1 normiert.

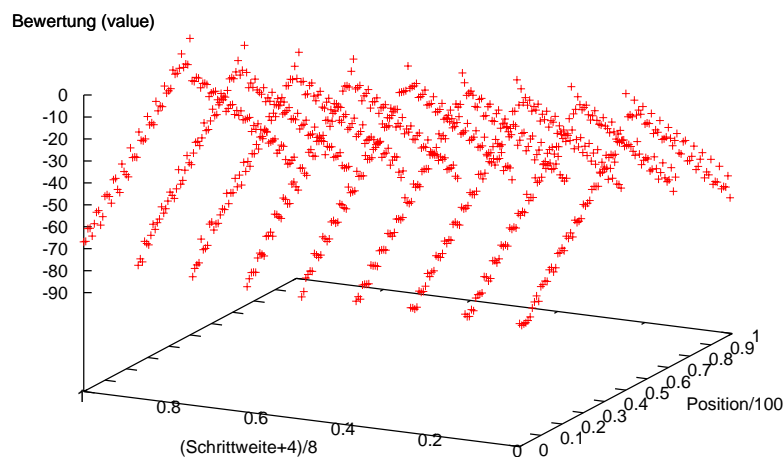


Abbildung 6.1: Ideale Topologie des Jumper-Szenarios

Eigenschaftsparameter	Wert	Bedeutung
Netztyp	0	Netz vom Typ NEFPROX/Mamdani
Anzahl der Eingangsgrößen	2	
Anzahl der Ausgangsgrößen	1	
Operator zur Verknüpfung der Regelprämissen	0	MIN
Defuzzifizierungsmethode	0	Center of Gravity
Anzahl von Fuzzy-Mengen pro Variable	16	
Form der Fuzzy-Mengen	0	Dreiecksfunktion
Lernrate σ	0.01	
maximale Regelanzahl	-1	beliebig
minimaler Fehler (Abbruchkrit.)	0.0	Lernen beenden wenn Fehler = 0.0
Abbruchkrit. $chepoch$	100	Lernen beenden wenn Fehler 100 Epochen lang nicht kleiner wird
min. Anzahl Epochen	0	
max. Anzahl Epochen	1000	
Lernrhythmus	1	batch-learning
Eigensch. der Fuzzy-Mengen: $nopass$	1	Fuzzy-Mengen dürfen während des Trainings nicht aneinander vorbeilaufen
$assym$	1	Fuzzy-Mengen dürfen während des Trainings assymetrisch werden

Tabelle 6.1: Eigenschaften des Netzes N_{J_1}

Die Anwendung eines NFident-Netzes in dieser Umgebung erweist sich jedoch als schwierig. Obwohl bewusst zunächst das relativ übersichtliche beschriebene Szenario "Jumper" gewählt wurde, um die Funktionalität der Neuro-Fuzzy-Netze in Verbindung mit dem Reinforcement Learning zu testen, zeigt sich, dass das Netz nicht in der Lage ist, die Aufgabe auszuführen. Wird ein nicht trainiertes Netz verwendet und trainiert, so zeigen sich auch nach mehreren tausend Lernepochen keinerlei Verbesserungen.

Um die Anforderungen zunächst noch zu verringern, wurden deswegen in einem zweiten Schritt mithilfe des Lookup-Tabel-Funktionsapproximators Trainingsdaten erzeugt, die die optimale Funktionstopologie für die betrachtete Aufgabe repräsentieren. Es wird außerhalb der Reinforcement-Learning-Umgebung ein Neuro-Fuzzy-Netz N_{J_1} erzeugt und mit diesen Daten trainiert. Die Eigenschaften des Netzes N_{J_1} sind in Tabelle 6.1 zusammengefasst.

Anschließend wird das Netz N_J in die Reinforcement-Learning-Umgebung eingebunden und es wird ein Lernprozess angestoßen. Abbildung 6.2 zeigt, wie das vortrainierte Netz die Topologie des Jumper-Szenarios approximiert. Hierbei zeigt sich, dass das Neuro-Fuzzy-Netz auch so nicht mit dem Reinforcement-Learning kombiniert werden kann. Obwohl das Netz zunächst recht gut in der Lage ist, die Aufgabe des Jumper-Szenarios zu erfüllen, wird es im Laufe des Lernprozesses sehr schnell immer schlechter, die dachförmige Topologie wird immer mehr abgeflacht, bis das Netz vollständig "zerstört" ist.

Der Grund für dieses Verhalten liegt in der Topologie des Beispiels. An der "Spitze" des Daches hat die Funktion teilweise eine Sprungstelle. Diese wird vom Netz N_J jedoch nicht abgebildet. Das Netz interpoliert zwischen den gegebenen Werten und es entsteht wie in Abbildung 6.2 zu sehen, ein Funktionsverlauf, der gerade im Zielbereich um die Position "50" nicht der gewünschten Funktion (im Bild rot) entspricht. Diese

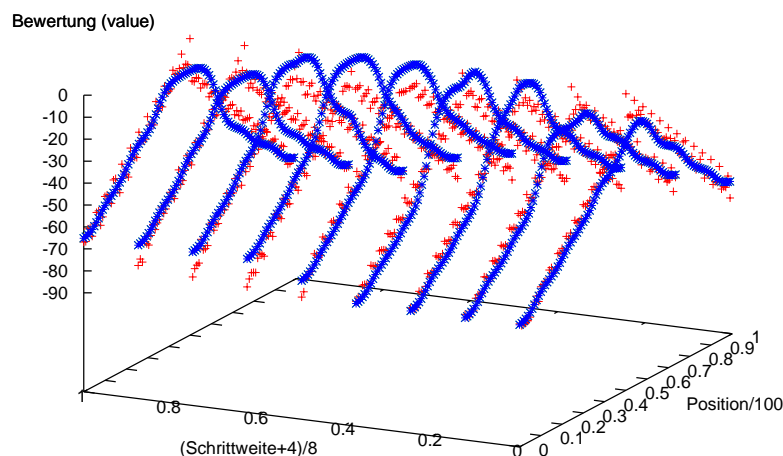


Abbildung 6.2: Ideale Topologie des Jumper-Szenarios (rot) und Approximation durch das vortrainierte Neuro-Fuzzy-Netz N_J (blau)

Ungenauigkeit führt dazu, dass das Netz sich mehr und mehr verschlechtert wenn man versucht es mithilfe des Reinforcement-Learning weiter zu verbessern.

