Steffen Niendieck

Optimierung von Fuzzy–Controllern

Von Untersuchungen hybrider Neuro–Fuzzy–Systeme zum
Entwurf des universellen konnektionistischen Modells MFOS
(Münsteraner–Fuzzy–Optimierungs–System)

– 2003 –

Informatik

Optimierung von Fuzzy–Controllern

Von Untersuchungen hybrider Neuro–Fuzzy–Systeme zum Entwurf des universellen konnektionistischen Modells MFOS (Münsteraner–Fuzzy–Optimierungs–System)

Inaugural–Dissertation
zur Erlangung des Doktorgrades der
Naturwissenschaften im Fachbereich Mathematik und Informatik
der Mathematisch–Naturwissenschaftlichen Fakultät
der Westfälischen Wilhelms–Universität Münster

vorgelegt von
Steffen Niendieck
aus Rheine
– 2003 –

Dekan:	Prof. Dr. K. Hinrichs
Erster Gutachter:	Prof. Dr. W.-M. Lippe
Zweiter Gutachter:	Prof. Dr. X. Jiang

Tag der mündlichen Prüfung:	02.02.2004
Tag der Promotion:	04.02.2004

The real world is pervasively imprecise and uncertain. Precision and certainty carry a cost. The guiding principle of soft computing is: Exploit the tolerance for imprecision, uncertainty and partial truth to achieve tractability, robustness and low solution cost.

L. Zadeh

Inhaltsverzeichnis

Einleitung

L. Zadehs [Zadeh, 1965] Definition verdeutlicht die Prinzipien des *SoftComputing.* Zur Lösung eines Problems werden anstelle von exakten Verfahren wie z.B. Differentialgleichungen Methoden verwendet, die Näherungslösungen bestimmen. Dem Verlust an Genauigkeit und Sicherheit stehen geringere Rechenkosten und eine größere Fehlertoleranz gegenüber.

Die Methoden des SoftComputing sind insbesondere deshalb so attraktiv für den Anwender, weil sie im Vergleich zu herkömmlichen Methoden einfachere bzw. automatisierte Lösungsmöglichkeiten mit z.T. sehr anschaulichen Problembeschreibungen bzw. Modellen zur Verfügung stellen. Unter dem Begriff SoftComputing sind im Wesentlichen die folgenden drei Bereiche zusammengefaßt:

- Neuronale Netze
- Fuzzy–Systeme
- Genetische Algorithmen

Neuronale Netze sind parallele Systeme, die aus vielen einfachen Einheiten bestehen, welche weitgehend miteinander verbunden sind (Konnektionismus). Ursprünglich wurden künstliche Neuronen und künstliche neuronale Netze als Modelle von biologischen Neuronen und dem Gehirn entwickelt. Insbesondere die Lernfähigkeit war ein Ziel bei der Entwicklung von künstlichen neuronalen Netzen. Aufbau, Struktur und Veränderungen durch Lernvorgänge wurden dem Gehirn nachempfunden, jedoch bleiben Komplexität und Leistungsfähigkeit der künstlichen Systeme weit hinter den biologischen Systemen zurück.

Neuronale Netze kommen speziell bei Klassifikationsaufgaben und bei Funktions–Approximationen zum Einsatz. Ihr großer Vorteil ist ihre Lernfähigkeit: neuronale Netze werden durch Präsentation von Trainingsbeispielen auf die Anwendung vorbereitet. D.h. eine konkrete Modellbildung oder das Erstellen und Lösen von Gleichungen ist zur Lösung eines Problems nicht mehr erforderlich. Im Prinzip wird bei der Anwendung eines neuronalen Netzes wie folgt vorgegangen:

Zunächst wird festgelegt, welche Art von künstlichem neuronalen Netz eingesetzt werden soll, danach wird die konkrete Struktur des verwendeten Netzes vorgegeben sowie einige Parameter gewählt. Anschließend geschieht die exakte Einstellung des neuronalen Netzes automatisch durch Trainingsverfahren.

Ein Nachteil dieser Methode ist, daß keine vorhandenen Informationen bzw. Regeln über das Ein– Ausgabe–Verhalten direkt implementiert werden können. Wissenserwerb findet ausschlielich durch Training statt.

Fuzzy–Systeme sind Systeme, die Methoden der Fuzzy–Logik bzw. Fuzzy–Arithmetik nutzen. Fuzzy–Logik ist eine Erweiterung der klassischen Logik auf Wahrheitswerte im gesamten Einheitsintervall $[0, 1]$. Auf diese Weise lassen sich Sachverhalte wesentlich präziser darstellen als in der klassischen Logik, die nur die Werte `wahr` und `falsch` kennt. Analog ist Fuzzy–Arithmetik eine Erweiterung der klassischen Arithmetik auf sogenannte Fuzzy–Zahlen. Haupt–Anwendung der Fuzzy–Logik sind Fuzzy–Controller. Sie kommen u.a. bei der Systemsteuerung zum Einsatz, z.B. um einen Brennofen zu regeln. Auch in Waschmaschinen und Videokameras ("Anti–Verwackelung") finden sich Fuzzy–Controller.

Ein Vorteil von Fuzzy–Controllern ist, daß die Steuerung mit einfachen WENN–DANN–Regeln realisiert wird, die von einem System–Experten formuliert werden. Auf diese Weise kann das Wissen und die Erfahrung des Experten genutzt werden, ohne die sonst üblichen komplizierten Differentialgleichungen aufstellen und lösen zu müssen. Ein Nachteil von Fuzzy–Controllern ist, daß sie nicht lernfähig sind. Vor dem Einsatz muß also jede Regel bekannt sein, ebenso wie die weiteren Bestandteile.

Genetische Algorithmen sind Such– bzw. Optimierungsalgorithmen, die der biologischen Evolution nachempfunden sind. Zur Bewältigung eines gegebenen Problems gibt es eine Menge (Population) von Lösungen, wobei jede durch eine spezielle Fitneßfunktion bewertet wird. Die Lösungen werden zumeist als Bitstrings codiert, um dann durch einzelne Änderungen im Code neue Lösungen zu generieren. Anschließend werden die schlechtesten Lösungen entfernt. Dieser Prozeß wird iterativ fortgesetzt, bis eine Lösung mit maximaler Fitneß erzeugt wurde. Schwierigkeiten bereiten hier die geeignete Wahl der Codierung und der Fitneß–Funktion sowie Konvergenz–Probleme.

Die einzelnen Gebiete des SoftComputing sind unabhängig voneinander entwickelt worden und basieren auf ganz unterschiedlichen – zum Teil gegensätzlichen – Ansätzen. Trotzdem oder gerade deshalb wuchsen sie in den letzten Jahren vermehrt zusammen. Durch die Kombination der unterschiedlichen Methoden entstehen Synergie–Effekte, die die Leistungsfähigkeit der Systeme deutlich erhöhen und völlig neue Möglichkeiten bieten. Die durch Kombination von Techniken und Algorithmen aus den Bereichen neuronale Netze, Fuzzy–Logik und genetische Algorithmen entstehenden Systeme werden als hybride Systeme bezeichnet.

Wie schon dieser kurze Überblick zeigt, heben sich insbesondere die Vor– und Nachteile von künstlichen neuronalen Netzen und Fuzzy–Controllern gegenseitig auf. Erstere sind lernfähig, erlauben jedoch keine direkte Einbindung von Regeln, bei letzteren ist es gerade umgekehrt. Es ist also naheliegend, insbesondere diese Ansätze zu kombinieren, um bessere Fähigkeiten zu erhalten. So sind bereits einige Modelle entwickelt worden, die neuronale Netze mit Fuzzy–Logik kombinieren (Fuzzy–Neuronale–Netze) bzw. Fuzzy–Controller in ein neuronales Netz transformieren (Neuro–Fuzzy–Systeme). Letztere sollen auch Schwerpunkt dieser Arbeit sein.

Diese Arbeit ist wie folgt aufgebaut:

Teil I (Kapitel 1 – 3) beschäftigt sich mit den theoretischen Grundlagen über das Lernen, neuronale Netze und Fuzzy–Logik, die in Teil II und Teil III vorausgesetzt werden.

In Kapitel 1 werden als Motivation für die vorliegende Arbeit einige Konzepte über Aufbau und Funktion des Gehirns und dessen Lernvorgänge kurz vorgestellt. Dabei wird nicht der Anspruch einer vollständigen Darstellung aller komplexen Abläufe im Gehirn erhoben – diese sind noch längst nicht vollständig erforscht. Der Fokus liegt in der Beschreibung einiger Strukturen und Abläufe, die in künstlichen Systemen übernommen wurden. Hierbei geht es insbesondere um die Fragen, wie sich das Gehirn durch Lernvorgänge verändert und auf welche Weise gespeichertes Wissen, z.B. in Form von Regeln, beschrieben werden kann.

Inhalt von Kapitel 2 ist eine systematische Einführung von künstlichen neuronalen Netzen. Da das Vorbild künstlicher Neuronen biologische Neuronen sind, werden zunächst die wichtigsten Bestandteile und prinzipiellen Funktionsweisen von biologischen Neuronen vorgestellt. Anschließend werden künstliche Neuronen definiert, die das Verhalten von biologischen Neuronen simulieren. Künstliche neuronale Netze bestehen aus einer Anzahl künstlicher Neuronen und Verbindungen zwischen diesen Neuronen. Mit der Hebbschen Lernregel wird der natürliche Lernvorgang simuliert.

Der geschichtliche Überblick zeigt die Entwicklung der künstlichen neuronalen Netze von den ersten Modellen bis zum wichtigsten Verfahren der Praxis, dem Backpropagation–Verfahren. Dieses und der sogenannte lineare Assoziierer werden in separaten Abschnitten ausführlich beschrieben. Das Kapitel endet mit der Feststellung, daß künstliche neuronale Netze universelle Approximatoren sind.

Inhalt von Kapitel 3 ist eine systematische Einführung von Fuzzy–Systemen. Nach Beschreibung der historischen Entstehung werden Fuzzy–Mengen durch Erweiterung des Bildbereichs der Indikatorfunktion klassischer Mengen auf das Intervall $[0, 1]$ definiert und die grundlegenden Operationen für Fuzzy–Mengen eingeführt. Analog werden Fuzzy–Relationen als Erweiterung klassischer Relationen definiert.

Die Fuzzy–Arithmetik ermöglicht es, mit Hilfe des Extensionsprinzips mathematische Funktionen für sogenannte Fuzzy–Zahlen zu berechnen.

Die Fuzzy–Aussagenlogik wird als Erweiterung der klassischen Aussagenlogik auf Wahrheitswerte zwischen 0 und 1 eingeführt. Unscharfes Schließen ist eine Methode, die in Fuzzy–Controllern ihre Anwendung findet. Mit dem Fuzzy–Controller nach Mamdani und dem Sugeno–Controller werden die wichtigsten Versionen von Fuzzy–Controllern ausführlich vorgestellt. Beide Versionen sind genau wie neuronale Netze universelle Approximatoren. Das Kapitel endet mit der Vorstellung eines bekannten Problems, welches mit Fuzzy–Controllern gelöst werden kann, dem Stabbalance–Problem.

Da genetische Algorithmen bei den weiteren Untersuchungen keine Rolle spielen, wird auf eine ausführliche Vorstellung verzichtet. Hier sei auf die Fachliteratur verwiesen, z.B. [Schöneburg, 1994] oder [Nissen, 1997].

Teil II (Kapitel 4 – 7) widmet sich speziell den auf neuronalen Netzen basierenden Optimierungsmethoden für Fuzzy–Controller nach Mamdani.

In den Kapiteln 4 und 5 werden mit dem Verfahren von Lin und Lee sowie dem NEFCON–Modell konkrete Realisierungen von Neuro–Fuzzy–Systemen vorgestellt. Beide Methoden ermöglichen unter bestimmten Voraussetzungen die Übertragung eines Fuzzy–Controllers nach Mamdani auf ein geeignetes neuronales Netz und die anschließende Optimierung der wichtigsten Bestandteile dieses Fuzzy–Controllers, der Fuzzy–Mengen und Regeln. Jedoch unterscheiden sich die vorgestellten Methoden in der Struktur des verwendeten Netzes und den verwendeten speziellen Lernverfahren.

Das Verfahren von Lin und Lee bewirkt, daß jede vorhandene Regel die optimale Schlußfolgerung (Konklusion) erhält. Zusätzlich werden die Fuzzy–Mengen angepaßt und bei Bedarf neue Fuzzy–Mengen erzeugt, jedoch nur sogenannte Eingabe–Partitions–Mengen. Das Erzeugen neuer Regeln ist mit dieser Methode nicht durchführbar.

Das NEFCON–Modell führt eine Anpassung der Fuzzy–Mengen durch und erzeugt neue Regeln. Das Erzeugen neuer Fuzzy–Mengen ist mit dieser Methode nicht durchführbar.

Sowohl das Verfahren von Lin und Lee als auch das NEFCON–Modell ermöglichen nur mit einigen Einschränkungen die Optimierung von Fuzzy–Controllern. Zum einen müssen als Voraussetzung für einen Einsatz dieser Verfahren bestimmte Voraussetzungen an den Fuzzy–Controller erfüllt werden. Zum anderen werden jeweils nur bestimmte Bestandteile des Fuzzy–Controllers angepaßt.

Dies führt dazu, daß in manchen Fällen die verwendeten Lernverfahren scheitern. Falls z.B. beim Konfigurieren des Verfahrens von Lin und Lee eine benötigte Regel nicht berücksichtigt wird, ist eine erfolgreiche Optimierung mit diesem System nicht möglich.

Gleiches gilt, falls das NEFCON–Modell mit zu wenigen Fuzzy–Mengen konfiguriert wird.

Diese Einschränkungen bekannter Methoden führten zum Wunsch, ein alternatives Verfahren zur Optimierung von Fuzzy–Controllern zu entwickeln. Zum einen sollten die Voraussetzungen für den Einsatz dieses Verfahrens möglichst gering sein, zum anderen sollten möglichst alle Bestandteile des Fuzzy–Controllers nach Wunsch des Anwenders angepaßt werden können. Das Resultat dieser Entwicklung wird in Kapitel 6 ausführlich beschrieben.

In Kapitel 6 wird das Münsteraner–Fuzzy–Optimierungs–System MFOS vorgestellt. Mit diesem Modell ist es möglich, einen potentiell beliebigen Fuzzy–Controller nach Mamdani in ein funktional äquivalentes neuronales Netz zu transformieren. Die Komponenten des Fuzzy–Controllers lassen sich danach mit unterschiedlichen Lernverfahren individuell optimieren. Anschließend ist eine Rücktransformation in einen separaten, optimierten Fuzzy–Controller möglich.

Bei der Entwicklung des Modelles und der Lernverfahren wurde darauf geachtet, die von anderen Systemen dieser Art bekannten Einschränkungen für den verwendeten Fuzzy–Controller und bei den Optimierungen zu vermeiden. Daher gibt es für den Einsatz des MFOS keine besonderen Voraussetzungen an den Fuzzy–Controller. Die speziell entwickelten Lernverfahren ermöglichen ohne besondere Einschränkungen die Anpassung der Bestandteile des Fuzzy–Controllers nach den Vorgaben des Anwenders. Mit dem MFOS–System lassen sich sowohl Regeln als auch Fuzzy–Mengen erzeugen, anpassen und auch löschen.

In Kapitel 7 werden die vorgestellten Systeme bezüglich ihrer Möglichkeiten, einen vorgegebenen Fuzzy–Controller zu optimieren, verglichen und bewertet. Es wird untersucht, welche Voraussetzungen an den verwendeten Fuzzy–Controller jeweils gemacht werden und welche Optimierungsmöglichkeiten die einzelnen Methoden zur Verfügung stellen. Dabei stellt sich heraus, daß lediglich das MFOS–System in der Lage ist, einen vorgegebenen Fuzzy–Controller in nahezu allen Fällen zu optimieren. Bei den anderen Systemen lassen sich Beispiele finden, in denen die vorhandenen Lernverfahren aus prinzipiellen Gründen versagen.

Bestimmte Optimierungsziele werden von den verschiedenen Systemen auf unterschiedliche Weise erreicht. Daher wird untersucht, welche dieser Verfahren für die jeweilige Aufgabe besser geeignet sind. Schließlich wird der Frage nachgegangen, ob sich bestimmte Lernverfahren eines Systems auf ein anderes System übertragen lassen, um dessen Leistungsfähigkeit zu erhöhen und eventuelle Einschränkungen für den Einsatz zu beseitigen.

Teil III (Kapitel 8 – 10) behandelt die auf neuronalen Netzen basierenden Optimierungsmethoden für Sugeno–Controller.

In Kapitel 8 wird stellvertretend für in der Literatur vorgestellte Optimierungs–Systeme für Sugeno–Controller das ANFIS–System vorgestellt. Dieses ermöglicht die Übertragung eines Sugeno–Controllers auf ein funktional äquivalentes neuronales Netz. Mit den Lernverfahren des ANFIS–Systems lassen sich vorhandene Fuzzy–Mengen und Regeln anpassen. Das Erzeugen zusätzlicher Fuzzy–Mengen und Regeln ist nicht möglich. Daher ist in manchen Fällen eine erfolgreiche Optimierung nicht möglich.

In Kapitel 9 wird eine spezielle Version des MFOS–Systems für Sugeno–Controller entwickelt, das MFOS–S–System. Das MFOS–S–System bietet grundsätzlich die gleichen Möglichkeiten wie das MFOS–M–System, jedoch speziell für Sugeno–Controller. D.h. mit diesem System ist es möglich, einen potentiell beliebigen Sugeno–Controller in ein funktional äquivalentes neuronales Netz zu transformieren und die einzelnen Bestandteile des Controllers mit unterschiedlichen Lernverfahren individuell zu optimieren. Da Sugeno–Controller anders aufgebaut sind als Fuzzy–Controller nach Mamdani, mußten sowohl der Aufbau des verwendeten neuronalen Netzes als auch die einzelnen Lernverfahren zum Teil vollständig neu entworfen werden.