Abhilfe schafft hierbei teilweise ein weiterer Trainingsschritt, bevor das Netz in die Reinforcement-Learning-Umgebung eingebunden wird: trainiert man das Netz N_J in einem zweiten Schritt mit 10 Trainingsdatensätzen aus dem Bereich Position = 50, für die alle gilt "Bewertung = 0", so bildet das Netz anschließend die Unstetigkeit in diesem Bereich der Funktionstopologie näherungsweise ab (Abbildung 6.3).

Bindet man das so entstandene Netz N_{J_2} in das Reinforcement-Learning ein, zeigen sich wesentliche Verbesserungen. Das Netz bleibt während des Lernvorgangs im Wesentlichen unverändert. Nach längerer Lern-Zeit treten jedoch weiterhin Verschlechterungen in der approximierten Topologie auf. Wird nun zusätzlich noch die Schrittweite α aus Gleichung 6.1 von 0.1 auf 0.01 verringert, ist während des Lernprozesses eine leichte Verbesserung der Funktionsapproximation durch N_{J_2} erkennbar.

Bisher wurde versucht, ein vortrainiertes Netz in das Reinforcement Learning einzubinden und es so weiter zu optimieren. Im Normalfall sind jedoch nicht ausreichend Trainingsdaten vorhanden, um ein Netz so vorzutrainieren. Wie in Abschnitt 5.1 gezeigt ist es jedoch möglich, ein Netz von Hand vor zu modellieren. Diese Möglichkeit wird im folgenden Abschnitt betrachtet.

6.2.1.1 Reinforcement Learning mit einem vormodellierten Netz

In einem zweiten Schritt soll nun an dem gegebene Jumper-Szenario getestet werden, wie sich ein durch Vorgabe von Regeln modelliertes Netz N_{J_3} in Verbindung mit dem

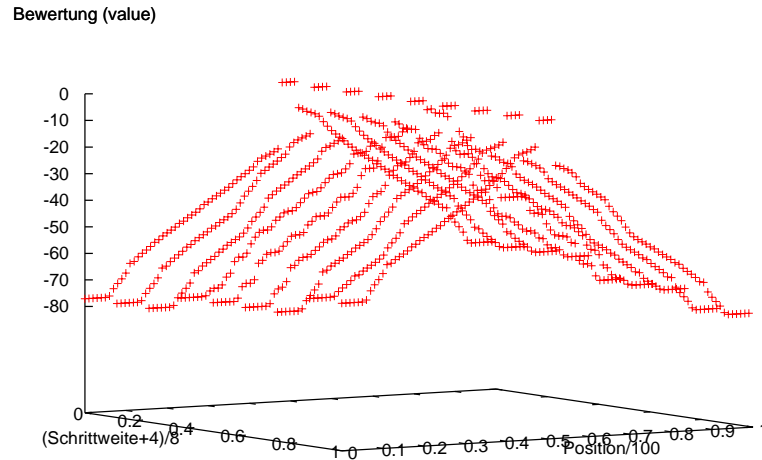


Abbildung 6.3: *Ideale Topologie des Jumper-Szenarios (rot) und Approximation durch das mit zwei Trainingsmengen trainierte Neuro-Fuzzy-Netz (blau)*

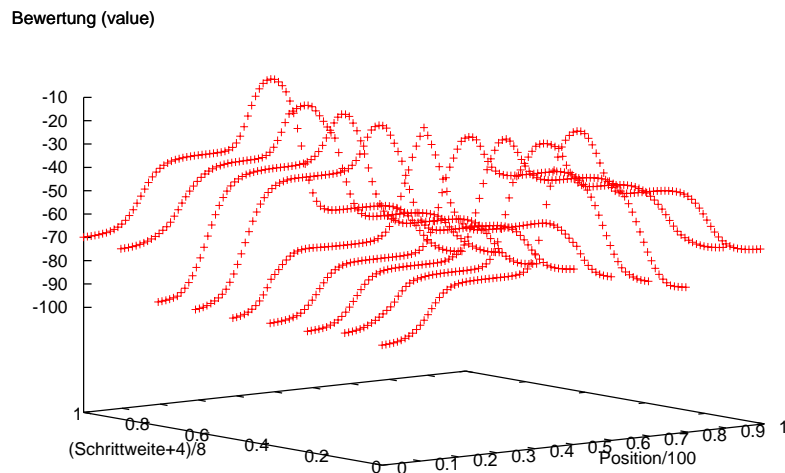
Reinforcement Learning verhält. Es wird deswegen versucht, ein Neuro-Fuzzy-Netz zu erstellen, das die gegebene Aufgabe erfüllt. Um die Anzahl der vorzugebenden Regeln in Grenzen zu halten werden die Ein- und Ausgangsgrößen durch nur jeweils 5 Fuzzy-Mengen partitioniert. Tabelle 6.2 fasst die Eigenschaften des Netzes N_{J_3} zusammen.

Idealerweise sollte ein Netz, welches die Funktionstopologie bereits grob richtig approximiert (welches also in etwas die Dachform aus Abbildung 6.1 annähert) in die Reinforcement-Learning-Umgebung integriert werden können, und dann innerhalb des Lernvorganges weiter angepasst werden. Bei zwei Eingangsgrößen und einer Ausgangsgröße mit je fünf Fuzzy-Mengen ergibt sich eine Regelmenge von maximal $5^2 = 25$ Regeln. Damit ist das Netz eindeutig definiert. Zunächst werden die Ein- und Ausgabebereiche mehr oder weniger willkürlich durch die fünf Fuzzy-Mengen partitioniert. Bei der Anpassung des Netzes während des Lernvorganges können sie angepasst werden, so dass sie in Kombination mit den (vorgegebenen) Regeln die Lernaufgabe erfüllen. Abbildung zeigt die Topologie eines möglichen Netzes. In Anhang A.3 ist die zu N_{J_3} gehörende Netzdatei angegeben.