In Kapitel 10 werden die in diesem Teil vorgestellten Systeme analog zu Kapitel 7 verglichen und bewertet. Welche Voraussetzungen an den Sugeno–Controller werden gemacht, welche prinzipiellen Optimierungsmöglichkeiten stehen zur Verfügung? Worin unterscheiden sich Verfahren, welche die gleichen Optimierungsziele haben? Lassen sich bestimmte Verfahren eines Systems auf das andere System übertragen?

Teil I

Theoretische Grundlagen

Kapitel 1

Über das Lernen

Bevor in den Kapiteln 2 und 3 eine systematische Einführung zum Thema biologische und künstliche Neuronen, künstliche neuronale Netze sowie Fuzzy–Syteme erfolgt, sollen in diesem Kapitel als Motivation einige Konzepte über Aufbau und Funktion des Gehirns und dessen Lernvorgänge kurz vorgestellt werden. Dabei wird nicht der Anspruch einer vollständigen Darstellung aller komplexen Abläufe im Gehirn erhoben – diese sind noch längst nicht vollständig erforscht. Der Fokus liegt in der Beschreibung einiger Strukturen und Abläufe, die in künstlichen Systemen übernommen wurden.

Nach dem heutigen Stand der Hirnforschung ist bekannt, daß jeder Lernvorgang das Gehirn verändert. Im Gehirn befinden sich ca. 100 Milliarden Nervenzellen (**Neuronen**), die vielfach miteinander verbunden sind. Nach der Geburt sind diese **Verbindungen** zunächst wenig strukturiert, Impulse eines Neurons werden in alle Richtungen an andere Neuronen weitergeleitet. Durch **Lernprozesse** werden bestimmte Verbindungen **gestärkt**, während andere Verbindungen **verkümmern**. So gewinnen die Verbindungen mit der Zeit an **Struktur**. D.h. es entsteht ein Netz aus Nervenzellen und geordneten Verbindungen. Allerdings läßt sich diese Entwicklung nicht beliebig fortsetzen. Bis zum Jugendalter ist eine ausgeprägte Verbindungsstruktur entstanden, die danach im Wesentlichen bestehen bleibt.

Selbstverständlich ist es auch im Erwachsenenalter noch möglich, neues Wissen zu erwerben. Allerdings wird dieses Wissen in die bestehende Struktur des Gehirns eingefügt, ohne sie grundlegend zu verändern. Es werden fast ausschließlich die bereits bestehenden Verbindungen verstärkt oder geschwächt. Neue Verbindungen werden nur noch selten erzeugt. Dies hat z.B. zur Folge, daß das Erlernen einer neuen Sprache für einen Erwachsenen wesentlich einfacher ist, wenn er schon eine verwandte Sprache beherrscht. Eine Sprache zu lernen, die sich strukturell von bereits gelernten Sprachen unterscheidet, fällt wesentlich schwerer.

In Tierversuchen erfolgreich erprobt ist das Prinzip **Lernen durch Wiederholung und Strafe:** springt die Versuchs–Rennmaus nicht nach einem Signalton hoch, erfolgt ein lästiger Stromschlag. Bereits nach wenigen Wiederholungen (im besten Fall nach zwei) haben alle Rennmäuse den Zusammenhang erkannt und springen unmittelbar

nach dem Signalton hoch. Die Vermeidung der Strafe ist jedoch nicht die einzige Konsequenz aus dem Lernerfolg: für jeden gelungenen Sprung gibt es eine körpereigene **Belohnung** durch Ausschüttung von Dopamin, welches ein Glückbefühl bewirkt. H. Scheich [Der Spiegel, 2002], Direktor am Leibniz–Institut für Neurobiologie, ist überzeugt, daß auch Menschen auf Lernerfolge mit Begeisterung reagieren: "Ein Kind lernt dann am besten, wenn es Aufgaben selbständig löst. Das Lustgefühl, das damit einhergeht, ist nachhaltiger als jede Belohnung von außen – anders, als viele Erziehungswissenschaftler meinen ".

Komplexe Gehirne lernen auf ähnliche Weise, gleichgültig, ob es sich um Gehirne von Vögeln, Säugetieren oder Fischen handelt. Die Grundlegenden neuronalen Mechanismen sind universell – von der Meeresschnecke Aplysia (s. [Zell, 1996]) bis zum Menschen. Um von der Vielzahl an Eindrücken, die das Gehirn in jeder Sekunde bekommt, nicht überfordert bzw. blockiert zu werden, ist es notwendig, wichtige Informationen herauszufiltern und **Kategorien** zu bilden. So es z.B. offensichtlich sinnvoll, Autos, Busse und Motorräder als "Fahrzeuge" zu begreifen und unter diesem Begriff zusammenzufassen.

Neben der Erforschung von Aufbau und Funktion des Gehirns und dem Ablauf von Lernprozessen ist die Modellierung lernfähiger künstlicher Systeme von Interesse. Einen besonderen Stand haben hier aufgrund ihrer Ähnlichkeit zum Gehirn und dessen Lernvorgängen die *künstlichen neuronalen Netze*. Ursprünglich wurden künstliche Neuronen als Simulation biologischer Neuronen entwickelt. Künstliche neuronale Netze sind strukturell ähnlich aufgebaut wie ein Gehirn. Ebenso sind die ersten entwickelten Lernverfahren für diese künstlichen Systeme an natürliche Lernvorgänge angelehnt.

Heute erreichen künstliche neuronale Netze bei weitem nicht die Komplexität eines Gehirns. Statt 100 Milliarden Neuronen werden meistens einige tausend Neuronen verwendet. Die heute üblichen Lernverfahren sind Algorithmen, die oft nicht mehr natürlich sondern mathematisch motiviert sind. Dennoch finden sich die oben vorgestellten **Konzepte** vom Lernen und Aufbau des Gehirns in vielen künstlichen neuronalen Netzen wieder:

Ein künstliches neuronales Netz besteht aus einer Anzahl von künstlichen Neuronen und Verbindungen zwischen diesen Neuronen. Durch die meisten verwendeten Lernverfahren werden bestehende Verbindungen gestärkt oder geschwächt. Die Stärke einer Verbindung wird üblicherweise durch eine reelle Zahl (*Gewicht)* angegeben.

Oftmals wird die Struktur eines Netzes vom Anwender vorgegeben, d.h. die Anzahl der Neuronen und die Verbindungen zwischen den einzelnen Neuronen werden vor dem Lernvorgang festgelegt. Beim Lernen werden nur noch die vorhandenen Verbindungen variiert. Eine übliche Methode des Lernens besteht darin, die berechneten Ergebnisse eines neuronalen Netzes zu bewerten und dessen Verbindungen dahingehend zu modifizieren, daß "gute" Ergebnisse gestärkt und "schlechte" weitestgehend vermieden

werden (vgl. strafen und belohnen). Der Lernvorgang eines künstlichen neuronalen Netzes geschieht in den meisten Fällen durch wiederholte Präsentation von sogenannten Trainingsbeispielen. Diese bestehen aus Eingabedaten und eventuell zugehörigen Ausgabedaten. Ein typisches Ziel des Lernvorganges besteht darin, Gemeinsamkeiten unter den Trainingsbeispielen zu erkennen und diese in Kategorien (Klassen) einzuteilen.

Wie diese kurze Vorstellung künstlicher neuronaler Netze zeigt, gibt es viele Gemeinsamkeiten zwischen einem Gehirn und seinen Lernprozessen mit künstlichen neuronalen Netzen und deren Lernverfahren. Eine systematische Einführung von künstlichen neuronalen Netzen ist Inhalt von Kapitel 2.

Eine weitere interessante Fragestellung ist, auf welche Weise das in natürlichen oder künstlichen Systemen gespeicherte Wissen beschrieben werden kann. Läßt sich das in einem Gehirn bzw. künstlichen neuronalen Netz über viele Neuronen verteilt gespeicherte Wissen extrahieren? Lassen sich komplexe Zusammenhänge durch anschauliche Regeln beschreiben? Wie lassen sich diese Regeln mathematisch modellieren?

Ein Experte zur Steuerung einer industriellen Anlage ist aufgrund seines Fachwissens und seiner Erfahrung in der Lage, Meßwerte zu interpretieren und daraus die richtigen Folgerungen zu schließen. Eine nachvollziehbare Beschreibung, wie er bei gegebenen Meßwerten auf eine konkrete Folgerung kommt, ist hingegen oftmals schwierig. **Fuzzy–Systeme** bieten die Möglichkeit, das Wissen eines Experten in Form von anschaulichen Regeln darzustellen und Meßwerte anschaulich zu beschreiben.

Beispiel 1.1
Eine Fuzzy–Regel zur Steuerung der Temperatur läßt sich wie folgt darstellen:

```
WENN Temperatur = niedrig DANN Heizleistung = hoch
```

Jedoch ist es nicht immer einfach, konkrete Regeln dieser Art zu erstellen. Auch wenn ein Experte detailliertes Wissen über die komplexen Zusammenhänge eines zu steuernden Systems besitzt, ist die Formulierung konkreter Regeln nicht einfach. Mißverständnisse, vergessene Sonderfälle etc. führen schnell dazu, daß eine Regel fehlerhaft formuliert oder vergessen wird.

Eine Methode zur Unterstützung der Formulierung bzw. automatischen Erstellung von Fuzzy–Regeln wäre daher sinnvoll. Die Kombination von künstlichen neuronalen Netzen mit sogenannten **Fuzzy–Controllern** erweist sich hierbei als besonders hilfreich. Die Lernverfahren eines neuronalen Netzes ermöglichen die Übertragung des in Form von Trainingsbeispielen vorliegenden Wissens auf ein spezielles künstliches neuronales Netz. Aufgrund der geeignet gewählten Struktur des verwendeten Netzes ist anschließend die Extraktion des gelernten Wissens in Form von Fuzzy–Regeln möglich.

Eine systematische Einführung von Fuzzy–Systemen ist Inhalt von Kapitel 3. In den weiteren Kapiteln dieser Arbeit (Kapitel 4 – Kapitel 10) soll untersucht werden, wie mit Hilfe von neuronalen Netzen Fuzzy–Regeln bzw. Fuzzy–Controller erstellt und **optimiert** werden können.

Kapitel 2

Neuronale Netze

Der mit Abstand leistungsfähigste "Computer" der Welt ist das menschliche Gehirn. Dieses besteht aus etwa einer Billion Zellen, von denen ca. 100 Milliarden Nervenzellen (Neuronen) sind. Jedes einzelne dieser Neuronen ist mit hunderten oder tausenden anderer Neuronen verbunden. Das funktionale Verhalten eines einzelnen Neurons ist sehr simpel: entweder es feuert einen Impuls ab, oder es feuert keinen Impuls ab. Seine herausragende Leistungsfähigkeit erhält das Gehirn erst durch die extrem hohe Anzahl dieser Zellen und die große Parallelität der Verbindungen. Der Wunsch, ein ähnlich leistungsfähiges und insbesondere lernfähiges künstliches System zu erschaffen, führte zur Entwicklung von künstlichen Neuronen und schließlich künstlichen neuronalen Netzen. Sie und ihr biologisches Vorbild sind Gegenstand dieses Kapitels.

2.1 Von biologischen Neuronen ...

Ein *biologisches Neuron* (Abb. 2.1) besteht im wesentlichen aus einem *Zellkern* sowie dem *Axon* als Ausgang und den *Dendriten* als Eingängen für die Ausgabe–Impulse anderer Neuronen. Um die Zelle herum befindet sich eine Membran, die kaum durchlässig ist für gelöste Ladungsträger *(Ionen)*. In– und außerhalb der Zelle befinden sich positiv geladene Natrium–Ionen, wobei die Konzentration im Inneren geringer ist als außen. Dadurch entsteht ein Spannungspotential an der Membran von ca. $-70mV$. Über die Dendriten empfängt das Neuron Ausgabe–Impulse anderer Neuronen, die sowohl erregend als auch hemmend sein können. Liegt die insgesamt resultierende Erregung über einer Schwelle, so wird die Membran durchlässig. Dadurch strömen plötzlich Natrium–Ionen nach innen, und gleichzeitig ebenfalls vorhandene Kalium–Ionen nach außen. Auf diese Weise baut sich ein Aktions–Potential von ca. $+40mV$ auf, das als Ausgabe–Impuls über das Axon abgegeben wird (das Neuron ist *aktiv*). Gleichzeitig wird mit Hilfe einer Ionen–Pumpe das ursprüngliche Konzentrations–Verhältnis wieder hergestellt.

Die Übertragung eines Impulses vom Axon eines Neurons auf einen Dendriten eines anderen Neurons erfolgt über die *Synapse* oder den *synaptischen Spalt*. Das Axon schüttet Neurotransmitter aus, die von Rezeptoren im Dendriten aufgenommen werden. Je nach

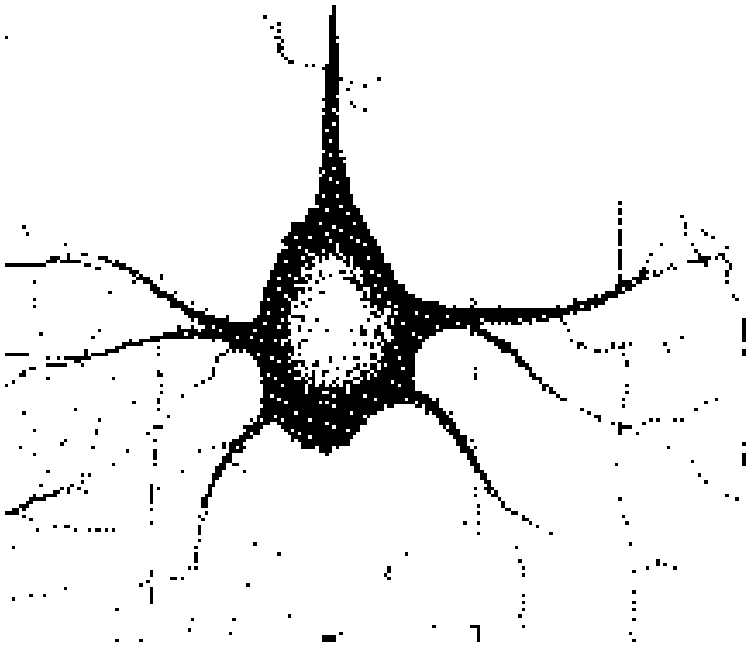

Abbildung 2.1: Ein biologisches Neuron

Art der Rezeptoren können die Neurotransmitter erregend oder hemmend wirken. Die erregenden und hemmenden Eingaben werden von dem Neuron verarbeitet. Erst wenn die resultierende gesamte Erregung über einer Schwelle liegt, wird das Neuron aktiv.

Sobald zwei miteinander verbundene Neuronen gleichzeitig aktiv sind, wird ihre Verbindung gestärkt. Dies geschieht entweder durch vermehrte Ausschüttung von Neurotransmittern, oder durch Bildung von zusätzlichen bzw. verbesserten Rezeptoren. Diese Verstärkung hat zur Folge, daß in Zukunft der Einfluß des sendenden Neurons auf die Aktivierung des anderen stärker ist. D.h. sein Anteil bei der Bildung der gesamten Erregung ist größer. Dieser Vorgang wird als *Hebbsche Lernregel* bezeichnet und ist tatsächlich ein wesentlicher Bestandteil eines jeden Lernvorganges im Gehirn.

Diese prinzipielle Funktionsweise von biologischen Neuronen dient als Vorbild für die Modellierung von künstlichen Neuronen und künstlichen neuronalen Netzen. Für das Verständnis der im Folgenden vorgestellten künstlichen Modelle sind weitergehende Kenntnisse der biochemischen Prozesse nicht erforderlich. Daher sei bezüglich deren Beschreibung auf die Fachliteratur (z.B. [Zell, 1996]) verwiesen.

2.2 . . . zu künstlichen neuronalen Netzen

Bevor wir zur Definition eine künstlichen Neurons kommen, betrachten wir zunächst einmal ein schematisches Modell eines biologischen Neurons (Abb. 2.2):

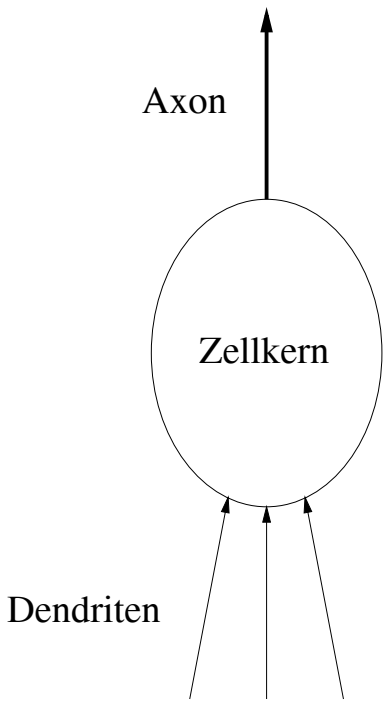

Abbildung 2.2: Schematisches Modell eines biologischen Neurons

Um das Verhalten eines biologischen Neurons zu simulieren, müssen Eingaben über die Dendriten empfangen und anschließend verarbeitet werden, so daß analog zum biologischen Vorbild bei genügend starker Erregung ein Impuls über das Axon abgegeben wird.