Aufgrund der wenigen zur Verfügung stehenden Fuzzy-Mengen unterscheidet sich die Topologie recht deutlich von der durch die Trainingsmengen modellierten Topologie. Allerdings sind die grundlegenden Eigenschaften enthalten:

- Wenn die Entfernung zum Zielpunkt "50" betragsmäßig groß ist, ist die Bewertung eines Zustands-Aktions-Paares sehr klein
- Bei Positionen < 50 werden positive Schrittweiten höher bewertet als negative,

Eigenschaftsparameter	Wert	Bedeutung
Netztyp	0	Netz vom Typ NEFPROX/Mamdani
Anzahl der Eingangsgrößen	2	
Anzahl der Ausgangsgrößen	1	
Operator zur Verknüpfung der Regelprämissen	0	MIN
Defuzzifizierungsmethode	0	Center of Gravity
Anzahl von Fuzzy-Mengen pro Variable	5	
Form der Fuzzy-Mengen	2	Gaußsche Glockenkurve
Lernrate σ	0.01	
maximale Regelanzahl	-1	beliebig
minimaler Fehler (Abbruchkrit.)	0.0	Lernen beenden wenn Fehler = 0.0
Abbruchkrit. $chgepoch$	100	Lernen beenden wenn Fehler 100 Epochen lang nicht kleiner wird
min. Anzahl Epochen	0	
max. Anzahl Epochen	1000	
Lernrhythmus	1	batch-learning
Eigensch. der Fuzzy-Mengen: $nopass$	1	Fuzzy-Mengen dürfen während des Trainings nicht aneinander vorbeilaufen
$assym$	1	Fuzzy-Mengen dürfen während des Trainings assymetrisch werden

Tabelle 6.2: Eigenschaften des Netzes N_{J_3} Abbildung 6.4: Approximierte Funktionstopologie für das Szenario "Jumper" durch das von Hand modellierte Netz N_{J_3}

bei Positionen > 50 werden entsprechend negative Schrittweiten höher bewertet

- Schritte die zum Ziel (bzw. nahe ans Ziel heran) führen werden am höchsten bewertet.

Idealerweise sollte das Reinforcement Learning das Netz N_{J_3} nun so beeinflussen, dass die Approximation sich langsam der gewünschten Topologie anpasst und die gestellte Aufgabe erfüllt wird. In verschiedenen Tests ergab sich jedoch, dass das vortrainierte Netz im Laufe des Lernvorganges nicht weiter verändert wird.

6.2.2 Anpassung des Netzes zur Robotersteuerung

Trotz der wenig vielversprechenden Ergebnisse wurde auch für das in Abschnitt 5.1 beschriebene Szenario, in welchem ein Roboter einen Ball anfahren soll, ein Neuro-Fuzzy-Netz erstellt, welches statt zu einem Zustand eine Aktion auszugeben Zustands-Aktionspaare bewertet. In einem gegebenen Zustand wird jeweils die am höchsten bewertete Aktion ausgeführt. Das manuell erstellte Netz ist in Anhang A.4 abgegeben.

In diesem Fall war es nicht möglich, alle Regeln von Hand zu erstellen. Dies wären bei 4 Eingangsvariablen die durch je 5 Fuzzy-Mengen partitioniert sind $4^5 = 625$ Regeln. Eine solche Regelmenge kann nicht mehr ohne Weiteres von Hand erstellt werden. Dieses Beispiel macht deutlich, wie schnell die Übersichtlichkeit bezüglich der Regelmenge verloren geht, selbst wenn eine relativ grobe (und damit in der Anwendung schlechtere!) Partitionierung der Variablen gewählt wird. Es ist jedoch möglich, Regeln zu erstellen, die nur auf einen Teil der Eingangsvariablen beziehen und deren Konklusion unabhängig von allen anderen Eingangsvariablen ist. Dadurch wird die Abdeckung des Eingangsraumes besser, auch wenn die formulierten Regeln weniger präzise sind. In diesem Fall soll das erstellte Netz lediglich als Ausgangspunkt für das Reinforcement Learning verwendet werden, so dass eine solche ungenaue Formulierung möglich ist. Das Netz hat somit nur 74 Regeln.

Auf Grund der in Abschnitt 6.2.1 beschriebenen Probleme die sich bei der Kopplung von Neuro-Fuzzy-Systemen mit dem Reinforcement Learning ergeben, war es im zeitlichen Rahmen dieser Arbeit nicht möglich, zu untersuchen, welche Verbesserungen das Reinforcement Learning für die Robotersteuerung mithilfe von Neuro-Fuzzy-Netzen liefert.

Kapitel 7

Zusammenfassung und Ausblick

Die Software NFident wurde dahingehend angepasst, dass sie für die Steuerung eines Roboters des Instituts für Verteilte und Parallele Systeme im Rahmen des RoboCup eingesetzt werden kann. Dabei wurde die Software zunächst im Hinblick auf die Einbindung in andere Softwaresysteme modifiziert.

In einigen Tests wurde gezeigt, dass die Software sehr gut in der Lage ist, Funktionen zu approximieren, auch wenn nur unvollständige Informationen über die gewünschte Funktion vorhanden sind. Da die Steuerung des Roboters im Grunde genommen ebenfalls die Approximation einer (unbekannten) Funktion darstellt, erschien es wahrscheinlich, dass die Software auch bei der Robotersteuerung gute Ergebnisse erzielt. Allerdings erwies sich die genannte Anwendung als problematisch. Grund hierfür ist vor allem, dass im Falle der Robotersteuerung keine konkreten Daten zum Trainieren des Neuro-Fuzzy-Netzes vorliegen. Es ist im Allgemeinen nicht bekannt, welche Aktion der Roboter in einem bestimmten Zustand idealerweise ausführen sollte.

Um das Neuro-Fuzzy-Netz trotzdem trainieren zu können, wurde das Neuro-Fuzzy-Modell in eine Reinforcement-Learning-Umgebung eingebunden. Es wurde ein Neuro-Fuzzy-System verwendet, um in jedem Zustand eine optimale Aktion auszuwählen. Diese Anwendung eines Neuro-Fuzzy-Netzes als Approximator der Q-Funktion beim Reinforcement Learning führte jedoch zu keinem zufriedenstellenden Ergebnis. Infolgedessen gestaltete sich die Verwendung von Neuro-Fuzzy-Netzen zur Robotersteuerung schwierig und es konnten nur wenige und recht einfache Netze zur Steuerung erstellt und getestet werden. Ein Weg, das steuernde Neuro-Fuzzy-Netz während der Anwendung weiter zu trainieren wurde nicht gefunden.

Aufgrund der Ergebnisse aus Abschnitt 4.2 bei der Funktionsapproximation ist es wahrscheinlich, dass die betrachteten Neuro-Fuzzy-Systeme trotz der wenig erfolgversprechenden Ergebnisse dieser Arbeit sehr wohl für die Steuerung eines Roboters eingesetzt werden können. Es muss lediglich ein Weg gefunden werden, geeignete Trainingsdaten zu generieren, um ein von Hand vormodelliertes Netz wie aus Abschnitt 5.1 während des Einsatzes weiter zu trainieren und anzupassen.

Anhang A

Netz- und Parameterdateien der Software NFident

A.1 NFident Parameter-Datei

Im folgenden ist beispielhaft eine Parameterdatei der Software NFident dargestellt. Über die Parameterdatei können die Netz- und Lernparameter eines Neuro-Fuzzy-Systems eingestellt werden.

Die einzelnen Parameter sind in der Datei selbst in Kommentaren erklärt.

```
#Parameter file for NFIDENT
#-----
#
#comment lines have a # or a % in the first column
#comments may be placed after a numeric parameter, but not after a string
#blank lines are ignored

#select a fuzzy system type (valid entries: 0 = Mamdani, 1 = TSK)
#Mamdani will be implemented by NEFPROX, TSK by ANFIS
fstype 0

#number of fuzzy sets for input (imsf) and output (omsf) variables
imsf 7
omsf 7

#to set the number of fuzzy sets for a specific variable only use e.g.
#imsf[1] 7
#omsf[0] 9
#remember: the first variable has index 0
#use after or instead of imsf and omsf commands

#forms of fuzzy sets for input (iform) and output (oform) variables
#0=triangular, 1=trapezoidal, 2=gaussian
iform 0
oform 0

#to set the fuzzy set type for a specific variable only use e.g.
#iform[1] 1
```

```

#oform[1] 1
#remember: the first variable has index 0
#use after or instead of iform and oform commands

#specify a percentage value by which the domains of the output variables are
#extended to the left and to the right, to better obtain output values
#which are near to the boundaries of the output domains (e.g. 0.2 = 20%)
outstretch 0.0

#select a t-norm for the fuzzy system (valid values: 0 = min, 1 = product)
tnorm 0

#select a t-conorm for the fuzzy system(valid values: 0 = max)
tconorm 0

#select a defuzzification strategy (for Mamdani only)
#valid entries are: 0 = COG, 1 = MOM
defuzz 1

#initialization for output parameters of a TSK fuzzy system, the
#coefficients are randomly chosen from [-x,+x] if x is specified below
tskinit 0.01

#learning rates can be set individually for each parameter (sigma_t is
#for the coefficients of a TSK model)
sigma_a 0.01
sigma_b 0.01
sigma_c 0.01
sigma_d 0.01
sigma_t 0.01

#to set all learning rates to an identical value, use the sigma command
sigma 0.01

#maximal number of rules, -1 means to use all rules found in the data
#(no evaluation), 0 means automatic evaluation (use percent), any positive
#number specifies the maximum number of admissible rules
maxrules -1

#percentage of all rule activations that has to be reached during rule
#evaluation if maxrules = 0 (automatic rulebase evaluation)
percent 0.75

#stop learning at this error value
minerror 0.0

#Epoch Definitions
#-----
#
#chgepoch determines after for how many epochs learning continues, after
#the smallest error so far was reached.
chgepoch 100
#minepoch specifies the minimum number of epochs for the learning procedure
minepoch 0
#maxepoch is the maximum number of epochs for the learning procedure
maxepoch 2000
#logepoch determines after how many epochs the log-file is updated
logepoch 10

#Learning Restrictions
#-----

```

```

#
#batch = 1 means batch learning, batch = 0 means online learning
batch 1
#at05 = 1 means fuzzy sets must intersect at height 0.5. This is not
#yet implemented
at05 0
#nopass = 1 means that fuzzy sets must not pass each other during training
nopass 1
#asym = 1 means that membership functions may become asymmetric
#during training (only evaluated for triangular membership functions)
asym 1

#FILENAMES
#-----
#
#netfilename: file of the trained fuzzy system
netfilename mg.nfs

#logfile: file where the course of error is logged
logfile mg.log

#gnufilename: file with gnuplot commands to plot the fuzzy sets
#usage: gnuplot <gnufilename>
gnufilename mg.gnu

#trainingfile: file with training data
trainingfile mg1.dat

#generalizationfile: file with validation data, leave out, if there is
#no validation data
generalizationfile mg2.dat

#testfile: file with test data, used after training. Leave out, if there
#is no test data
#testfile mg2.dat

#loadnet: file with fuzzy system to be loaded before training
#Leave out, if no fuzzy system shall be loaded
#loadnet mg.nfs

#trainresult: file where the target outputs and actual outputs of the
#training data are stored after training
(1 row: targets, 2 row: actual outputs, 3 row: target - output)
trainresult mg1.res

#genresult: for validation results, compare trainresult
genresult mg2.res

#testresult: for test results, compare trainresult
testresult mg3.res

#nfident saves the best fuzzy system during training to the file given
#by savenet (default is saved.nfs). If a * is used within the filename,
#the epoch number will be inserted at this position. By this you get
#a history of the best fuzzy systems during training.
#(there is still a bug in the filename creation routine)
#savenet saved*.nfs

```

A.2 NFident Netz-Datei für Robotersteuerung (1)

Die folgenden Angaben entsprechen der Netzdatei, die für die Steuerung eines Roboters in Abschnitt 5.1 von Hand erstellt wurde.

Die einzelnen Angaben sind in der Datei selbst durch Kommentare erklärt.

```
# Network file of NFIDENT
# Type of Fuzzy System (0 = Mamadani, 1 = TSK)
TYPE
0

# Dimensions of the fuzzy system: inputs, rules, outputs
DIMENSIONS
2 25 2

# T-norm to use with the fuzzy system (0 = min, 1 = product).
T-NORM
0

# T-conorm to use with the fuzzy system (0 = max)
T-CONORM
0

# Defuzzyfication procedure (0 = COG, 1 = MOM)
DEFUZZIFICATION
1

# Parameters of the fuzzy sets for all variables
# For each variable: number of fuzzy sets, lower and
# upper bounds, name, then a row for each fuzzy set.
# Parameters: fstype, a, b, c, d, ls, rs, name
# First input then output variables.
VARIABLES
5 0.000000 12000.0 Distance
2 300.000000 0.000000 0.000 0.000000 1 0 zero
2 800.000000 1000.00 0.000 0.000000 0 0 veryClose
2 2000.000 2500.000 0.000 0.000000 0 0 close
2 5000.000 6000.000 0.00 0.000000 0 0 far
2 8000.000 12000.000 0.00 0.000000 0 1 veryFar

5 -200.0 200.0 deltaTheta
2 130.0 -200.0 0.0 0.000000 1 0 greatNegative
2 50.0 -70.0 0.0 0.000000 0 0 negative
2 30.0 0.0 0.0 0.000000 0 0 zero
2 50.0 70.0 0.0 0.000000 0 0 positive
2 130.0 200.0 0.0 0.000000 0 1 greatPositive

5 0.0 1600.0 outVel
2 90.0 0.0 00.0 0.000000 1 0 zero
2 150.0 200.0 0.0 0.000000 0 0 verySlow
2 500.0 600.0 0.0 0.000000 0 0 slow
2 500.0 1000.0 0.0 0.000000 0 0 fast
2 500.0 1600.0 0.0 0.000000 0 1 veryFast

5 -280.0 280.0 outRot
2 200.0 -280.0 0.0 0.000000 1 0 fastNegative
2 150.0 -200.0 0.0 0.000000 0 0 slowNegative
```

```

2 100.0 0.0 0.0 0.000000 0 0 zero
2 150.0 200.0 0.0 0.000000 0 0 slowPositive
2 200.0 280.0 0.0 0.000000 0 1 fastPositive

# Rule base of the fuzzy system.
# One row for each rule, the entries are indices of
# fuzzy sets (counting from 0, -1 means the respective variable is not used).

RULES
0 0 0 4
0 1 0 4
0 2 0 2
0 3 0 0
0 4 0 0
1 0 0 4
1 1 1 4
1 2 1 2
1 3 1 0
1 4 0 0
2 0 2 4
2 1 2 3
2 2 2 2
2 3 2 1
2 4 2 0
3 0 3 4
3 1 3 3
3 2 3 2
3 3 3 1
3 4 3 0
4 0 4 4
4 1 4 3
4 2 4 2
4 3 4 1
4 4 4 0

# End of NFIDENT network file