Dies führt zur folgenden Definition:

Definition 2.1 *Ein künstliches Neuron besteht aus:*

- *einem Gewichtsvektor* $\vec{w} \in \mathbb{R}^n$
- *einer Aktivitätsfunktion* $f_a : \mathbb{R}^n \times \mathbb{R}^n \longrightarrow \mathbb{R}$
- *einer Ausgabefunktion* $f_o : \mathbb{R} \longrightarrow \mathbb{R}$

Die einzelnen Werte von $\vec{w}$ *heißen Gewichte.*

Seine Eingaben $\vec{x} \in \mathbb{R}^n$ *bekommt das Neuron durch berechnete Ausgaben anderer Neuronen bzw. von außen.*

Die Ausgabe o *ergibt sich durch* $o = f_o\big(f_a(\vec{x}, \vec{w})\big)$.

Als Aktivitätsfunktion wird oft die gewichtete Summe der Eingaben verwendet, als Ausgabefunktion eine *Schwellenwertfunktion*:

Definition 2.2 *Sei $\theta \in \mathbb{R}$. Eine Funktion f_o mit:*

$$f_o(x) = \begin{cases} 1, & falls\ x \geq \theta \\ 0, & falls\ x < \theta \end{cases}$$

heißt Schwellenwertfunktion, θ ist der Schwellenwert.

Mit einer Schwellenwertfunktion als Ausgabefunktion wird das Verhalten eines biologischen Neurons erfolgreich simuliert:

Sämtliche Eingaben werden mit der Aktivitätsfunktion verarbeitet. Dabei wird die Stärke einer Verbindung durch das entsprechende Gewicht repräsentiert. Hemmende Verbindungen werden durch negative Gewichts–Werte realisiert. Erst wenn die insgesamt resultierende Erregung oder *Aktivierung* (der Wert von f_a) über der Schwelle θ liegt, gibt das Neuron einen Impuls ab (die Ausgabe ist 1), andernfalls gibt das Neuron keinen Impuls ab (die Ausgabe ist 0).

Definition 2.3 *Sei N ein künstliches Neuron. Der Funktionswert der Aktivitätsfunktion $f_a(\vec{x}, \vec{w})$ heißt Aktivierung von N.*

Häufig wird statt einer Schwellenwertfunktion eine *sigmoide Funktion* als Ausgabefunktion verwendet:

Definition 2.4 *Seien $a, b \in \mathbb{R}$ mit $a < b$. Eine Funktion $f : \mathbb{R} \longrightarrow [a, b]$ heißt sigmoid, wenn sie monoton wachsend und differenzierbar ist und folgendes gilt:*

$$\lim_{x \to -\infty} f(x) = a, \lim_{x \to +\infty} f(x) = b$$

Eine typische sigmoide Funktion ist:

Beispiel 2.1

$$f(x) = \frac{1}{1 + e^{-x}}$$

Durch die Verwendung einer sigmoiden Funktion wird der Ausgabebereich eines Neurons auf ein reelles Intervall $[a, b]$ erweitert. Ein anderer Vorteil ist die Differenzierbarkeit dieser Funktion, da sie Voraussetzung für die Anwendung eines Gradientenabstiegsverfahrens ist (s. Abschnitt 2.6).

Um dem Vorbild Gehirn etwas näher zu kommen, betrachten wir nun das Zusammenwirken mehrerer künstlicher Neuronen in einem Verbund:

Definition 2.5 *Ein künstliches neuronales Netz ist ein Paar $(\mathcal{N}, \mathcal{V})$ mit $\mathcal{N}$ einer endlichen Menge von künstlichen Neuronen und $\mathcal{V}$ einer Menge von Verbindungen, so daß ein gerichteter Graph entsteht. Dabei gilt:*

- *jeder Knoten ist ein künstliches Neuron*
- *jede Kante ist eine Verbindung zwischen zwei Neuronen*
- *jedes Neuron kann beliebig viele verschiedene Eingaben über die ankommenden Verbindungen empfangen*
- *jedes Neuron gibt genau einen Ausgabewert über die weggehenden Verbindungen an beliebig viele andere Neuronen weiter*
- *es kann zusätzlich Verbindungen von und nach außen geben, für externe Eingaben und Ausgaben*

In vielen Fällen ergeben die Verbindungen eine Aufteilung der Neuronen in einzelne Schichten. Dabei werden jeweils die Ausgaben der Neuronen einer Schicht an die Neuronen der darüberliegenden Schicht weitergegeben (*feedforward–Netz*). Die erste Schicht ist dann die Eingabeschicht für externe Eingaben, die letzte Schicht ist die Ausgabeschicht, deren Ausgabe–Werte als einzige extern zur Verfügung stehen.

Definition 2.6
Ein neuronales Netz heißt total verbunden, wenn jedes Neuron eine Verbindung zu jedem anderen Neuron besitzt.

Ein in Schichten aufgeteiltes neuronales Netz heißt schichtweise total verbunden, wenn jedes Neuron einer Schicht eine Verbindung zu jedem Neuron der darüberliegenden Schicht besitzt (s. Abb. 2.3).

Bei den Architekturen neuronaler Netze gibt es fast beliebige Variationen, etwa Verbindungen über eine Schicht hinweg (*shortcuts*), freie Anordnung der Neuronen ohne Schichtenbildung, etc.

Wird die Ausgabe eines Neurons direkt oder indirekt (über einen Zyklus mit anderen Neuronen) an dieses Neuron weitergeleitet, ergibt sich ein *rekursives Netz*. Abweichend

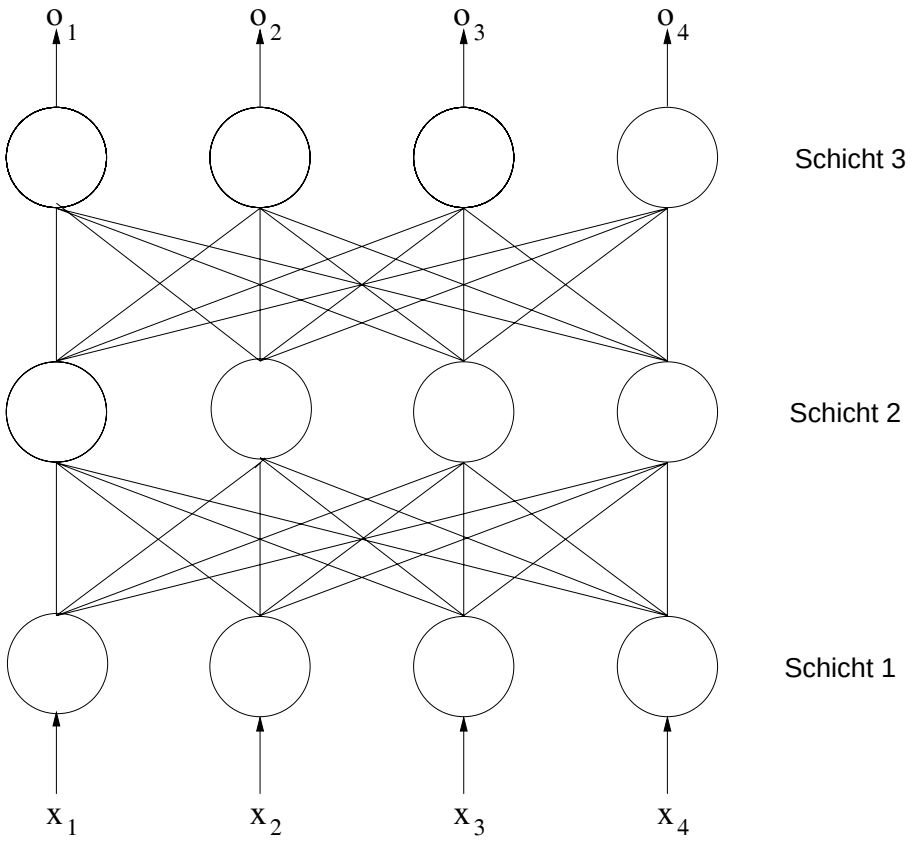

Abbildung 2.3: Ein dreischichtiges, schichtweise total verbundenes neuronales Netz

zur Definition werden die Gewichte oft zur übersichtlicheren Darstellung den Verbindungen zugeordnet. Numeriert man alle Neuronen durch, so läßt sich mit w_{ij} das Gewicht der Verbindung von Neuron Nr. i zu Neuron Nr. j bezeichnen.

In einem Netz mit mehreren Schichten erfolgt die Berechnung der Ausgabe des gesamten Netzes üblicherweise wie folgt:

1. die Neuronen aus Schicht 1 bekommen von außen ihre Eingabe
2. jedes Neuron aus Schicht 1 berechnet selbständig seinen Ausgabewert
3. dieser Ausgabewert wird jeweils über die Verbindungen an alle (oder auch nur einige) Neuronen aus Schicht 2 weitergeleitet
4. die Neuronen aus Schicht 2 berechnen jeweils ihren Ausgabewert, der auf die gleiche Weise an Schicht 3 weitergegeben wird, usw.
5. die Ausgaben der letzten Schicht ergeben die Netzausgabe.

Die Ausgaben der Neuronen aus inneren Schichten sind zu keinem Zeitpunkt von außen erkennbar. Deshalb werden diese auch als *verborgene Schichten* bezeichnet.

2.3 Lernmöglichkeiten für neuronale Netze

Nach den grundlegenden Definitionen wenden wir uns nun dem Begriff "Lernen" zu, für den neuronale Netze bekannt sind. Beim biologischen Vorbild Gehirn erfolgt das Lernen gemäß der Hebbschen Lernregel durch Verstärkung der Verbindung von gleichzeitig aktiven Neuronen. Da bei künstlichen neuronalen Netzen die Stärke einer Verbindung durch das zugehörige Gewicht gegeben ist, läßt sich dieser Vorgang durch eine Änderung des Gewichts–Wertes durchführen. Die Hebbsche Lernregel ist also ohne Schwierigkeiten auf künstliche neuronale Netze übertragbar:

Definition 2.7 *Hebbsche Lernregel: Sei ein künstliches neuronales Netz gegeben. Bekommt Neuron Nr. j eine Eingabe von Neuron Nr. i, und sind beide Neuronen gleichzeitig stark aktiv, so wird das Gewicht w_{ij} der zugehörigen Verbindung gemäß folgender Formel erhöht:*

$$\Delta w_{ij} = \eta \cdot o_i \cdot a_j$$

mit Δw_{ij} der Gewichtsänderung, $\eta \in \mathbb{R}$ einer Lernrate zur Steuerung der Größe der Änderung (oft aus $[0, 1]$), o_i die Ausgabe von Neuron Nr. i und a_j der Aktivierung von Neuron Nr. j.

Die Lernrate η bewirkt, daß die Änderung des Gewichts w_{ij} nur zu einem Bruchteil des Wertes von $o_i \cdot a_j$ erfolgt. Auf diese Weise werden zu große Änderungen vermieden. Bei sehr hohen Werten eines Gewichtes ist der Einfluß der zugehörigen Verbindung auf die Aktivierung zu stark, so daß andere Verbindungen bedeutungslos werden. Dies ist auch ein Nachteil der Hebbschen Lernregel, da sie nur eine Errhöhung der Gewichts–Werte vorsieht und keine Verringerung, so daß nach einiger Zeit die Gewichte zu groß werden.

Die meisten in der Praxis verwendeten Lernverfahren für künstliche neuronale Netze sind Variationen der Hebbschen Lernregel. Bei praktisch relevanten Lernregeln wie dem *Backpropagation–Verfahren* (s. Abschnitt 2.6) ist im Gegensatz zur Hebbschen Lernregel in der ursprünglichen Definition auch eine Verringerung der Gewichte möglich. Dadurch ist eine korrekte Anpassung gut zu erreichen.

Es gibt auch alternative Lernverfahren, die Verbindungen oder ganze Neuronen einfügen bzw. löschen. Was aber ist mit dem Begriff "Lernen" im Zusammenhang mit künstlichen neuronalen Netzen eigentlich gemeint? Genauer: was soll durch das Lernen erreicht werden?

Ein typisches Lernziel für ein neuronales Netz ist die Approximation einer mathematischen Funktion $f : \mathbb{R}^n \longrightarrow \mathbb{R}^m$. Das Netz bekommt zum Lernen bzw. Trainieren eine Menge von Paaren aus Eingaben $\vec{x}$ und gewünschten Ausgaben $\vec{y} = f(\vec{x})$ präsentiert. Dabei sollen die Gewichte so eingestellt werden, daß anschließend zu jedem Wert $\vec{x}$

aus der Trainingsmenge die berechnete Ausgabe möglichst genau $f(\vec{x})$ ist. Weiterhin sollen berechnete Ausgaben zu unbekannten Eingaben, die nicht in der Trainingsmenge waren, zu einer guten Approximation von f führen.

Die Fähigkeit, zu unbekannten (d.h. nicht trainierten) Eingaben die korrekte Ausgabe zu berechnen, wird *Generalisierungsfähigkeit* genannt. Eine andere übliche Aufgabe ist die Klassifikation unbekannter Datensätze. Das neuronale Netz soll Gemeinsamkeiten unter den Eingaben erkennen und unterschiedliche Klassen bilden, so daß ähnliche Eingaben der selben Klasse zugeordnet werden. In diesem Fall werden dem Netz nur Eingabe–Datensätze ohne gewünschte Ausgaben präsentiert.

Definition 2.8 *Grundsätzlich werden folgende drei Trainingsmethoden unterschieden:*

1. *Beim überwachten Lernen werden dem Netz in jedem Trainingsbeispiel ein Musterpaar (engl: pattern = Muster) $(\vec{x}^{(p)}, \vec{y}^{(p)})$ aus Eingabe und gewünschter Ausgabe präsentiert. Nach Berechnung der Netzausgabe wird der (quadratische) Ausgabefehler des Netzes zwischen berechneter Ausgabe $\vec{o}^{(p)}$ und gewünschter Ausgabe $\vec{y}^{(p)}$ bei Muster p, F_p bestimmt, z.B. bei m Ausgabe–Neuronen durch:*

$$F_p = \sum_{j=1}^{m} \left(y_j^{(p)} - o_j^{(p)} \right)^2$$

   Ziel ist es, durch Änderung der Gewichte das Netz so einzustellen, daß der Fehler bei allen Trainingsmustern minimal ist. Maßstab hierfür ist meistens der mittlere quadratische Fehler über alle Trainingsbeispiele.

2. *Bei verstärkendem Lernen wird dem Netz nur mitgeteilt, ob die berechnete Ausgabe bei einem Trainingsbeispiel gut oder schlecht ist. Entsprechend dieser Bewertung erfolgt die Anpassung so, daß gute Ergebnisse verstärkt und schlechte nach Möglichkeit vermieden werden.*

3. *Selbstorganisierende Netze bekommen gar keine Bewertung der berechneten Ausgabe. Ihr Lernziel ist es, bei den Eingabebeispielen bestimmte Muster oder Gemeinsamkeiten zu erkennen. Die Änderungen in dem Netz bewirken eine Einteilung der Beispiele in verschiedene Kategorien bzw. Klassen. Nach dem Training sollen auch unbekannte Datensätze richtig klassifiziert werden. D.h. ein unbekannter Datensatz soll der Klasse zugeordnet werden, dessen erzeugende Trainingsbeispiele diesem Datensatz am ähnlichsten sind.*

Das Training eines künstlichen neuronalen Netzes verläuft üblicherweise so, daß dem Netz in mehreren (d.h. bis zu tausenden) Durchgängen alle Trainingsbeispiele der Reihe

nach präsentiert werden. Dabei erfolgt entweder sofort für jedes Beispiel eine geringfügige Änderung der Gewichte (*online–Training*), oder die Änderungen werden jeweils für eine bestimmte Anzahl von Beispielen aufsummiert und gemittelt und erst danach tatsächlich durchgeführt (*offline–Training*). In jedem Fall ist es das Ziel, die Gewichte so einzustellen, daß nach dem Training alle Beispiele optimal verarbeitet werden.

Zunächst einmal gilt daher: je länger trainiert wird, desto besser ist die Anpassung des Netzes an die Trainingsdaten. Hieraus kann aber nicht gefolgert werden, daß grundsätzlich ein längeres Training immer zu besseren Ergebnissen führt:

Die eigentliche Aufgabe des Netzes ist es, nach dem Training bei unbekannten Eingaben gute Ergebnisse zu liefern (z.B. die Approximation der gelernten Funktion). Hierfür ist eine gute Generalisierungsfähigkeit des Netzes erforderlich. Wird das Netz nun sehr lange trainiert, ist es zu genau an die speziellen Trainingsdaten angepaßt, wodurch die Generalisierungsfähigkeit verringert wird. Dies hat zur Folge, daß bei unbekannten Eingaben die Ergebnisse bei fortdauerndem Training schlechter werden. Dieser Effekt wird als *Overtraining* bezeichnet.

Um das Overtraining zu reduzieren, werden zwei disjunkte Mengen von Datensätzen verwendet, die Trainingsmenge und die *Validations–Menge*. Beim Training wird die Trainingsmenge verwendet und parallel der Ausgabe–Fehler des Netzes auf der Validations–Menge bestimmt. Zunächst wird dieser Fehler absinken, doch nach einiger Zeit plötzlich wieder ansteigen. Dieser Wechsel ergibt einen guten Zeitpunkt, das Training zu beenden. Um ferner zu vermeiden, daß das Netz auch die Reihenfolge der Trainingsdaten lernt, empfiehlt es sich, die Datensätze nach jedem Durchgang zu permutieren. So kann eine bessere Approximations–Fähigkeit erreicht werden.