```

A.3 NFident Netz-Datei für Jumper-Szenario

Die folgenden Angaben entsprechen der Netzdatei, die für das Jumper-Szenario in Abschnitt 6.2 von Hand erstellt wurde.

Die einzelnen Angaben sind in der Datei selbst durch Kommentare erklärt.

```

# Network file of NFIDENT
# Type of Fuzzy System (0 = Mamadani, 1 = TSK)
TYPE
0
# Dimensions of the fuzzy system: inputs, rules, outputs
DIMENSIONS
2 25 1

```

66ANHANG A. NETZ- UND PARAMETERDATEIEN DER SOFTWARE NFIDENT

T-norm to use with the fuzzy system (0 = min, 1 = product).

T-NORM

0

T-conorm to use with the fuzzy system (0 = max)

T-CONORM

0

Defuzzyfication procedure (0 = COG, 1 = MOM)

DEFUZZIFICATION

0

Parameters of the fuzzy sets for all variables

For each variable: number of fuzzy sets, lower and

upper bounds, name, then a row for each fuzzy set.

Parameters: fstype, a, b, c, d, ls, rs, name

First input then output variables.

VARIABLES

5 0.000000 1.000000 Position

2 0.200000 0.000000 0.000000 0.000000 1 0 weitKleiner50

2 0.200000 0.300000 0.000000 0.000000 0 0 nahKleiner50

2 0.100000 0.500000 0.000000 0.000000 0 0 null

2 0.200000 0.700000 0.000000 0.000000 0 0 nahGroesser50

2 0.200000 1.000000 0.000000 0.000000 0 1 wetGroesser50

5 0.000000 1.000000 Bewegung

2 0.200000 0.000000 0.000000 0.000000 1 0 schnellPos

2 0.200000 0.300000 0.000000 0.000000 0 0 langsamPos

2 0.100000 0.500000 0.000000 0.000000 0 0 null

2 0.200000 0.700000 0.000000 0.000000 0 0 langsamNeg

2 0.200000 1.000000 0.000000 0.000000 0 1 schnellNeg

5 -100.00000 10.0000000 Bewertung

2 20.00000 -100.00000 0.000000 0.000000 1 0 sehrSchlecht

2 20.000000 -70.000000 0.000000 0.000000 0 0 schlecht

2 20.000000 -40.000000 0.000000 0.000000 0 0 mittel

2 30.000000 0.000000 0.0000000 0.000000 0 0 gut

2 9.0000000 10.00000000 0.00000000 0.000000 0 1 sehrGut

Rule base of the fuzzy system.

One row for each rule, the entries are indices of

fuzzy sets (counting from 0, -1 means the respective variable is not used).

RULES

0 0 0

0 1 0

0 2 0

0 3 0

0 4 1

1 0 1

1 1 1

1 2 1

1 3 2

1 4 2

2 0 3

2 1 4

2 2 4

2 3 4

```

2 4 3
3 0 2
3 1 2
3 2 1
3 3 1
3 4 1
4 0 1
4 1 0
4 2 0
4 3 0
4 4 0

```

A.4 NFident Netz-Datei für Robotersteuerung (2)

Die folgenden Angaben entsprechen der Netzdatei, die für die Robotersteuerung in Kombination mit dem Reinforcement Learning in Abschnitt 6.2 von Hand erstellt wurde.

Die einzelnen Angaben sind in der Datei selbst durch Kommentare erklärt.

```

# Network file of NFIDENT
# Type of Fuzzy System (0 = Mamadani, 1 = TSK)
TYPE
0
# Dimensions of the fuzzy system: inputs, rules, outputs
DIMENSIONS
4 74 1
# T-norm to use with the fuzzy system (0 = min, 1 = product).
T-NORM
0
# T-conorm to use with the fuzzy system (0 = max)
T-CONORM
0
# Defuzzyfication procedure (0 = COG, 1 = MOM)
DEFUZZIFICATION
1
# Parameters of the fuzzy sets for all variables
# For each variable: number of fuzzy sets, lower and
# upper bounds, name, then a row for each fuzzy set.
# Parameters: fstype, a, b, c, d, ls, rs, name
# First input then output variables.
VARIABLES
5 0.000000 12000.0 Distance
2 300.000000 0.000000 0.000 0.000000 1 0 zero
2 800.000000 1000.00 0.000 0.000000 0 0 veryClose
2 2000.000 2500.000 0.000 0.000000 0 0 close
2 5000.000 6000.000 0.00 0.000000 0 0 far
2 8000.000 12000.000 0.00 0.000000 0 1 veryFar

```

68ANHANG A. NETZ- UND PARAMETERDATEIEN DER SOFTWARE NFIDENT

```
5 -200.0 200.0 deltaTheta
2 130.0 -200.0 0.0 0.000000 1 0 greatNegative
2 65.0 -70.0 0.0 0.000000 0 0 negative
2 5.0 0.0 0.0 0.000000 0 0 zero
2 65.0 70.0 0.0 0.000000 0 0 positive
2 130.0 200.0 0.0 0.000000 0 1 greatPositive

5 0.0 1600.0 outVel
2 90.0 0.0 0.0 0.000000 1 0 zero
2 150.0 200.0 0.0 0.000000 0 0 verySlow
2 500.0 600.0 0.0 0.000000 0 0 slow
2 500.0 1000.0 0.0 0.000000 0 0 fast
2 500.0 1600.0 0.0 0.000000 0 1 veryFast

5 -280.0 280.0 outRot
2 200.0 -280.0 0.0 0.000000 1 0 fastNegative
2 150.0 -200.0 0.0 0.000000 0 0 slowNegative
2 100.0 0.0 0.0 0.000000 0 0 zero
2 150.0 200.0 0.0 0.000000 0 0 slowPositive
2 200.0 280.0 0.0 0.000000 0 1 fastPositive

5 -500.0 500.0 value
2 350.0 -500.0 0.0 0.000000 1 0 veryBad
2 200.0 -250.0 0.0 0.000000 0 0 bad
2 100.0 0.0 0.0 0.000000 0 0 zero
2 200.0 250.0 0.0 0.000000 0 0 good
2 350.0 500.0 0.0 0.000000 0 1 veryGood

# Rule base of the fuzzy system.
# One row for each rule, the entries are indices of
# fuzzy sets (counting from 0, -1 means the respective variable is not used).
```

RULES

```
0 0 0 4 4
0 1 0 4 4
0 2 0 2 4
0 3 0 0 4
0 4 0 0 4
1 0 0 4 4
1 1 1 4 4
1 2 1 2 4
1 3 1 0 4
1 4 0 0 4
2 0 2 4 4
2 1 2 3 4
2 2 2 2 4
2 3 2 1 4
2 4 2 0 4
3 0 3 4 4
3 1 3 3 4
3 2 3 2 4
3 3 3 1 4
3 4 3 0 4
4 0 4 4 4
4 1 4 3 4
4 2 4 2 4
```

```
4 3 4 1 4
4 4 4 0 4
0 -1 0 -1 4
0 -1 1 -1 3
0 -1 2 -1 2
0 -1 3 -1 1
0 -1 4 -1 0
1 -1 0 -1 3
1 -1 1 -1 4
1 -1 2 -1 3
1 -1 3 -1 1
1 -1 4 -1 0
2 -1 0 -1 2
2 -1 1 -1 3
2 -1 2 -1 3
2 -1 3 -1 4
2 -1 4 -1 2
3 -1 0 -1 0
3 -1 1 -1 1
3 -1 2 -1 2
3 -1 3 -1 3
3 -1 4 -1 4
4 -1 0 -1 0
4 -1 1 -1 0
4 -1 2 -1 1
4 -1 3 -1 2
4 -1 4 -1 4
-1 0 -1 0 0
-1 0 -1 1 0
-1 0 -1 2 1
-1 0 -1 3 3
-1 0 -1 4 4
-1 1 -1 0 0
-1 1 -1 1 0
-1 1 -1 2 1
-1 1 -1 3 3
-1 1 -1 4 4
-1 2 -1 0 0
-1 2 -1 1 1
-1 2 -1 2 4
-1 2 -1 3 1
-1 2 -1 4 0
-1 3 -1 0 3
-1 3 -1 1 4
-1 3 -1 2 1
-1 3 -1 3 0
-1 3 -1 4 0
-1 4 -1 0 4
-1 4 -1 1 2
-1 4 -1 2 1
-1 4 -1 3 0
-1 4 -1 4 0
```

```
# End of NFIDENT network file
```


Anhang B

Beschreibung der implementierten Funktionen

B.1 NFidentNet

Methode	Beschreibung
<code>bool loadNet(const string& netfilename)</code>	Lädt ein bestehendes Netz aus einer Netzdatei (<code>netfilename</code>)
<code>bool loadNetAndData(const string& netfilename, const string& datafilename)</code>	Lädt ein bestehendes Netz aus einer Netzdatei (<code>netfilename</code>) sowie Trainingsdaten aus einer Datendatei (<code>datafilename</code>)
<code>bool testNet(const string& netfilename, const string& datafilename)</code>	Lädt ein bestehendes Netz aus einer Netzdatei (<code>netfilename</code>) sowie Trainingsdaten aus einer Datendatei (<code>datafilename</code>) und testet das Netz mit diesen Daten. Schreibt das Ergebnis in eine Datei <code>result.dat</code>
<code>void plotNet(const string& plotfilename)</code>	Erzeugt eine GNU-PLOT-Datei (<code>plotfilename</code>) die das zur Zeit geladene Netz graphisch darstellt
<code>saveNet(const string& netfilename)</code>	Speichert das zur Zeit geladene Netz unter <code>netfilename</code>
<code>newNet(const string& commandfilename, const string& trainingfilename, const string& finalnetfilename, const string& generalizationfilename, const string& initialnetfilename)</code>	Erzeugt ein neues Netz. Netzparameter werden aus <code>commandfilename</code> geladen, Trainingsdaten aus <code>trainingfilename</code> . Das neue Netz wird unter <code>finalnetfilename</code> gespeichert. Die weiteren Parameter sind fakultativ: <code>generalizationfilename</code> enthält Generalisierungsdaten, <code>initialnetfilename</code> enthält ein Netz, welches geladen und mit den gegebenen Angaben trainiert wird

Tabelle B.1: Wichtige Methoden der Klasse NFidentNet (1)