Nach dem Training kann das Netz im optimalen Fall für jedes Trainingsbeispiel die korrekte Ausgabe berechnen. Um die Generalisierungsfähigkeit zu überprüfen, wird der mittlere quadratische Fehler auf einer (nicht trainierten) Testmenge bestimmt. Wenn dieser gering ist, kann davon ausgegangen werden, daß das Training erfolgreich war. Andernfalls ist ein erneutes Training erforderlich, eventuell unter Verwendung einer anderen Lernrate. Falls nach einigen Versuchen noch keine zufriedenstellenden Ergebnisse erzielt werden, ist es erforderlich, eine andere Struktur des Netzes zu wählen (z.B. mehr verborgene Neuronen oder eine zusätzliche verborgene Schicht einfügen) und das Training erneut durchzuführen.

2.4 Geschichte der künstlichen neuronalen Netze

1943 entwarfen Warren McCulloch und Walter Pitts [McCPit, 1943] das erstes Modell eines künstlichen Neurons, die *McCulloch–Pitts–Zelle*. Sie zeigten, daß im Prinzip ein einfaches Netz jede arithmetische oder logische Funktion berechnen kann. Praktische Anwendungen gab es allerdings erst später.

1949 formulierte Donald O. Hebb [Hebb, 1949] die Hebbsche Lernregel, die bis heute in den meisten Lernverfahren verwendet wird. 1958 gab es den ersten erfolgreichen Neurocomputer, das *Mark I Perzeptron* von Frank Rosenblatt und Charles Wightman [Rosenblatt, 1958]. Es konnte bereits Muster wie z.B. einfache Ziffern richtig erkennen. In seinem Buch "Principles of Neurodynamics" [Rosenblatt, 1962] veröffentlichte Rosenblatt das *Perzeptron–Konvergenz–Theorem*. Damit bewies er, daß das Perzeptron mit seinem Lernverfahren alles, was es repräsentieren kann, auch lernen kann.

1960 stellten Bernard Widrow und Marcian E. Hoff [WidHof, 1960] das *Adaline* vor, ein Netz, das schnell und genau lernen kann. 1965 gab Nils Nilson in "Learning Machines" [Nilson, 1965] einen Überblick über die verschiedenen Modelle dieser Zeit. Damals glaubte man, bereits die grundlegenden Prinzipien selbstlernender intelligenter Systeme zu kennen.

1969 fand diese erste Blütezeit neuronaler Netze ein jähes Ende: Marvin Minsky und Seymour Papert [MinPap, 1969] zeigten, daß ein einzelnes Perzeptron viele wichtige Funktionen, wie z.B. die X–OR Funktion, überhaupt nicht repräsentieren kann. Weil sie daraus folgerten, daß auch andere Netzmodelle den gleichen Einschränkungen unterliegen, kam die Forschung auf diesem Gebiet fast zum Stillstand.

Dennoch wurden in den Jahren danach wichtige theoretische Grundlagen für aktuelle Entwicklungen geschaffen. 1972 stellte Teuvo Kohonen [Kohonen, 1972] einen *linearen Assoziierer* vor. 1974 entwickelte Paul Werbos [Werbos, 1974] das *Backpropagation–Verfahren*, heute das wichtigste Verfahren in der Praxis. In den folgenden 20 Jahren hat Stephen Grossberg [Grossberg, 1976] u.a. verschiedene Modelle der *Adaptiven Resonanz–Theorie (ART)* erstellt.

1980 stellten Fukushima, Miyake und Ito [Fukushima, 1980] das *Neocognitron* vor. 1982 beschrieb John Hopfield [Hopfield, 1982] das *Hopfield–Netz*, ein total verbundenes Netz zur Simulation des Verhaltens von *Spinglas–Atomen* und zur Lösung von Minimierungs–Aufgaben wie dem bekannten *Traveling Salesman Problem*. Im selben Jahr veröffentlichte Teuvo Kohonen [Kohonen, 1982] seine *selbstorganisierenden Karten*.

Seit Anfang der achtziger Jahre befindet sich die Forschung an neuronalen Netzen im Aufschwung. Großen Anteil daran haben die Veröffentlichungen von John Hopfield. Noch größer ist jedoch der Einfluß durch die Entwicklung und weite Verbreitung des Backpropagation–Verfahrens. Obwohl bereits 1974 von Paul Werbos entwickelt, wurde es erst 1986 durch Rumelhart, Hinton und Williams [RuHiWi, 1986] allgemein bekannt.

Das Backpropagation–Verfahren basiert auf einem mathematischen Gradientenabstiegsverfahren. Es ist ein im Vergleich zu anderen bis dahin bekannten Methoden sehr schnelles Lernverfahren für mehrschichtige Netze. Die Verbreitung des Backpropagation–Verfahrens war der große Durchbruch in der Entwicklung neuronaler Netze. Seitdem

sind neuronale Netze ein immer aktuelles Thema, mit dem sich viele Forscher beschäftigen. Es gibt inzwischen Variationen von Backpropagation, die noch schneller sind, z.B. das *Quickprop–Verfahren* von S.E. Fahlman [Fahlman, 1988]. Einen ganz anderen Ansatz bietet die Kombination von neuronalen Netzen mit Methoden der Fuzzy–Theorie. Mehr dazu in den Kapiteln 4 – 6.

Wie dieser geschichtliche Überblick zeigt, gibt es eine ganze Reihe von Modellen künstlicher neuronaler Netze. Einige davon sind Standard–Modelle geworden, die in der heutigen Praxis zu verschiedenen Zwecken zum Einsatz kommen. Neben dem klassischen *linearen Assoziierer* (Abschnitt 2.5) soll im Folgenden das *Backpropagation–Verfahren* (Abschnitt 2.6) als wichtigstes Verfahren der Praxis genauer beschrieben werden. In vielen Lehrbüchern zum Thema werden weitere Modelle vorgestellt, exemplarisch seien hier die Standard-Werke [Zell, 1996] und [Rojas, 1993] genannt.

2.5 Der lineare Assoziierer

Die Aufgabe eines *linearen Assoziierers* ist es, eine Menge von L Vektorpaaren

$(\vec{X}^1, \vec{Y}^1), \ldots, (\vec{X}^L, \vec{Y}^L), X^l \in \mathbb{R}^n, Y_l \in \mathbb{R}^m, l = 1, \ldots, L,$

zu assoziieren.

D.h. zur Eingabe $(x_1^l, \ldots, x_n^l)$ soll die berechnete Ausgabe jeweils $(y_1^l, \ldots, y_m^l)$ sein. Hierzu wird ein zweischichtiges neuronales Netz verwendet (Abb. 2.4).

Definition 2.9 *Ein linearer Assoziierer ist ein schichtweise total verbundenes zweischichtiges neuronales Netz mit n Eingabe–Neuronen und m Ausgabe–Neuronen:*

- *die Eingabeschicht dient zur Verteilung der Eingaben, d.h. die Aktivitäts– und Ausgabefunktionen sind jeweils die Identität*
- *die Ausgabe–Neuronen berechnen die Netzausgabe jeweils als gewichtete Summe ihrer Eingaben. D.h. die Aktivitätsfunktion ist die gewichtete Summe, die Ausgabefunktion die Identität.*

Entsprechend der Anwendung werden zwei unterschiedliche Variationen eines linearen Assoziierers unterschieden:

Definition 2.10
Ein linearer Assoziierer heißt auto–assoziativ, falls gilt:
$\vec{Y}^l = \vec{X}^l$ *für jedes* $l = 1, \ldots, L$.

Ein linearer Assoziierer heißt hetero–assoziativ, falls gilt:
$\vec{Y}^l \neq \vec{X}^l$ *für mindestens ein* l.

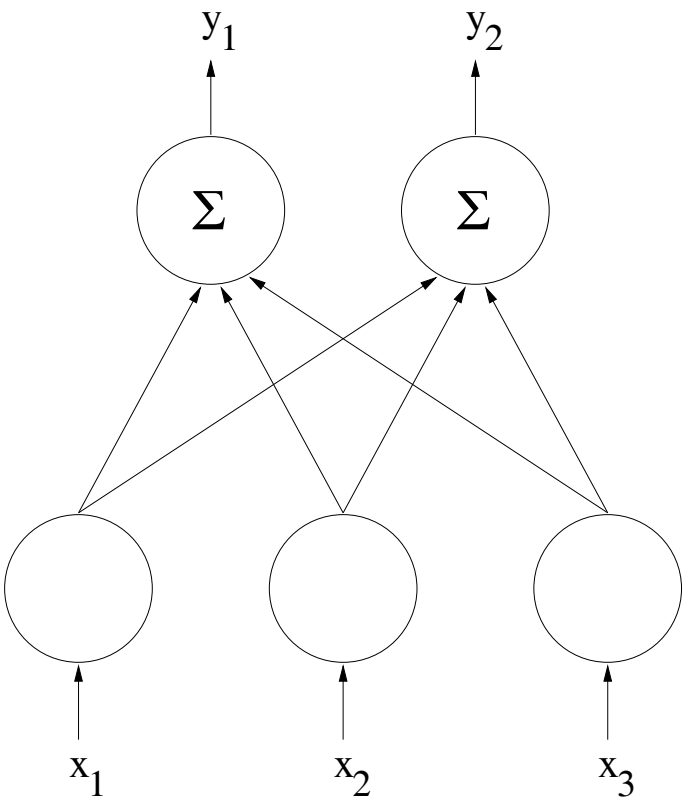

Abbildung 2.4: Ein linearer Assoziierer (3 Eingaben, 2 Ausgaben)

Die L Eingabe– und Ausgabe–Vektoren $\vec{X}^l$ und $\vec{Y}^l$, $l = 1, \ldots, L$, lassen sich jeweils als Spalten in eine Matrix eintragen:

$$X = \left(\vec{X}^1, \ldots, \vec{X}^L\right), \qquad Y = \left(\vec{Y}^1, \ldots, \vec{Y}^L\right)$$

Die Ausgabe o_j^l von Neuron $N_{2,j}$ der Ausgabeschicht bei Eingabe $\vec{X}^l$ wird berechnet gemäß:

$$o_j^l = \sum_{i=1}^{n} w_{ij} \cdot x_i^l$$

mit w_{ij} dem Gewicht der Verbindung zwischen Eingabe–Neuron $N_{1,i}$ und Ausgabe–Neuron $N_{2,j}$.

Auch die zu den Eingaben $\vec{X}^l$ berechneten Ausgabe–Vektoren $\vec{O}^l$, $l = 1, \ldots, L$, lassen sich in eine Matrix eintragen:

$$O = \left(\vec{O}^1, \ldots, \vec{O}^L\right)$$

Für die Gesamtheit der berechneten Ausgabe–Vektoren gilt somit:

$$O = W \cdot X$$

Die Aufgabe des Netzes ist entsprechend

$$Y = W \cdot X$$

zu erfüllen. D.h. die optimale Gewichtsmatrix W ergibt sich durch Lösung eines linearen Gleichungssystems:

$$W = Y \cdot X^{-1}$$

Die Inverse Matrix X^{-1} zu X existiert jedoch nicht immer. Eine mögliche Lösung bietet in diesem Fall die Verwendung der *Pseudo–Inversen*:

Definition 2.11 *Sei A eine reelle Matrix. A^+ heißt Pseudo–Inverse von A, falls gilt:*

1. $A \cdot A^+ \cdot A = A$

2. $A^+ \cdot A \cdot A^+ = A^+$

3. $A^+ \cdot A = (A^+ \cdot A)^T, A \cdot A^+ = (A \cdot A^+)^T$

wobei ^{T} für transponieren steht.

Der folgende Satz zeigt, daß die Pseudo–Inverse immer existiert und bei der Bestimmung der bestmöglichen Gewichtsmatrix hilft:

Satz 2.1
1. Zu jeder reellen Matrix A existiert die Pseudo–Inverse A^+ und ist eindeutig bestimmt. Falls A invertierbar ist, gilt $A^+ = A^{-1}$.

2. Seien X und Y reelle Matrizen. Dann minimiert $W = Y \cdot X^+$ die quadratische Norm von $W \cdot X - Y$.

Beweis 2.1 *Da diese Sätze bekannte mathematische Aussagen darstellen, sei zum Beweis auf die Fachliteratur verwiesen.* □

Folgerung dieses Satzes ist, daß mit der Gewichtsmatrix $W = Y \cdot X^+$ die optimale Gewichtsmatrix eines linearen Assoziierers zu den Eingaben X und gewünschten Ausgaben Y gefunden ist, d.h. W minimiert den quadratischen Fehler des Netzes. Zur Bestimmung der Pseudo–Inversen gibt es numerische Standard–Verfahren, auf die hier nicht eingegangen werden soll.

Eine Alternative zu numerischen Methoden bietet die Hebbsche Lernregel mit Lernrate $\eta = 1$:

- die Gewichtsmatrix wird mit 0 initialisiert
- nun wird der Reihe nach jedes Vektorpaar $\vec{X}^l, \vec{Y}^l$, $l = 1, \ldots, L$, einmal angelegt, d.h. das Training ist nicht iterativ
- die Gewichtsänderung erfolgt in jedem Schritt gemäß $\Delta w_{ij} = y_i^l \cdot x_j^l$
- die Gewichtsmatrix W_1 nach dem ersten Trainingsbeispiel ist somit $W_1 = \vec{Y}^1 \cdot \vec{X}^{1T}$
- in jedem weiteren Schritt wird zur aktuellen Gewichtsmatrix W die Matrix $W_l = \vec{Y}^l \cdot \vec{X}^{lT}$, $l = 2, \ldots, L$, hinzu addiert

Somit gilt für die endgültige Gewichtsmatrix W bei Anwendung dieser Regel:

$$W = W_1 + \ldots + W_L = Y \cdot X^T$$

Mit der Hebbschen Lernregel ergibt sich als Gewichtsmatrix nicht die optimale Lösung $Y \cdot X^+$, sondern $Y \cdot X^T$. Dies ist der erste Term einer Summenentwicklung von $Y \cdot X^+$ und ergibt häufig einen hinreichend geringen Fehler. Vorteil dieser Methode ist die einfache Anwendung der Lernregel und der Verzicht auf numerische Methoden.

2.6 Das Backpropagation–Verfahren

Das Backpropagation–Verfahren ist ein Lernverfahren für klassische mehrschichtige feedforward–Netze mit mindestens einer verborgenen Schicht. Die Aufgabe des Netzes ist es, eine Funktion $f : A \subset \mathbb{R}^n \longrightarrow \mathbb{R}^m$ zu approximieren. Im Gegensatz zu zweischichtigen Netzen ohne verborgene Schicht können auch nicht–lineare Funktionen wie X–OR gelernt werden. Es findet ein überwachtes Lernen statt, die Trainingsmenge besteht daher aus Beispielpaaren $\left(\vec{x}^{(p)}, \vec{y}^{(p)} = f(\vec{x}^{(p)})\right)$.

Schon vor Entdeckung des Backpropagation–Verfahrens war bekannt, daß die X–OR Funktion mit einem dreischichtigen Netz berechenbar ist. Allerdings gab es keine Methoden, um so ein Netz zu trainieren. Das Problem war, daß es keine Möglichkeit gab, den Fehler oder ein passendes Gewicht eines verborgenen Neurons zu berechnen. Dies änderte sich erst durch das Backpropagation–Verfahren.

Das verwendete Netz wird *Multilayer–Perzeptron (MLP)* genannt. Die Eingabeschicht dient nur zur Verteilung der Eingaben. In jeder anderen Schicht berechnet die Aktivitätsfunktion in jedem Neuron die gewichtete Summe der Eingaben plus einem *Bias–Input* θ. Der Bias–Input wird einfach zur gewichteten Summe der Eingaben hinzu addiert. Somit gilt für die Aktivitätsfunktion von Neuron Nr. j aus Schicht h (h nicht die Eingabeschicht) $f_{a\_h,j}$:

$$f_{a\_h,j} = \sum_{i \in S_{h-1}} w_{h,i,j} \cdot o_{h-1,i} + \theta$$

mit S_{h-1} der $h-1$-ten Schicht, $w_{h,i,j}$ dem Gewicht der Verbindung zwischen Neuron Nr. i aus Schicht $h-1$ und Neuron Nr. j aus Schicht h sowie θ dem Bias–Input.

In manchen Modellen wird der Bias–Input auch als Eingabe eines zusätzlichen Neurons betrachtet, das konstant 1.0 ausgibt. In diesem Fall wird er, wie jede andere Eingabe auch, mit einem Gewichtswert multipliziert und aufaddiert. Der Bias–Input ersetzt den bei Schwellenwertfunktionen verwendeten Schwellenwert. Das Training bewirkt grundsätzlich eine Trennung der Eingabe–Daten durch eine Hyperebene. Der Bias–Input verschiebt diese vom Ursprung und erleichtert somit die optimale Einstellung.

Als Ausgabefunktion wird in jedem Neuron (außer in Schicht 1) die gleiche, nicht–lineare, differenzierbare Funktion f_o verwendet, meistens eine sigmoide Funktion. Mit

einer linearen Ausgabefunktion könnte das Netz nur lineare Funktionen berechnen. Die Differenzierbarkeit ist generell Voraussetzung für das Lernverfahren, wie bei der Herleitung deutlich wird (s. Abschnitt 2.7).