Methode	Beschreibung
<code>int getDimensions()</code>	Die Dimensionen des zur Zeit geladenen Netzes werden in <code>_inputDim</code> und <code>_outputDim</code> gespeichert
<code>getDimensions(const string& netfilename)</code>	Die Dimensionen eines in der Datei <code>netfilename</code> gespeicherten Netzes werden in <code>_inputDim</code> und <code>_outputDim</code> gespeichert
<code>int getNumberOfRules()</code>	Gibt die Anzahl der Regel des zur Zeit geladenen Netzes zurück
<code>void setNetType(int netType)</code>	Legt den Netztyp fest (<code>netType=0</code> \Rightarrow NEF-PROX, <code>netType=1</code> \Rightarrow ANFIS)
<code>void setDefuzzification(int defuzz)</code>	Legt die Defuzzifizierungsstrategie fest (<code>defuzz=0</code> \Rightarrow Center of Gravity, <code>defuzz=1</code> \Rightarrow Mean of Maximum)
<code>void setDimensions(int inputDim,int outputDim)</code>	Legt die Anzahl der Dimensionen fest
<code>void setInputFuzzySets(int numberInputSets, int formInputSets)</code>	Legt die Anzahl und Form der Fuzzy-Mengen der Eingangsvariablen fest (<code>formInputSets=0</code> \Rightarrow Dreieck, <code>formInputSets=1</code> \Rightarrow Trapez, <code>formInputSets=0</code> \Rightarrow gaußsche Glockenkurve)
<code>void setOutputFuzzySets(int numberOutputSets, int formOutputSets)</code>	Legt die Anzahl und Form der Fuzzy-Mengen der Ausgangsvariablen fest (<code>formOutputSets=0</code> \Rightarrow Dreieck, <code>formOutputSets=1</code> \Rightarrow Trapez, <code>formOutputSets=0</code> \Rightarrow gaußsche Glockenkurve)
<code>void setTNorm(int tnorm)</code>	Legt fest mit welcher t-norm die Regelprämissen verknüpft werden (<code>tnorm=0</code> \Rightarrow MIN, <code>tnorm=1</code> \Rightarrow Produkt)
<code>void setLearningRate(double sigma_a, double sigma_b, double sigma_c, double sigma_d, double sigma_t)</code>	Legt die Lernrate σ fest
<code>void setEpochDef(int chgepoch, int maxepoch, int minepoch, int logepoch)</code>	Definiert, nach wievielen Epochen das Training abgebrochen wird (<code>maxepoch</code>), wieviele Epochen lang mindestens trainiert wird (<code>minepoch</code>), wie oft das Netz zwischengespeichert wird (<code>logepoch</code>) und nach wievielen Epochen, in denen sich der Fehler nicht ändert, das Training beendet wird (<code>chgepoch</code>)
<code>void setLearningRestrictions(int batch, int nopass, int asym)</code>	Definiert den Lernvorgang: <code>batch=0</code> \Rightarrow online Lernen, <code>batch=1</code> \Rightarrow batch Lernen, <code>nopass=1</code> \Rightarrow Fuzzy-Mengen dürfen während des Trainings nicht aneinander vorbei laufen, <code>asym=1</code> \Rightarrow dreieckförmige Fuzzy-Mengen dürfen während des Trainings assymetrisch werden
<code>void setMaxRules(int maxrules)</code>	Legt die maximale Anzahl von Regeln fest
<code>void loadRules(int** rulebase, int numberOfRules)</code>	Lädt Regeln aus <code>rulebase</code>
<code>void CreateFuzzySystem()</code>	Generiert entsprechend der gesetzten Parameter ein neues, untrainiertes Netz. Achtung: es muss mindestens ein Satz von Trainingsdaten geladen sein.

Tabelle B.2: Wichtige Methoden der Klasse NFidentNet (2)

Methode	Beschreibung
<code>void readTrainingData(double** pattern, int numberOfPatterns)</code>	Liest Trainingsdaten aus <code>pattern</code> ein
<code>void readGeneralizationData(double** pattern, int numberOfPatterns)</code>	Liest Generalisierungsdaten aus <code>pattern</code> ein
<code>void trainNet()</code>	Trainiert das zur Zeit geladene Netz mit den eingelesenen Trainings- und Generalisierungsdaten
<code>void propagatePattern(double* invec, double* outvec)</code>	Speichert die Ausgabe des Netzes für den Eingabevektor <code>invec</code> in <code>outvec</code>
<code>backpropagatePattern(double* invec, double* outvecTarget, int epoch)</code>	Trainiert das Netz mit einem einzelnen Trainingsdatensatz bestehend aus den Eingangswerten <code>invec</code> und den Ausgangswerten <code>outvecTarget</code> . Maximale Anzahl von Epochen wird durch <code>epoch</code> vorgegeben.

Tabelle B.3: Wichtige Methoden der Klasse NFidentNet (3)

Über diese Klasse kann auf alle Funktionalitäten der Software NFident der Universität Magdeburg zugegriffen werden.

Es können Neuro-Fuzzy-Netze vom Typ NEFPROX oder ANFIS erzeugt, geladen, gespeichert und trainiert werden, sowie alle Netz- und Lernparameter gesetzt werden.

Abbildungsverzeichnis

2.1	Neuronales feedforward Netz	5
2.2	Zugehörigkeitsfunktionen	8
2.3	Fuzzy-Partitioning	8
2.4	Verknüpfung von Regelprämissen	9
2.5	Verknüpfung von Regelkonklusionen	10
2.6	Defuzzifizierungsmethoden	11
2.7	Fuzzy-Regler	11
3.1	NEPROX-System	16
3.2	ANFIS-System	22
4.1	Zielfunktion und Trainingsfunktion	29
4.2	Approximierter Funktionsverlauf zu Beginn	30
4.3	Approximierter Funktionsverlauf nach Training mit W_1	30
4.4	Approximierter Funktionsverlauf nach Training mit W_2	31
4.5	Approximierter Funktionsverlauf nach Training mit W_3	31
4.6	Approximierter Funktionsverlauf nach Training mit W_1 und W_2	31
4.7	Approximierter Funktionsverlauf nach Training mit W_1 , W_2 und W_3	31
4.8	Approximierter Funktionsverlauf nach Training mit W_1 und $W_2 \cup W_3$	32
4.9	Approximierter Funktionsverlauf nach Training mit W_1 , W_2 , W_3 und $f_1(x)$	32
4.10	Zielfunktion 3D	33
4.11	Zielfunktion 3D und Trainingsmengen	34
4.12	Zielfunktion 3D und erste Approximation	35
4.13	Zielfunktion und Approximation nach zweitem Training	35
4.14	Zielfunktion und Approximation nach drittem Training	35
4.15	Zielfunktion und Approximation nach viertem Training	35

4.16	Sollwertabweichung nach drittem Training	36
4.17	Delta der approximierten Werte nach zweitem und drittem Training . .	37
4.18	Zielfunktion und Approximation nach zweiten (online) Training	37
4.19	Zielfunktion und Approximation nach dritten (online) Training	38
4.20	Zielfunktion und Approximation nach vierten (online) Training	38
5.1	Simulation Fußball	44
5.2	Partitionierung der Variable <i>Distance</i>	44
5.3	Partitionierung der Variable <i>deltaTheta</i>	44
5.4	Partitionierung der Variable <i>outVel</i>	45
5.5	Partitionierung der Variable <i>outRot</i>	45
5.6	Sollgeschwindigkeit über Distanz	46
5.7	<i>rotVel</i> über <i>deltaTheta</i>	46
5.8	<i>rotVel</i> über <i>deltaTheta</i> bei kleinem Abstand vom Ball	46
6.1	Topologie des Jumper-Szenarios	52
6.2	Approximierte Topologie des Jumper-Szenarios	54
6.3	Zweite approximierte Topologie des Jumper-Szenarios	55
6.4	Funktionstopologie des von Hand modellierten Netzes N_{J_3}	56