Definition 2.12 *Ein mehrschichtiges neuronales Netz mit den beschriebenen Eigenschaften heißt Multilayer–Perzeptron (abgekürzt: MLP).*

Oft wird die Struktur eines Multilayer–Perzeptrons durch schichtweise Angabe der Anzahl der verborgenen Neuronen angegeben:

Beispiel 2.2 *Ein 4–3–3–2 MLP (s. Abb. 2.5) enthält in der Eingabeschicht 4 Neuronen, in der ersten und zweiten verborgenen Schicht 3 Neuronen sowie in der Ausgabeschicht 2 Neuronen.*

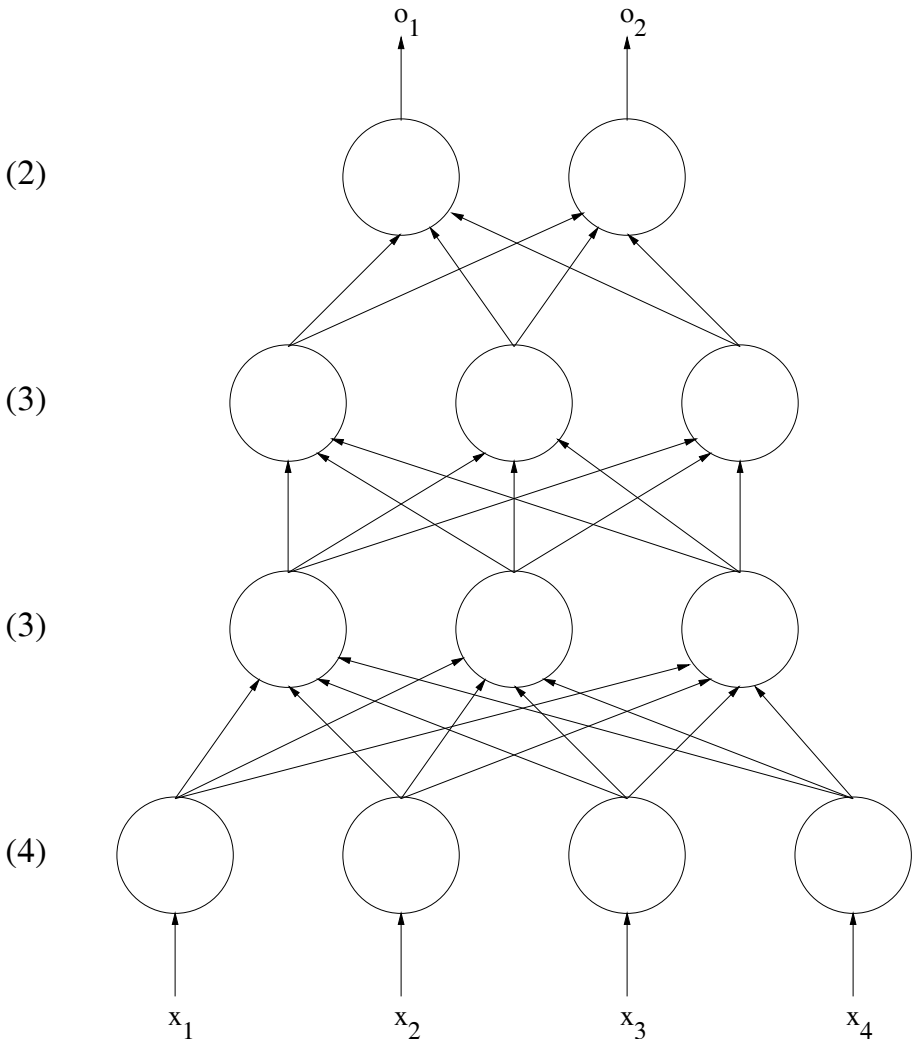

Abbildung 2.5: Ein 4–3–3–2 MLP

Das Lernen geschieht beim Backpropagation–Verfahren in zwei alternierenden Phasen, dem *Forward–Pass* und dem *Backward–Pass*. Vor dem Training werden die Gewichte des Netzes mit zufälligen Werten initialisiert. In der ersten Phase wird dem Netz ein Eingabe–Muster präsentiert und die Ausgabe des Netzes bei Verwendung der aktuellen Gewichte berechnet. In der zweiten Phase wird der Fehler dieser Ausgabe bestimmt und rückwärts durch das Netz propagiert. Auf diese Weise kann nacheinander, Schicht für Schicht, der Fehleranteil von jedem verborgenen Neuron bestimmt werden. Abhängig davon wird anschließend die Änderung der Gewichte festgelegt.

Dieser Vorgang wird der Reihe nach für alle Trainingsbeispiele wiederholt durchgeführt. Um zu vermeiden, daß sich die Gewichtsänderungen verschiedener Beispiele gegenseitig aufheben, ist es möglich, offline–Training durchzuführen: die Anpassung der Gewichte erfolgt hierbei jeweils erst nach Repräsentation einer vorgegebenen Anzahl der Trainingsbeispiele. Die einzelnen Werte zur Gewichtsänderung werden dazu aufsummiert und gemittelt, bevor die Gewichte angepaßt werden.

Durch das Backpropagation–Verfahren soll für jedes Eingabemuster der Fehler der Netzausgabe so gering wie möglich werden. Genauer soll der *mittlere quadratische Fehler* minimiert werden. Die Idee beim Backpropagation ist es, den mittleren quadratischen Fehler als *Fehlerfunktion* F in Abhängigkeit aller Gewichte zu betrachten:

Definition 2.13 *Fehlerfunktion*

$$F\left(\vec{W}\right) := \lim_{N\to\infty} \frac{1}{N} \cdot \sum_{p=1}^{N} F_p\left(\vec{x}^{(p)}, \vec{W}\right)$$

Dabei bezeichnet $\vec{W} = (w_1, \ldots, w_q)$ den Gewichtsvektor des Netzes, d.h. $\vec{W}$ besteht aus den durchnumerierten Gewichten aller Verbindungen des Netzes.

$$F_p\left(\vec{x}^{(p)}, \vec{W}\right) := \frac{1}{2} \cdot \sum_{j=1}^{m} \left(\vec{y}_j^{(p)} - o_{H,j}^{(p)}(\vec{W})\right)^2$$

ist der quadratische Fehler bei Eingabe $\vec{x}^{(p)}$ und verwendetem Gewichtsvektor $\vec{W}$, wobei $\vec{o}_H^{(p)}(\vec{W})$ die berechnete Netzausgabe zu Eingabe $\vec{x}^{(p)}$ und Gewichtsvektor $\vec{W}$ ist (Der Faktor $\frac{1}{2}$ ist ein "Dummy", der beim Ableiten gegen die Potenz 2 weggekürzt wird, s. Abschnitt 2.7).

Nach [Hecht–Nielsen, 1991] ist F fast sicher konvergent sowie stetig und differenzierbar in $\vec{W}$. Mathematisch ergibt nun die Funktion F in Abhängigkeit des Gewichtsvektors

$\vec{W}$ die *Fehleroberfläche.* Dort soll nun mit Hilfe eines *Gradientenabstiegsverfahrens* ein globales Minimum gefunden werden. D.h. es soll unter allen möglichen Gewichtsvektoren einer gefunden werden, bei dem der mittlere quadratische Fehler minimal ist. Da der Gradient eines Vektors gerade in die Richtung des steilsten Anstiegs zeigt, erfolgt eine Bewegung in die entgegengesetzte Richtung. Das ist nicht immer die Richtung des steilsten Abstiegs, aber meistens ein "guter" Weg.

Bei einem H–schichtigen MLP geschieht nun die Änderung des Gewichtes $w_{h,i,j}$ der Verbindung von Neuron i aus Schicht $h-1$ zu Neuron j aus Schicht h, $2 \leq h \leq H$, beim p–ten Muster nach folgender Formel:

Definition 2.14 *Verallgemeinerte δ–Regel*

$$\Delta w_{h,i,j}^{(p)} = -\eta \cdot \delta_{h,j}^{(p)} \cdot o_{h-1,i}^{(p)}$$

mit

$$\delta_{h,j}^{(p)} = \begin{cases} f'_{o\_h,j}\left(f_{a\_h,j}^{(p)}\right) \cdot \left(y_j^{(p)} - o_{H,j}^{(p)}\right), & \text{für } h = H \\ f'_{o\_h,j}\left(f_{a\_h,j}^{(p)}\right) \cdot \sum\limits_{\tilde{j} \in S_{h+1}} \delta_{h+1,\tilde{j}}^{(p)} \cdot w_{h+1,j,\tilde{j}}, & \text{für } 2 \leq h \leq H-1 \end{cases}$$

sowie:

- $\eta > 0$ *der Lernrate*
- $f_{a\_h,j}^{(p)}$ *der gewichteten Summe der Eingaben von Neuron j aus Schicht h (inkl. Bias–Input) beim p–ten Eingabe–Muster*
- $o_{h-1,i}^{(p)}$ *bzw.* $o_{H,j}^{(p)}$ *der Ausgabe von Neuron i bzw. j aus Schicht $h-1$ bzw. H beim p–ten Eingabe–Muster*
- $y_j^{(p)}$ *der gewünschten Ausgabe von Neuron j der Ausgabeschicht beim p–ten Eingabe–Muster*
- S_{h+1} *der $h+1$–ten Schicht*

Definition 2.15 $\delta_{h,j}$ *ist der Fehleranteil von Neuron Nr. j aus Schicht h.*

Der Fehleranteil $\delta_{h,j}$ von Neuron j aus Schicht h wird für $h = H$ (Ausgabeschicht) direkt bestimmt, in den verborgenen Schichten h, $2 \leq h \leq H-1$, wird $\delta_{h,j}$ jeweils mit Hilfe der Fehleranteile der Neuronen der darüberliegenden Schicht bestimmt.

Auf diese Weise ist es rekursiv möglich, für alle verborgenen Neuronen ihre Fehleranteile zu berechnen und die Gewichte anzupassen.

Definition 2.16 *Dieses Lernverfahren heißt in Anlehnung an die sogenannte* Delta–Regel *für ein zweischichtiges Perzeptron (s. [Zell, 1996]) verallgemeinerte Delta–Regel oder einfach Backpropagation–Algorithmus.*

Die Backpropagation–Lernregel ist auch auf anders aufgebaute Netze anwendbar, vorausgesetzt, die im Forward–Pass verwendeten Funktionen sind differenzierbar. Im folgenden Abschnitt 2.7 wird die Herleitung der verallgemeinerten Delta–Regel vorgeführt.

2.7 Herleitung der Backpropagation–Regel

Um den gewünschten Gradientenabstieg durchführen zu können, ist der Gradient der Fehlerfunktion F zu bestimmen. Da F fast sicher konvergent und differenzierbar ist, gilt:

$$\nabla_{\vec{W}} F(\vec{W}) = \lim_{N\to\infty} \frac{1}{N} \cdot \sum_{p=1}^{N} \nabla_{\vec{W}} F_p\left(\vec{x}^{(p)}, \vec{W}\right)$$

Es genügt also, die partiellen Ableitungen von F_p nach den Gewichten zu bestimmen. Dabei ist jedes Gewicht w_k des Gewichtsvektors genau ein Verbindungsgewicht $w_{h,i,j}$.

Somit ergeben sich folgende Berechnungen:

$$F_p\left(\vec{x}^{(p)}, \vec{W}\right) := \frac{1}{2} \cdot \sum_{j=1}^{m} \left(\vec{y}_j^{(p)} - o_{H,j}^{(p)}(\vec{W})\right)^2$$

Die Ausgabe $o_{h,j}^{(p)}$ eines Neurons j aus Schicht $h, 2 \le h \le H$, beim p–ten Muster ergibt sich durch

$$o_{h,j}^{(p)} = f_{o\_h,j}\left(f_{a\_h,j}^{(p)}\right)$$

mit

$$f_{a\_h,j}^{(p)} = \sum_{i\in S_{h-1}} w_{h,i,j} \cdot o_{h-1,i}^{(p)} + \theta$$

Nun muß F_p nach $w_{h,i,j}$ abgeleitet werden. Mit der Kettenregel ergibt sich:

$$\Delta w_{h,i,j}^{(p)} = -\frac{\partial F_p}{\partial w_{h,i,j}} = -\frac{\partial F_p}{\partial f_{a\_h,j}^{(p)}} \cdot \frac{\partial f_{a\_h,j}^{(p)}}{\partial w_{h,i,j}}$$

Aus der Definition von $f_{a\_h,j}^{(p)}$ ergibt sich für den zweiten Faktor:

$$\frac{\partial f_{a\_h,j}^{(p)}}{\partial w_{h,i,j}} = \frac{\partial}{\partial w_{h,i,j}} \cdot \sum_{\tilde{i} \in S_{h-1}} w_{h,\tilde{i},j} \cdot o_{h-1,\tilde{i}}^{(p)} + \theta = o_{h-1,i}^{(p)}$$

Der erste Faktor ergibt den Fehleranteil eines Neurons j aus Schicht h bei Muster Nr. p:

$$\delta_{h,j}^{(p)} = -\frac{\partial F_p}{\partial f_{a\_h,j}^{(p)}}$$

Einsetzen von $\delta_{h,j}^{(p)}$ und $o_{h-1,i}^{(p)}$ und hinzufügen der Lernrate ergibt die gewünschte Formel:

$$\Delta w_{h,i,j}^{(p)} = -\eta \cdot \delta_{h,j}^{(p)} \cdot o_{h-1,i}^{(p)}$$

Nun muß noch $\delta_{h,j}^{(p)}$ bestimmt werden. Erneute Anwendung der Kettenregel ergibt:

$$\delta_{h,j}^{(p)} = -\frac{\partial F_p}{\partial f_{a\_h,j}^{(p)}} = -\frac{\partial F_p}{\partial o_{h,j}^{(p)}} \cdot \frac{\partial o_{h,j}^{(p)}}{\partial f_{a\_h,j}^{(p)}}$$

Nach der Definition von $o_{h,j}^{(p)}$ ist

$$\frac{\partial o_{h,j}^{(p)}}{\partial f_{a\_h,j}^{(p)}} = f'_{o\_h,j}\left(f_{a\_h,j}^{(p)}\right)$$

Für den ersten Faktor der Gleichung muß unterschieden werden, ob h die Ausgabeschicht ist oder nicht.

1. h ist die Ausgabeschicht:

Aus der Definition von F_p folgt:

$$\frac{\partial F_p}{\partial o_{h,j}^{(p)}} = -\left(y_j^{(p)} - o_{H,j}^{(p)}\right)$$

Also ist

$$\delta_{h,j}^{(p)} = f'_{o\_h,j}\left(f_{a\_h,j}^{(p)}\right) \cdot \left(y_j^{(p)} - o_{H,j}^{(p)}\right)$$

2. h ist nicht die Ausgabeschicht, $2 \leq h \leq H-1$:

Mit der Kettenregel ergibt sich eine Summe über die Neuronen aus der nächsthöheren Schicht, mit denen das Neuron j aus Schicht h verbunden ist:

$$\frac{\partial F_p}{\partial o_{h,j}^{(p)}} = \sum_{\tilde{j} \in S_{h+1}} \left(\frac{\partial F_p}{\partial f_{a\_h+1,\tilde{j}}^{(p)}} \cdot \frac{\partial f_{a\_h+1,\tilde{j}}^{(p)}}{\partial o_{h,j}^{(p)}} \right)$$

$$\overset{\text{Def. } f_a}{=} \sum_{\tilde{j} \in S_{h+1}} \left(\frac{\partial F_p}{\partial f_{a\_h+1,\tilde{j}}^{(p)}} \cdot \frac{\partial}{\partial o_{h,j}^{(p)}} \cdot \sum_{\tilde{i} \in S_h} w_{h+1,\tilde{i},\tilde{j}} \cdot o_{h,\tilde{i}}^{(p)} + \theta \right)$$

$$= \sum_{\tilde{j} \in S_{h+1}} \frac{\partial F_p}{\partial f_{a\_h+1,\tilde{j}}^{(p)}} \cdot w_{h+1,j,\tilde{j}}$$

$$= -\sum_{\tilde{j} \in S_{h+1}} \delta_{h+1,\tilde{j}}^{(p)} \cdot w_{h+1,j,\tilde{j}}$$

Einsetzen ergibt schließlich die gesuchte Formel:

$$\delta_{h,j}^{(p)} = f'_{o\_h,j}\left(f_{a\_h,j}^{(p)}\right) \cdot \sum_{\tilde{j} \in S_{h+1}} \delta_{h+1,\tilde{j}}^{(p)} \cdot w_{h+1,j,\tilde{j}}$$

Die Fehleranteile $\delta_{h,j}$ von Neuronen aus verborgenen Schichten werden demnach mit Hilfe der Fehleranteile von Neuronen aus der nächst höheren Schicht berechnet. Somit ergibt sich eine rekursive Prozedur, um im Backward–Pass schichtweise die Fehleranteile zu bestimmen. Bei der Durchführung der Gewichtsänderungen ist darauf zu achten, die $\delta_{h,j}$ jeweils mit den alten Gewichts–Werten (vor der Änderung) zu berechnen, da diese im Forward–Pass und bei der Herleitung eingesetzt wurden.

2.8 Modifikationen von Backpropagation

Das Backpropagation–Verfahren ist ein *lokales Verfahren*, d.h. es hat keine globale Information über die Fehleroberfläche zur Verfügung. Die Gewichtsänderungen basieren also nur auf Kenntnis des Gradienten an jeweils einer Stelle. Dadurch können einige Probleme bei der Suche eines globalen Minimums entstehen.

Je mehr Gewichte es in dem Netz gibt, desto zerklüfteter wird die Fehleroberfläche. Da kann es passieren, daß das Backpropagation–Verfahren in einem lokalen Minimum hängen bleibt, das größer als das globale Minimum ist. In einem lokalen Minimum nimmt der Gradient den Wert 0 an, so daß keine Änderung der Gewichte mehr erfolgt. Oft kann bei Verwendung einer kleinen Lernrate ein ausreichend gutes suboptimales lokales Minimum gefunden werden. Eine allgemeingültige Empfehlung für den günstigsten Wert der Lernrate gibt es allerdings nicht. Dieser hängt von der gewählten Netzstruktur und der Art der Trainingsdaten ab und muß experimentell bestimmt werden.

Wenn zu Beginn des Trainings jeweils alle Gewichte einer Schicht die gleichen Werte haben, ändern sich diese Gewichte in jedem Schritt schichtweise jeweils um den selben Wert. Daher können sie durch das Backpropagation–Verfahren keine unterschiedlichen Werte mehr annehmen, so daß ein Lernerfolg unmöglich wird. Dieses Problem wird mit *Symmetry–Breaking* bezeichnet. Es läßt sich durch Verwendung von zufälligen, unterschiedlichen Startgewichten leicht vermeiden.

Die Größe der Gewichtsänderungen hängt vom Betrag des Gradienten ab. Daher wird das Verfahren an flachen Stellen sehr langsam. Auf ganz ebenen Stellen kommt es sogar zum vollständigen Stillstand, da der Gradient dort 0 ist. Von außen kann man in diesem Fall nicht erkennen, ob der Stillstand an einer ebenen Stelle oder in einem Minimum erfolgt ist. Denn auch in einem Minimum ist der Gradient 0. Ebenso kann es passieren, daß das Verfahren in einer steilen Schlucht hin und her springt (oszilliert). Der Gradient ist dann so groß, daß in einem Schritt über die Schlucht hinweg gesprungen wird.

Die nächste Änderung der Gewichte bewirkt dann einen Sprung zurück zur anderen Seite, usw. (s. Abb. 2.6).

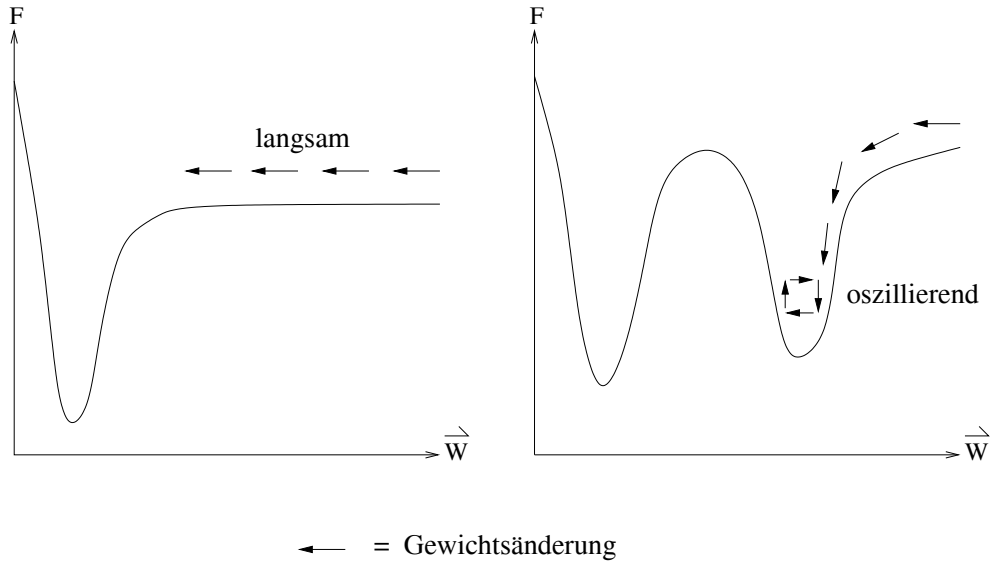

Abbildung 2.6: Mögliches Verhalten an flacher Stelle / in steiler Schlucht

Eine mögliche Lösung für diese beiden Probleme ist die zusätzliche Verwendung des *Momentum–Terms* bei der Backpropagation–Formel. Dieser berücksichtigt für die Berechnung der Gewichtsänderung zum Zeitpunkt $t+1$ die bis zum Zeitpunkt t erfolgten Änderungen. Dabei haben die weiter zurückliegenden Änderungen zunehmend weniger Einfluß. Die Anpassung der Gewichte erfolgt dann nach der Formel:

Definition 2.17 *Momentum–Version*

$$\Delta w_{h,i,j}^{(p)}(t+1) = -(1-\alpha) \cdot \eta \cdot \delta_{h,j}^{(p)} \cdot o_{h-1,i}^{(p)} + \alpha \cdot \Delta w_{h,i,j}^{(p)}(t)$$

wobei α ein Parameter aus $[0,1[$ ist, der Momentum–Term.

Die Anwendung der Momentum–Version bewirkt gleichzeitig eine Beschleunigung in flachen Gegenden und ein Abbremsen in steilen Schluchten.

Es kann auch vorkommen, daß das Verfahren aus dem globalen oder einem ähnlich guten Minimum wieder herausspringt. Befindet sich so ein Minimum in einer steilen Schlucht, dann ist der Gradient so groß, daß die Gewichtsänderung eventuell einen Sprung in ein anderes, weniger gutes Minimum bewirkt. Allerdings tritt dieses Problem nur selten auf.

Grundsätzlich gibt es leider keinen optimalen Wert für die Lernrate η. Zu kleine Werte machen das Lernen sehr langsam, zu große Werte verstärken die Probleme bei tiefen Schluchten. Eine erfolgversprechende Möglichkeit ist es, die Lernrate während des Trainings langsam zu verringern. Noch besser ist es, nach einer Idee von R. A. Jacobs [Jacobs, 1988], für jedes Gewicht w_k eine eigene, variable Lernrate η_k zu verwenden. Dadurch kann die in jeder Dimension und an verschiedenen Stellen unterschiedliche Krümmung der Fehleroberfläche berücksichtigt werden. Dies wird am besten durch die $\bar{\delta} - \delta$-*Regel* realisiert:

Definition 2.18 $\underline{\bar{\delta} - \delta\text{–Regel}}$

$$\eta_k(t+1) = \eta_k(t) + \Delta\eta_k(t)$$

mit

$$\Delta\eta_k(t) = \begin{cases} \kappa, & \textit{falls } \bar{\delta}_k(t-1) \cdot \delta_k(t) > 0 \\ -\phi \cdot \eta_k(t), & \textit{falls } \bar{\delta}_k(t-1) \cdot \delta_k(t) < 0 \\ 0, & \textit{sonst} \end{cases}$$

sowie

$$\delta_k(t) = \frac{\partial F}{\partial w_k}(t)$$

und

$$\bar{\delta}_k(t) = (1-\theta) \cdot \delta_k(t) + \theta \cdot \bar{\delta}_k(t-1)$$

κ, θ, ϕ *sind Konstanten:* $\kappa > 0$*;* $\theta, \phi \in [0,1]$.

Diese Methode hat folgende Auswirkungen: wenn das Vorzeichen der partiellen Ableitung von F nach einem Gewicht w_k gleich bleibt, wächst die zugehörige Lernrate η_k linear, sie wird also zunehmend größer. Sobald sich jedoch das Vorzeichen ändert, sinkt die Lernrate exponentiell, sie wird also sehr schnell wieder klein.

Die Heuristik hinter diesen Modifikationen besagt, daß bei der Überwindung von langen, flachen Plateaus auf der Fehleroberfläche die partielle Ableitung stets das gleiche Vorzeichen hat, während in zerklüfteten Gegenden das Vorzeichen oft wechselt. Somit wird durch die $\bar{\delta} - \delta$–Regel ein flaches Plateau sehr schnell überwunden, um sofort danach mit kleinen Schritten fortzufahren. Beim möglichen Oszillieren wird die Lernrate nach wenigen Schritten so klein, daß dieses schnell vorbei ist. Daher werden auch mit dieser Regel sowohl die Probleme an flachen Stellen als auch die Probleme in steilen Schluchten vermieden.

2.9 Universalität von neuronalen Netzen

Abschließend soll nun der Frage nachgegangen werden, welche Fähigkeiten neuronale Netze theoretisch haben. Zu wissen, welche Funktionen grundsätzlich von einem — wie auch immer aufgebauten — neuronalen Netz repräsentiert werden können, ist von großem Vorteil. Ist für eine Klasse von Funktionen die Repräsentation durch ein neuronales Netz mit Sicherheit irgendwie möglich, lohnt sich die Suche nach einem geeigneten Netz eher, als wenn gar nicht klar ist, ob ein positives Ergebnis überhaupt existieren kann. Ein wichtiges Ergebnis in diesem Zusammenhang liefert der *Satz von Kolmogorov*:

Satz 2.2 *Satz von Kolmogorov*

Sei $f : [0,1]^n \longrightarrow \mathbb{R}^m$ *eine stetige Funktion. Dann gilt:*

f kann durch ein dreischichtiges feedforward-Netz mit den folgenden Eigenschaften exakt implementiert werden.

1. *Die Eingabeschicht besteht aus n Verteilerneuronen.*
2. *Die verborgene Schicht besteht aus* $2 \cdot n + 1$ *Neuronen, die jeweils ihre Ausgabe wie folgt berechnen:*

$$o_{2,k} = \sum_{i=1}^{n} \left(\lambda(n)^k \cdot \Psi(n, o_{1,i} + k \cdot \varepsilon) \right) + k, \ 1 \leq k \leq 2 \cdot n + 1$$

   mit

   (a) $\lambda(n) \in \mathbb{R}$, *konstant*
   (b) $\Psi : \mathbb{R}^2 \longrightarrow \mathbb{R}$, *stetig und monoton*
   (c) $\varepsilon \in \mathbb{Q}$, *konstant, mit* $0 < \varepsilon \leq \delta$, $\delta > 0$ *beliebig*

3. *Die Ausgabeschicht besteht aus m Neuronen, die ihre Ausgabe wie folgt berechnen:*

$$o_{3,j} = \sum_{k=1}^{2 \cdot n + 1} g_j(f, \varepsilon, o_{2,k}), \ 1 \leq j \leq m$$

   mit g_j *stetig und reell.*

Dabei hängt die Funktion Ψ aus Schicht 2 nur von n ab, jedoch nicht von f selber. Die Funktionen g_j aus Schicht 3 hängen von f und ε ab, jedoch nicht von n.

Dieser Satz macht eine reine Existenzaussage und liefert keinen Hinweis für die Wahl von Ψ und den g_j. Ebenso ist der Beweis nicht konstruktiv. Daher sei diesbezüglich auf

[Hecht–Nielsen, 1987] verwiesen. Obwohl sich aus diesem Satz keine "Anleitung" zum Erstellen des neuronalen Netzes ergibt, ist er von grundlegender Bedeutung für dieses Forschungsgebiet. Denn damit wissen wir, daß neuronale Netze *universelle Approximatoren* sind.

Für den Einsatz von neuronalen Netzen sprechen folgende Vorteile:

- Lernfähigkeit
- Parallelität
- verteilte Wissensrepräsentation
- Universalität

Dem stehen folgende Nachteile gegenüber:

- Wissenserwerb ausschließlich durch lernen
- Analyse des Gelernten schwierig (Blackbox)
- Trainingsdaten erforderlich
- keine direkte Übernahme von Regeln eines Experten

Fuzzy–Controller sind Steuersysteme, die gerade diese Nachteile von neuronalen Netzen nicht haben. Im folgenden Kapitel 3 wird eine Einführung in die Fuzzy–Theorie gegeben, ebenso eine Vorstellung von Fuzzy–Controllern.

Kapitel 3

Fuzzy–Systeme

Fuzzy–Mengen sind eine Erweiterung der klassischen Mengen in der Mathematik, mit deren Hilfe sich viele Sachverhalte genauer und flexibler darstellen lassen. Z.B. ist nach Festlegung des internationalen Instituts der Statistik von 1887 (s. [Brockhaus, 1998]) eine *Großstadt* eine Stadt mit mehr als 100.000 Einwohnern. Ist also eine Stadt mit 99.999 Einwohnern eine Kleinstadt? Bei klassischen Mengen muß hier eine feste Zuordnung getroffen werden, während Fuzzy–Mengen eine graduelle Zuordnung erlauben, die wesentlich feinere Beschreibungen ermöglicht. Eine Stadt mit 99.999 Einwohnern ist etwa zum Grad $0,99$ eine Großstadt und zum Grad $0,01$ eine Kleinstadt, während eine Stadt mit 50.000 Einwohnern nur zum Grad $0,5$ eine Großstadt ist. Fuzzy–Mengen und ihre wichtigste Anwendung, der *Fuzzy–Controller*, sind Gegenstand dieses Kapitels.

3.1 Historische Entstehung

Die Fuzzy–Logik (deutsch: unscharfe Logik) stellt eine Erweiterung der klassischen zweiwertigen Logik dar, welche ihre Ursprünge bereits in der alten griechischen Philosophie hat: Platons Vermutung, daß es einen dritten Bereich zwischen "wahr" und "falsch" geben müsse, ist der antike Vorläufer des Prinzips der Fuzzy–Logik. Dennoch postulierte Platons Schüler Aristoteles das Gesetz vom ausgeschlossenen Dritten, das die Entwicklung der Mathematik und der Logik für die nächsten zwei Jahrtausende bestimmen sollte.

Moderne Philosophen wie G. Hegel und B. Russel nahmen schließlich Platons Vermutung wieder auf. So schrieb Russel [Russel, 1923]: "The law of excluded middle is true, when precise symbols are employed, but it is not true, when symbols are vague, as, in fact, all symbols are." Beispielhaft stellte er die Ungenauigkeit in der Sprache an der Farbe rot dar. Diese Farbe steht nicht für eine genau festgelegte Wellenlänge, sondern sie beschreibt einen bestimmten Bereich im Spektrum aller Farben. Ferner formulierte Russel das Paradoxon des Barbiers von Sevilla: Wenn alle Männer, die sich nicht selber rasieren, vom Barbier rasiert werden, wer rasiert dann den Barbier? Diese Frage ist in der zweiwertigen Logik nicht lösbar.

Zur selben Zeit führt J. Lukasiewicz [Lukasiewicz, 1957] als erster eine systematische Alternative zur zweiwertigen Logik des Aristoteles ein. Lukasiewics zeigt, daß es Sätze gibt, denen keiner der beiden Wahrheitswerte "wahr" (1) oder "falsch" (0) zugeordnet werden kann. Hieraus folgert er, daß es einen dritten Wahrheitswert geben muß, den er "possible" ($\frac{1}{2}$) nennt. Später entwickelte er dazu noch vier– und fünf–wertige Logiken. Auch unendlich–wertige Logiken auf dem Intervall $[0, 1]$, die in der Fuzzy–Logik konkret definiert werden, hielt er für möglich.

Die Ideen von Russel nahm M. Black [Black, 1937] wieder auf und entwarf ein Verfahren, mit dem er die Unschärfe von Symbolen numerisch darstellen konnte. Dabei wird die Ungenauigkeit oder Vagheit eines Symbols unter Einbeziehung des Komplements definiert. Black ging davon aus, daß es mindestens ein Element gibt, das weder zum Symbol noch zu dessen Komplement vollständig gehört. Die Menge der Elemente, die nicht eindeutig zugeordnet werden können, nennt er *frings* (deutsch: Fransen).

Die heute bekannte Fuzzy–Theorie wurde schließlich von L. Zadeh begründet. 1965 veröffentlichte er den Artikel "Fuzzy–Sets" [Zadeh, 1965], in dem er die Mathematik der Fuzzy–Set–Theorie beschreibt. Darin verbindet er die Idee der Fransen mit der Idee der unendlichwertigen Logik von Lukasiewics. Als wichtigste Anwendung dieser Theorie entwarf E. H. Mamdani [Mamdani, 1975] 1975 den Fuzzy–Controller.

3.2 Fuzzy–Mengen

Zu den nötigen theoretischen Grundlagen für die Beschreibung eines Fuzzy–Controllers gehören u.a. Fuzzy–Mengen und Operationen auf Fuzzy–Mengen. Was unterscheidet Fuzzy–Mengen von "gewöhnlichen" oder *crispen* Mengen? Der entscheidende Schritt bei Fuzzy–Mengen, wie auch in der Fuzzy–Theorie und der Fuzzy–Logik, ist die Erweiterung des klassischen binären 0 – 1 bzw. ja – nein Prinzips. Was ist damit gemeint?

Eine Teilmenge $A \subset X$ einer Grundmenge X läßt sich durch ihre *Indikatorfunktion* oder *charakteristische Funktion* $\mathcal{X}_A : X \longrightarrow \{0, 1\}$ eindeutig definieren:

Definition 3.1 *Sei $A \subset X$ Teilmenge einer Menge X. Die Funktion $\mathcal{X}_A : X \longrightarrow \{0, 1\}$ mit:*

$$\mathcal{X}_A(x) = \begin{cases} 1 : x \in A \\ 0 : x \notin A \end{cases}$$

heißt Indikatorfunktion von A.

Für jedes $x \in X$ gibt es dabei nur zwei Möglichkeiten: entweder ist x ein Element der Menge A oder nicht. Dabei gilt für jedes $x \in X$ immer einer dieser Fälle, und zwar nur

einer. Dementsprechend hat die Indikatorfunktion für jedes $x \in X$ entweder den Wert 0 oder den Wert 1, und keinen anderen.

Bei Fuzzy–Mengen wird der Bildbereich der Indikatorfunktion auf das ganze Intervall $[0, 1]$ erweitert:

Definition 3.2 *Sei X eine Menge. Eine Fuzzy–Menge $\widetilde{A} \subset X$ ist eine Menge von geordneten Paaren:*

$\widetilde{A} = \{(x, \mu_{\widetilde{A}}(x)) : x \in X\}$.