Tabellenverzeichnis

4.1	Parameter einer Netzdatei	26
4.2	Netzeigenschaften die vom Benutzer über Funktionen vorgegeben werden können	26
4.3	Eigenschaften des für die 2D-Funktionsapproximation verwendeten Netzes	30
4.4	Eigenschaften des für die 3D-Funktionsapproximation verwendeten Netzes	34
6.1	Eigenschaften des Netzes N_{J_1}	53
6.2	Eigenschaften des Netzes N_{J_3}	56
B.1	Wichtige Methoden der Klasse NFidentNet (1)	71
B.2	Wichtige Methoden der Klasse NFidentNet (2)	72
B.3	Wichtige Methoden der Klasse NFidentNet (3)	73

Abkürzungen und Formelzeichen

Abkürzungen

ANFIS	Adaptive Neuro Fuzzy Inference System
COG	Center of Gravity (Defuzzifizierungsstrategie)
MOM	Mean of Maximum (Defuzzifizierungsstrategie)
NEFCON	Neuro Fuzzy Controller
NEFPROX	Neuro Fuzzy Function Approximation
TSK	Tokagi-Sugeno-Kang (Fuzzy-Regler)

Formelzeichen

A	linguistischer Term
a, b, c, d	Parameter der Fuzzy-Mengen
A_u	Aktivierungsfunktion eines Neurons
L	Lernaufgabe eines Neuro-Fuzzy-Systems
N	Neuro-Fuzzy-Netz
O_u	Ausgabefunktion eines Neurons
R	linguistische Regel eines Fuzzy-Systems
r	Belohnung (reward) beim Reinforcement Learning
S	Neuro-Fuzzy-Netz
s	Eingabemuster eines Neuro-Fuzzy-Systems
t	Ausgabemuster eines Neuro-Fuzzy-Systems
U	Menge von Neuronen eines Neuronalen Netzes oder Neuro-Fuzzy-Systems
u	Neuron
X	Eingangsmenge eines Fuzzy-Systems
δ	Deltawert bei Neuro-Fuzzy-Systemen
η	Stellgröße eines Fuzzy-Systems
Y	Ausgangsmenge eines Fuzzy-Systems
μ	Fuzzy-Menge (membership function)
ν	Fuzzy-Menge (membership function)
ξ	Messgrößen eines Fuzzy-Systems
σ	Lernrate bei Neuro-Fuzzy-Systemen
τ	Erfüllungsgrad einer Regelprämisse

Literaturverzeichnis

- [Dria93] D. Driakov, H. Hellendoorn, M. Reinfrank. *An Introduction to Fuzzy-Control*. Springer Verlag.
- [Jang97] J.-S.R. Jang, C.-T. Sun, E. Mizutani. *Neuro-Fuzzy and Soft-Computing*. Prentice-Hall, 1. Aufl.. Auflage, 1997.
- [Lee90] C.C. Lee. *FuzzyLogic in Control System: Fuzzy Logic Controller, Part I und II*. IEEE Trans. Systems, Man & Cybernetics.
- [Mich02] K.Micheks, F. Klawonn, R. Kruse, A. Nürnberger. *Fuzzy-Regelung*. Springer Verlag, 1.Aufl.
- [Nauck94] Detlef Nauck, Frank Klawonn, Rudolf Kruse. *Neuronale Netze und Fuzzy Systeme*. Vieweg-Verlag, Braunschweig/Wiesbaden, 1. Aufl.. Auflage, 1994.
- [Rose58] F. Rosenblatt. *The Perceptron: A Probabilistic Model for Information Storage and Organization in Brain*. Psychological Review, 65.
- [Volk03] E.Volk. *Lernen in Multiagenten Systemen (MAS): Reinforcement-Lernen*. Universität Stuttgart, Fakultät Informatik - IPVR. 2003.
- [www01] Nikolaus Petry. *Fuzzy Logik und neuronale Netze*. JurPC, Internet-Zeitschrift für Rechtsinformatik. <http://www.jurpc.de/aufsatz/19990187.htm>. Juni 2004.
- [www02] Institut für Informatik. *Einführung in Neuronale Netze*. Westfälische Wilhelms Universität Münster. <http://wwwmath.uni-muenster.de/SoftComputing/lehre/material/wwwnscript/bio.html>. Juni 2004.
- [www03] Institut für Informatik. *Einführung in Fuzzy Systeme*. Westfälische Wilhelms Universität Münster. <http://wwwmath.uni-muenster.de/math/inst/info/Professoren/Lippe/lehre/skripte/wwwFuzzyScript/fseincl.html> Juni 2004.
- [www04] Jan-Hendrik Schleimer. *Neuro-Fuzzy-Hybridsysteme*. <http://www.cis.hut.fi/schleime/neuro-fuzzy.pdf> August 2002.

- [www05] Detlef Nauck. *Ein Neuro-Fuzzy System zur Funktionsapproximation*. <ftp://fuzzy.cs.uni-magdeburg.de/pub/papers/niiia97.ps.gz> 1997.
- [www06] Stefan ten Hagen, Ben Kröse. *A Short Introduction to Reinforcement Learning*. In: Proc. BENELEARN-97, 7th Belgian-Dutch Conference on Machine Learning, pp. 7-12. 1997 <http://www.intellektik.informatik.tu-darmstadt.de/klausvpp/GAILS/RL/ShortIntroToRL.pdf> University of Amsterdam, Department of Mathematics, Computer Science, Physics and Astronomy.
- [www07] Jörg Steffens. *Einsatz von Hierarchischem Q-Learning zur Steuerung von Laufmaschinen*. http://joerg.steffens.com/publications/steffens_1999.pdf Rheinische Friedrich-Wilhelms-Universität Bonn, Institut für Informatik III. Januar 1999.
- [www08] Fakultät für Informatik. *Softwareagenten*. http://www.witi.cs.uni-magdeburg.de/iti_db/lehre/softwareagenten/skripte/agenten06.pdf Otto-von-Guericke-Universität Magdeburg. Stand: Juni 2004.
- [www09] Fraunhofer Institut Mikroelektronische Schaltungen und Systeme. *Fuzzy-Lexikon*. http://www.imsdd.fhg.de/englisch/produktgebiete/pdf/f_lexikon.pdf November 1998.

Erklärung

Hiermit versichere ich, diese Arbeit selbständig verfaßt und nur die angegebenen Quellen benutzt zu haben.

(Juliane Kanne)