$\mu_{\widetilde{A}} : X \longrightarrow [0, 1]$ *heißt Zugehörigkeitsfunktion,*

$\mu_{\widetilde{A}}(x)$ *ist der Zugehörigkeitsgrad von x zur Fuzzy–Menge $\widetilde{A}$.*

Beispiel 3.1 $\{1, 3, 5\} \subset \{1, 2, 3, 4, 5\}$ *(crispe Teilmenge)*

Beispiel 3.2 $\{(1, 0.2), (2, 0.5), (3, 1.0), (4, 0.5), (5, 0.2)\} \subset \{1, 2, 3, 4, 5\}$ *(Fuzzy–Menge), wobei $(2, 0.5)$ bedeutet, daß die Zahl 2 mit Grad 0.5 zu dieser Fuzzy–Menge gehört.*

Der Unterschied zu "gewöhnlichen" Mengen ist also, daß nicht mehr einfach festgelegt wird, ob ein Element x in einer Menge enthalten ist oder nicht. Stattdessen wird für jedes Element x angegeben, zu welchem Grad es in der Menge liegt. Dabei bedeutet wie bei der Indikatorfunktion $\mu_{\widetilde{A}}(x) = 0$, daß x gar nicht in $\widetilde{A}$ liegt, $\mu_{\widetilde{A}}(x) = 1$ bedeutet, daß x ganz in $\widetilde{A}$ liegt. Somit ist die Zugehörigkeitsfunktion eine echte Erweiterung der charakteristischen Funktion, welche nur die Werte 0 und 1 annimmt.

Es gibt verschiedene Arten von Fuzzy–Mengen, die sich in ihrer Zugehörigkeitsfunktion und in der Art der Repräsentation unterscheiden. So läßt sich bei einer endlichen Grundmenge X für jeden Punkt $x \in X$ einzeln der Zugehörigkeitsgrad angeben, um eine Fuzzy–Menge zu definieren. Bei unendlichen Grundmengen ist natürlich eine (nicht notwendig stetige) Funktionsvorschrift mit Werten aus dem Intervall $[0, 1]$ erforderlich. Folgende Fuzzy–Mengen haben sich in der Praxis bewährt und treten besonders häufig auf:

Definition 3.3 *Gauß–Mengen, gegeben durch m (Mitte) und $w > 0$ (Weite):*

$$\mu_{\widetilde{A}}(x) = \exp\left(-\left(\frac{x - m}{w}\right)^2\right)$$

Definition 3.4 *Dreiecks–Mengen, gegeben durch l (links), m (Mitte) und r (rechts):*

$$\mu_{\widetilde{A}}(x) = \begin{cases} 0: & x \leq l \text{ oder } x \geq r \\ 1: & x = m \\ \frac{x-l}{m-l}: & l < x < m \\ \frac{m-x}{r-m} + 1: & m < x < r \end{cases}$$

Alternativ lassen sich Dreiecks–Mengen auch analog zu Gauß–Mengen mit zwei Parametern m (Mitte) und $w > 0$ (Weite) definieren:

$$\mu_{\widetilde{A}}(x) = \begin{cases} 0: & x \leq m - w \text{ oder } x \geq m + w \\ 1: & x = m \\ \frac{x-m+w}{w}: & m - w < x < m \\ \frac{m-x}{w} + 1: & m < x < m + w \end{cases}$$

Definition 3.5 *Trapez–Mengen, gegeben durch l (links), m_1 (Mitte 1), m_2 (Mitte 2) und r (rechts):*

$$\mu_{\widetilde{A}}(x) = \begin{cases} 0: & x \leq l \text{ oder } x \geq r \\ 1: & m_1 \leq x \leq m_2 \\ \frac{x-l}{m_1-l}: & l < x < m_1 \\ \frac{m_2-x}{r-m_2} + 1: & m_2 < x < r \end{cases}$$

Definition 3.6 *Sei $\widetilde{A} \subset X$ eine Fuzzy–Menge. Wenn es genau ein $m \in X$ gibt mit $\mu_{\widetilde{A}}(m) = 1$, dann heißt m der Modalwert von $\widetilde{A}$.*

Der Parameter w von Gauß– bzw. Dreiecks–Mengen bezeichnet die Weite dieser Mengen.

Beispiel 3.3 *Gauß–Mengen und Dreiecks–Mengen besitzen den Modalwert m. Trapez–Mengen haben keinen Modalwert, da bei ihnen auf der ganzen Strecke von m_1 bis m_2 der Zugehörigkeitsgrad den Wert 1 hat.*

Fuzzy–Mengen werden häufig durch den Graphen ihrer Zugehörigkeitsfunktion dargestellt. Die nachfolgende Abbildung 3.1 verdeutlicht die Unterschiede zwischen den verschiedenen Arten von Fuzzy–Mengen.

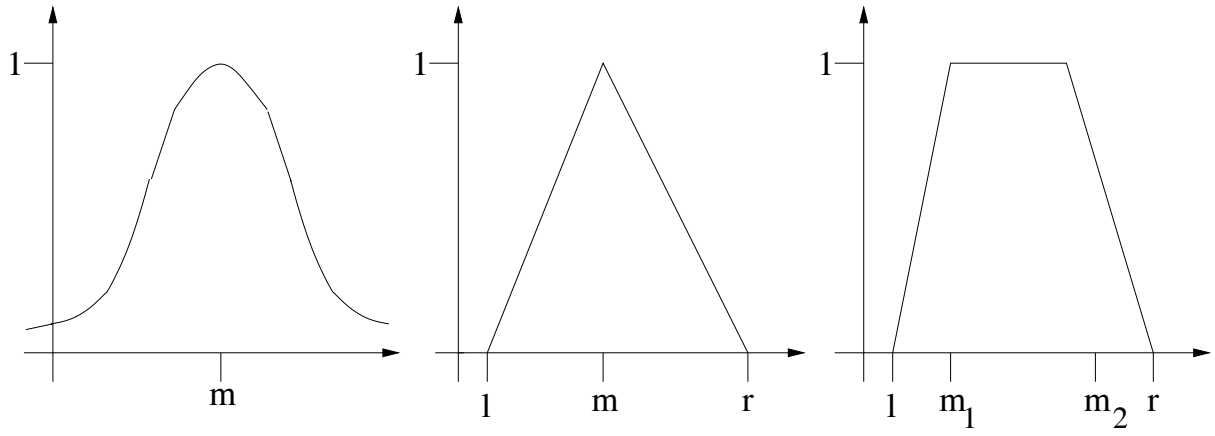

Abbildung 3.1: Gauß–Menge, Dreiecks–Menge, Trapez–Menge

Die Menge der Punkte, für die der Zugehörigkeitsgrad einer auf $\mathbb{R}$ definierten Fuzzy–Menge $\widetilde{A}$ größer bzw. größer oder gleich einem Wert α ist, nennt man (strengen) α–Schnitt:

Definition 3.7 *Sei $\widetilde{A} \subset \mathbb{R}$ eine Fuzzy–Menge auf der Grundmenge $\mathbb{R}$.*

$\widetilde{A}_\alpha := \{x \in \mathbb{R} \mid \mu_{\tilde{A}}(x) \geq \alpha\}$ *mit $\alpha \in [0,1]$ heißt* α*–Schnitt von $\tilde{A}$,*

$\widetilde{A}_{\bar{\alpha}} := \{x \in \mathbb{R} \mid \mu_{\tilde{A}}(x) > \alpha\}$ *mit $\alpha \in [0,1]$ heißt strenger α–Schnitt von $\tilde{A}$.*

Der Abschluß der Menge von Punkten, für die der Zugehörigkeitsgrad einer auf $\mathbb{R}$ definierten Fuzzy–Menge $\widetilde{A}$ größer als 0 ist, heißt Träger von $\widetilde{A}$

Definition 3.8 *Sei $\widetilde{A} \subset \mathbb{R}$ eine Fuzzy–Menge. Der Abschluß von $\widetilde{A}_{\bar{0}}$ heißt der Träger von $\widetilde{A}$.*

Beispiel 3.4 *Für eine Dreiecks–Menge (l, m, r) ist der Träger das Intervall $[l, r]$.*

3.3 Operationen auf Fuzzy–Mengen

Genau wie für crispe Mengen gibt es auch für Fuzzy–Mengen verschiedene Operationen wie Vereinigungs–, Durchschnitts– und Komplement–Bildung. Da eine Fuzzy–Menge mit Hilfe ihrer Zugehörigkeitsfunktion definiert wird, werden die Zugehörigkeitsgrade der neuen Menge direkt aus den Zugehörigkeitsgraden der alten Mengen berechnet:

Definition 3.9 *Seien $\widetilde{A} \subset X$ und $\widetilde{B} \subset X$ Fuzzy–Mengen mit Zugehörigkeitsfunktionen $\mu_{\widetilde{A}}$ und $\mu_{\widetilde{B}}$. Die Zugehörigkeitsfunktionen der Vereinigungs–Menge $\widetilde{A} \cup \widetilde{B}$, der*

Durchschnitts–Menge $\tilde{A} \cap \tilde{B}$ *und der Komplement–Menge* $\tilde{A}^c$ *ergeben sich wie folgt:*

$$\mu_{\widetilde{A}\cup\widetilde{B}}(x) = \max\left\{\mu_{\widetilde{A}}(x), \mu_{\widetilde{B}}(x)\right\} \qquad \textit{(Fuzzy–Vereinigung)}$$

$$\mu_{\widetilde{A}\cap\widetilde{B}}(x) = \min\left\{\mu_{\widetilde{A}}(x), \mu_{\widetilde{B}}(x)\right\} \qquad \textit{(Fuzzy–Durchschnitt)}$$

$$\mu_{\widetilde{A}^c}(x) = 1 - \mu_{\widetilde{A}}(x) \qquad \textit{(Fuzzy–Komplement)}$$

Diese Definitionen entsprechen genau der Berechnung der neuen Indikatorfunktion bei crispen Mengen. Die so berechneten Operatoren für Fuzzy–Mengen heißen auch *triviale Operatoren* (Abb. 3.2).

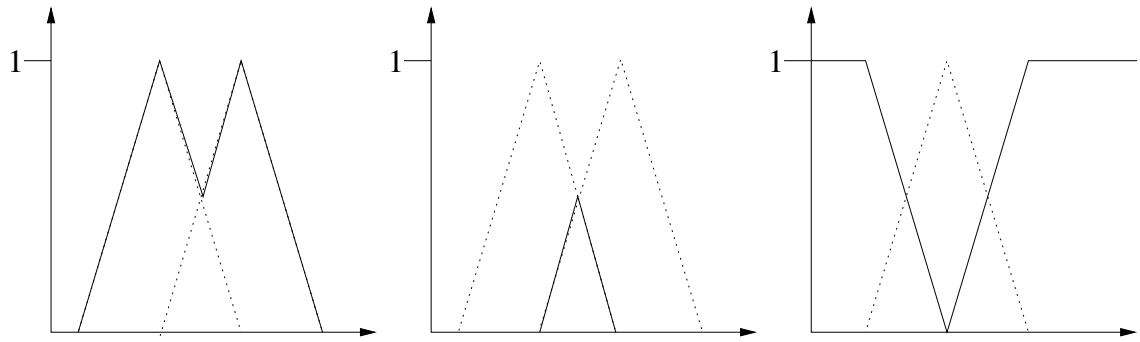

Abbildung 3.2: Vereinigung, Durchschnitt und Komplement von Fuzzy–Mengen

Im Gegensatz zu crispen Mengen ist es jedoch bei Fuzzy–Mengen nicht verbindlich festgelegt, wie die Vereinigung, der Durchschnitt und das Komplement definiert werden müssen. Aus diesem Grund gibt es allgemeine Definitionen, die die Mindestanforderungen an einen Operator festlegen, der für eine dieser Verknüpfungen verwendet wird. Damit werden Klassen von Operatoren beschrieben, aus denen jeweils ein Vertreter nach Bedarf ausgewählt wird:

Definition 3.10
Eine Funktion $c : [0,1] \longrightarrow [0,1]$ *heißt Fuzzy–Komplement, wenn gilt:*

- $c(0) = 1$, $c(1) = 0$ *(Randbedingungen)*
- *c ist monoton fallend*

Eine Funktion $t : [0,1]^2 \longrightarrow [0,1]$ *heißt Fuzzy–Durchschnitt oder T–Norm, wenn gilt:*

- $t(0,0) = 0,\ t(a,1) = t(1,a) = a,\ t(1,1) = 1$ *(Randbedingungen)*
- *t ist kommutativ*
- *t ist monoton:* $a \leq b \Rightarrow t(a,c) \leq t(b,c)$
- *t ist assoziativ:* $t\big(a, t(b,c)\big) = t\big(t(a,b), c\big)$.

Eine Funktion $s : [0,1]^2 \longrightarrow [0,1]$ *heißt Fuzzy–Vereinigung oder T–Conorm, wenn gilt:*

- $s(0,0) = 0,\ s(a,0) = s(0,a) = a,\ s(1,1) = 1$ *(Randbedingungen)*
- *s ist kommutativ*
- *s ist monoton und assoziativ*

Aus den Randbedingungen ergibt sich ein Verhalten, das an Stellen mit Zugehörigkeitsgrad 0 oder 1 genau den entsprechenden Operationen für crispe Mengen entspricht. Somit sind analog zur Zugehörigkeitsfunktion auch diese Operatoren immer Erweiterungen der entsprechenden klassischen Operatoren. Assoziativität und Kommutativität sichern, daß die Vereinigung oder der Durchschnitt mehrerer Mengen nicht von der Reihenfolge der Berechnungen abhängt. Außerdem ist es dadurch leicht möglich, das Ergebnis für mehrere Mengen rekursiv zu berechnen.

Diese Definitionen legen nur die Mindestanforderungen an einen Operator fest. Weitere sinnvolle Eigenschaften wie Stetigkeit und das bei gewöhnlichen Mengen selbstverständliche *Idempotenzgesetz* werden hier nicht gefordert. Das bedeutet, die Gleichungen $\widetilde{A} \cup \widetilde{A} = \widetilde{A}$ und $\widetilde{A} \cap \widetilde{A} = \widetilde{A}$ sind für Fuzzy–Mengen i.A. nicht richtig. Ebenso ist für ein Fuzzy–Komplement nicht per Definition gefordert, daß es *involutiv* ist, d.h. $c(c(a)) = a$ muß nicht gelten. I.A. ist also das Komplement vom Komplement einer Fuzzy–Menge nicht wieder die ursprüngliche Fuzzy–Menge.

Beispiel 3.5 *Bounded Difference und Bounded Sum:*

$$t(a,b) = \max\big\{0, a+b-1\big\}$$
$$s(a,b) = \min\big\{1, a+b\big\}$$

Die Werte

$$t(0.4, 0.4) = \max\{0, 0.4 + 0.4 - 1\} = 0 \neq 0.4$$

und

$$s(0.4, 0.4) = \min\{1, 0.4 + 0.4\} = 0.8 \neq 0.4$$

zeigen, daß diese Operatoren das Idempotenzgesetz nicht erfüllen.

Beispiel 3.6
$c(a) = \frac{1}{2} \cdot (1 + \cos \pi \cdot a)$ *ist ein Fuzzy–Komplement, welches nicht involutiv ist:*

$$c\big(c(\tfrac{1}{3})\big) = c\big(\tfrac{1}{2} \cdot (1 + \cos \tfrac{\pi}{3})\big) = c\big(0.75\big) = \tfrac{1}{2} \cdot \big(1 + \cos \pi \cdot 0.75\big) = 0.15 \neq \tfrac{1}{3}$$

Definiert man das Fuzzy–Komplement gemäß $c(a) = 1 - a$, so läßt sich stets zu einer T–Norm die *duale T–Conorm* berechnen bzw. zu einer T–Conorm die *duale T–Norm*:

Definition 3.11
Sei t eine T–Norm. Die zu t duale T–Conorm s_d *ist definiert durch:*

$$s_d(a, b) = 1 - t(1 - a, 1 - b)$$

Sei s eine T–Conorm. Die zu s duale T–Norm t_d *ist definiert durch:*

$$t_d(a, b) = 1 - s(1 - a, 1 - b)$$

Neben den trivialen Operatoren gibt es noch eine ganze Reihe anderer Operatoren. Einige davon verwenden einen Parameter, mit dem sich das Verhalten verändern läßt.

Beispiel 3.7 *Yagers Operatoren:*

$$c(a) = (1 - a^w)^{\frac{1}{w}}$$
$$t(a, b) = 1 - min\big\{1, \big((1 - a)^w + (1 - b)^w\big)^{\frac{1}{w}}\big\}$$
$$s(a, b) = min\big\{1, (a^w + b^w)^{\frac{1}{w}}\big\}$$

mit $w \in]0, \infty[$.

Dabei konvergiert t für $w \to \infty$ gegen das gewöhnliche Minimum und s gegen das gewöhnliche Maximum. Weitere Operatoren finden sich z.B. in [Tilli, 1993].

Die von crispen Mengen $A \subset X$ bekannten Gesetze vom Widerspruch ($A \cap A^c = \emptyset$) und vom ausgeschlossenen Dritten ($A \cup A^c = X$) gelten für Fuzzy–Mengen nicht: Die Zugehörigkeitsfunktion der leeren Menge ist konstant 0, die der ganzen Menge ist konstant 1.

Beispiel 3.8 *Sei $\widetilde{A} \subset X$ eine Fuzzy–Menge. Sei $x \in X$ gegeben mit $\mu_{\widetilde{A}}(x) = 0.2$. Dann gilt mit den trivialen Operatoren:*

$$\begin{aligned}\mu_{\widetilde{A}\cap\widetilde{A}^c}(x) &= \min\left\{\mu_{\widetilde{A}}(x), \mu_{\widetilde{A}^c}(x)\right\} = \min\left\{0.2, 1-\mu_{\widetilde{A}}(x)\right\} \\ &= \min\left\{0.2, 1-0.2\right\} = \min\left\{0.2, 0.8\right\} = 0.2 \neq 0\end{aligned}$$

$$\begin{aligned}\mu_{\widetilde{A}\cup\widetilde{A}^c}(x) &= \max\left\{\mu_{\widetilde{A}}(x), \mu_{\widetilde{A}^c}(x)\right\} = \max\left\{0.2, 1-\mu_{\widetilde{A}}(x)\right\} \\ &= \max\left\{0.2, 1-0.2\right\} = \max\left\{0.2, 0.8\right\} = 0.8 \neq 1\end{aligned}$$

Jeder T–Norm bzw. T–Conorm läßt sich mit den trivialen Operatoren und den Operatoren aus Beispiel 3.9 nach oben bzw. unten abschätzen:

Beispiel 3.9

$$t_w(a,b) = \begin{cases} a: & b = 1 \\ b: & a = 1 \\ 0: & sonst \end{cases}$$

$$s_w(a,b) = \begin{cases} a: & b = 0 \\ b: & a = 0 \\ 1: & sonst \end{cases}$$

definieren eine T–Norm bzw. T–Conorm.

Satz 3.1 *Seien t und s eine beliebige T–Norm bzw. T–Conorm und t_w und s_w die in Beispiel 3.9 definierten Operatoren. Dann gilt für alle $a, b \in [0,1]$:*

$$t_w(a,b) \leq t(a,b) \leq \min\{a,b\}$$

$$s_w(a,b) \geq s(a,b) \geq \max\{a,b\}$$

Beweis 3.1 *Der Beweis erfolgt durch einfaches Nachrechnen unter Ausnutzung der Definitionen von T–Norm bzw. T–Conorm.* □

Aufgrund dieser Eigenschaften wird das Minimum auch als optimistischer Durchschnittsoperator bezeichnet und analog das Maximum als pessimistischer Vereinigungsoperator.

3.4 Fuzzy–Relationen

Fuzzy–Relationen sind genau so eine Erweiterung von "gewöhnlichen" Relationen, wie Fuzzy–Mengen eine Erweiterung von "gewöhnlichen" Mengen sind. Eine *Relation* auf $X \times Y$ ist bekanntlich eine Teilmenge von $X \times Y$.

Definition 3.12 *Seien X und Y Mengen. Eine Fuzzy–Relation auf $X \times Y$ ist eine Fuzzy–Menge $\widetilde{R} \subset X \times Y$. $\mu_{\widetilde{R}}(x_i, y_j)$ gibt dabei an, zu welchem Grad (x_i, y_j) die Relation erfüllt.*

Durch diese Definition ist eine wesentlich feinere Abstufung möglich als bei der crispen Relation. So läßt sich z.B. bei der Kleiner–Relation "$<$" angeben, zu welchem Grad x_i kleiner ist als y_j. Mit $\mu_<(1, 1.1) = 0.1$ und $\mu_<(1, 1000) = 1$ wird verdeutlicht, daß 1 nur wenig kleiner ist als 1.1, aber wesentlich kleiner als 1000. Endliche Relationen werden häufig in Matrix–Form dargestellt. An der Stelle (i, j) der Matrix wird dazu der Zugehörigkeitsgrad $\mu_{\widetilde{R}}(x_i, y_j)$ eingetragen:

Beispiel 3.10

$$\widetilde{R} := \begin{array}{c} \\ x_1 \\ x_2 \end{array} \begin{array}{c} \begin{array}{ccc} y_1 & y_2 & y_3 \end{array} \\ \begin{pmatrix} 0.6 & 0.2 & 0.7 \\ 0.5 & 0.4 & 0.8 \end{pmatrix} \end{array}$$

Der Eintrag 0.6 an der Stelle $(1, 1)$ der Matrix gibt dabei an, daß $\mu_{\widetilde{R}}(x_1, y_1) = 0.6$ gilt.

Da Fuzzy–Relationen als Fuzzy–Mengen definiert werden, sind für sie dieselben Operationen wie für Fuzzy–Mengen definiert. Um zwei Fuzzy–Relationen mit den Operatoren für Fuzzy–Mengen zu kombinieren, müssen sie auf dem gleichen Grundraum $X \times Y$ definiert sein. Um zwei Relationen auf verschiedenen Grundräumen zu kombinieren, gibt es spezielle Berechnungsvorschriften für die Komposition von Relationen. Am gebräuchlichsten sind die *Max–Min–Komposition* und die *Max–Produkt–Komposition*:

Definition 3.13 *Seien X, Y und Z Mengen. Seien $\widetilde{R}_1 \subset X \times Y$ und $\widetilde{R}_2 \subset Y \times Z$ Fuzzy–Relationen.*

Die Max–Min–Komposition $\widetilde{R}_2 \circ_{MM} \widetilde{R}_1$ von $\widetilde{R}_1$ und $\widetilde{R}_2$ ist dann eine Fuzzy–Relation auf $X \times Z$. Ihre Zugehörigkeitsfunktion $\mu_{\widetilde{R}_2 \circ_{MM} \widetilde{R}_1}$ wird definiert durch:

$$\mu_{\widetilde{R}_2 \circ_{MM} \widetilde{R}_1}(x_i, z_j) = \max_{y \in Y} \Big\{ \min \big\{ \mu_{\widetilde{R}_1}(x_i, y), \mu_{\widetilde{R}_2}(y, z_j) \big\} \Big\}$$

Die Max–Produkt–Komposition $\widetilde{R}_2 \circ_{MP} \widetilde{R}_1$ von $\widetilde{R}_1$ und $\widetilde{R}_2$ ist ebenfalls eine Fuzzy–Relation auf $X \times Z$. Ihre Zugehörigkeitsfunktion $\mu_{\widetilde{R}_2 \circ_{MP} \widetilde{R}_1}$ wird definiert durch:

$$\mu_{\widetilde{R}_2 \circ_{MP} \widetilde{R}_1}(x_i, z_j) = \max_{y \in Y} \Big\{ \mu_{\widetilde{R}_1}(x_i, y) \cdot \mu_{\widetilde{R}_2}(y, z_j) \Big\}$$

Generell ergeben sich bei der Max–Min–Komposition Werte, die gleich groß oder geringfügig größer als die Werte der Max–Produkt–Komposition sind. So wie das Minimum der optimistische Vereinigungsoperator ist, ist die Max–Min–Komposition die optimistische Komposition.

Beispiel 3.11 *Seien folgende Fuzzy–Relationen $\widetilde{R}_1 \subset X \times Y$ und $\widetilde{R}_2 \subset Y \times Z$ gegeben:*

$$\widetilde{R}_1 := \begin{array}{c} \\ x_1 \\ x_2 \end{array} \begin{array}{c} \begin{array}{ccc} y_1 & y_2 & y_3 \end{array} \\ \begin{pmatrix} 0.4 & 0.3 & 0.1 \\ 0.8 & 0.3 & 0.9 \end{pmatrix} \end{array}$$

$$\widetilde{R}_2 := \begin{array}{c} \\ y_1 \\ y_2 \\ y_3 \end{array} \begin{array}{c} \begin{array}{cc} z_1 & z_2 \end{array} \\ \begin{pmatrix} 0.7 & 0.3 \\ 0.4 & 0.6 \\ 0.5 & 0.2 \end{pmatrix} \end{array}$$

Für die Max–Min–Komposition $\widetilde{R}_2 \circ_{MM} \widetilde{R}_1$ und die Max–Produkt–Komposition $\widetilde{R}_2 \circ_{MP} \widetilde{R}_1$ gilt:

$$\mu_{\widetilde{R}_2 \circ_{MM} \widetilde{R}_1}(x_1, z_1) = \max \Big\{ \min \big\{0.4, 0.7\big\}, \min \big\{0.3, 0.4\big\}, \min \big\{0.1, 0.5\big\} \Big\} =$$

$$\max \Big\{0.4, 0.3, 0.1\Big\} = 0.4$$

bzw.

$$\mu_{\widetilde{R}_2 \circ_{MP} \widetilde{R}_1}(x_1, z_1) = \max\left\{0.4 \cdot 0.7, 0.3 \cdot 0.4, 0.1 \cdot 0.5\right\} =$$

$$\max\left\{0.28, 0.12, 0.05\right\} = 0.28$$

Die weiteren Werte werden analog berechnet und es ergibt sich:

$$\widetilde{R}_2 \circ_{MM} \widetilde{R}_1 = \begin{array}{c} \\ x_1 \\ x_2 \end{array} \begin{array}{c} \begin{array}{cc} z_1 & z_2 \end{array} \\ \begin{pmatrix} 0.4 & 0.3 \\ 0.7 & 0.3 \end{pmatrix} \end{array}$$

$$\widetilde{R}_2 \circ_{MP} \widetilde{R}_1 = \begin{array}{c} \\ x_1 \\ x_2 \end{array} \begin{array}{c} \begin{array}{cc} z_1 & z_2 \end{array} \\ \begin{pmatrix} 0.28 & 0.18 \\ 0.56 & 0.24 \end{pmatrix} \end{array}$$

3.5 Fuzzy–Arithmetik

Neben Fuzzy–Mengen und Fuzzy–Relationen als Erweiterung der klassischen Mengen bzw. klassischen Relationen werden Fuzzy–Zahlen als Erweiterung der reellen Zahlen definiert.

Fuzzy–Zahlen sind Fuzzy–Mengen mit speziellen Eigenschaften. Um mit Fuzzy–Zahlen wie "ungefähr vier" zu rechnen, wird das *Extensionsprinzip* verwendet. Damit werden mathematische Funktionen, wie die einfache Addition oder $f(x) = \frac{1}{x}$, für Fuzzy–Zahlen definiert. Der Funktionswert einer auf diese Weise definierten Funktion ist wieder eine Fuzzy–Zahl. Fuzzy–Arithmetik spielt bei den weiteren Betrachtungen dieser Arbeit keine Rolle, dennoch soll hier eine kurze Einführung erfolgen.

Definition 3.14 *Sei $\widetilde{A} \subset X$ eine Fuzzy–Menge. Die Höhe von $\widetilde{A}$ ist definiert als:*

$$hgt(\widetilde{A}) = \sup_{x \in X} \mu_{\widetilde{A}}(x)$$

$\widetilde{A}$ heißt normal , falls gilt: $hgt(\widetilde{A}) = 1$.

Beispiel 3.12 *Eine Dreiecks–Mengen $\widetilde{A} = (l, m, r)$ ist normal, da für ihren Modalwert m gilt:*

$$\mu_{\widetilde{A}}(m) = 1$$

Definition 3.15 *Sei $\widetilde{A} \subset X$ eine Fuzzy–Menge. $\widetilde{A}$ heißt konvex , falls für alle $x_1, x_2 \in X$ und für alle $\lambda \in [0, 1]$ gilt:*

$$\mu_{\widetilde{A}}\big(\lambda \cdot x_1 + (1 - \lambda) \cdot x_2\big) \geq \min\left\{\mu_{\widetilde{A}}(x_1), \mu_{\widetilde{A}}(x_2)\right\}$$

Beispiel 3.13 *Abb. 3.3 verdeutlicht den Unterschied zwischen konvexen und nicht konvexen Fuzzy–Mengen.*

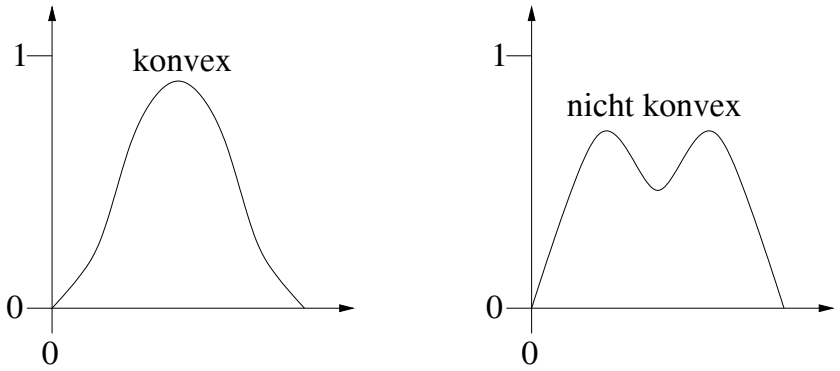

Abbildung 3.3: Konvexe Fuzzy–Menge und nicht konvexe Fuzzy–Menge

Definition 3.16 *Eine Fuzzy–Zahl ist eine Fuzzy–Menge $\widetilde{A} \subset \mathbb{R}$, für die gilt:*

1. *$\widetilde{A}$ ist normal und konvex*
2. *Die Zugehörigkeitsfunktion $\mu_{\widetilde{A}}$ von $\widetilde{A}$ ist stückweise stetig*
3. *Es gibt genau ein $x_0 \in \mathbb{R}$ mit $\mu_{\widetilde{A}}(x_0) = 1$*

Typische Fuzzy–Zahlen sind Dreiecksmengen und Gauß–Mengen. Trapezmengen sind keine Fuzzy–Zahlen, da es mehr als einen Punkt gibt mit Zugehörigkeitsgrad 1.

Beispiel 3.14

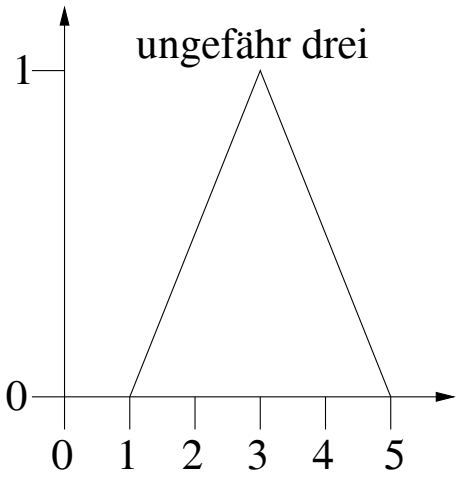

Abbildung 3.4: Dreiecks–Zahl "ungefähr drei"

Um mit Fuzzy–Zahlen Funktionen zu berechnen, wird das *Extensionsprinzip* benötigt. Das Ziel ist es, eine reelle Funktion

$$f : X \longrightarrow Y \text{ mit } X, Y \subset \mathbb{R}$$

oder

$$f : X_1 \times \ldots \times X_n \longrightarrow Y \text{ mit } X_1 \times \ldots \times X_n \subset \mathbb{R}^n, Y \subset \mathbb{R}$$

zur Funktion $\widetilde{f}$ für Fuzzy–Zahlen zu erweitern.

D.h. es soll $\widetilde{f}(\widetilde{A}) = \widetilde{B}$ mit Hilfe von f berechnet werden, wobei $\widetilde{A}$ eine Fuzzy–Zahl auf X bzw. $X_1 \times \ldots \times X_n$ ist und $\widetilde{B}$ eine Fuzzy–Zahl auf Y.

Die naheliegende Festlegung

$\mu_{\widetilde{B}}(y) := \mu_{\widetilde{A}}(x)$, falls $f(x) = y$ gilt,

scheitert: falls f nicht injektiv ist, gibt es mehrere Werte zu Auswahl. Dieses Problem wird durch das Extensionsprinzip gelöst:

Definition 3.17 *Extensionsprinzip*

Sei $f : X_1 \times \ldots \times X_n \longrightarrow Y$, mit $X_1 \times \ldots \times X_n \subset \mathbb{R}^n, Y \subset \mathbb{R}$, eine Funktion.
Seien $\widetilde{A}_i \subset X_i$, $i = 1, \ldots, n$, Fuzzy–Zahlen.

Dann wird die Erweiterung $\widetilde{f}$ von f auf Fuzzy–Zahlen mit dem Extensionsprinzip wie folgt definiert:

$\widetilde{f}(\widetilde{A}_1, \ldots, \widetilde{A}_n) = \widetilde{B}$ *mit:*

$$\mu_{\widetilde{B}}(y) = \begin{cases} \sup\limits_{(x_1,\ldots,x_n)|y=f(x_1,\ldots,x_n)} \left(\min\left\{\mu_{\widetilde{A}_1}(x_1), \ldots, \mu_{\widetilde{A}_n}(x_n)\right\}\right) : \textit{falls } f^{-1}(y) \neq \emptyset \\ 0 : \textit{sonst} \end{cases}$$

Im eindimensionalen Fall ($f : X \longrightarrow Y$) entfällt die Bestimmung der Minima. Dadurch vereinfacht sich die Berechnung von $\widetilde{B}$:

Definition 3.18 *Extensionsprinzip (eindimensional)*

Sei $f : X \longrightarrow Y$, mit $X, Y \subset \mathbb{R}$, eine Funktion.
Sei $\widetilde{A} \subset X$ eine Fuzzy–Zahl.

Dann wird die Erweiterung $\widetilde{f}$ von f auf Fuzzy–Zahlen mit dem Extensionsprinzip wie folgt definiert:

$\widetilde{f}(\widetilde{A}) = \widetilde{B}$ *mit:*

$$\mu_{\widetilde{B}}(y) = \begin{cases} \sup\limits_{x|y=f(x)} \left(\mu_{\widetilde{A}}(x)\right) : \textit{falls } f^{-1}(y) \neq \emptyset \\ 0 : \textit{sonst} \end{cases}$$

Mit Hilfe des Extensionsprinzips läßt sich jede reelle Funktion auf Fuzzy–Zahlen fortsetzen. Insbesondere ist für $\widetilde{f}(\widetilde{A}) = \widetilde{B}$ der Modalwert von $\widetilde{B}$ immer der mit der ursprünglichen Funktion f berechnete Funktionswert des Modalwerts von $\widetilde{A}$:

Satz 3.2
Sei $f : X \longrightarrow Y$, $X, Y \subset \mathbb{R}$, eine Funktion,
sei $\widetilde{f}$ die mit dem Extensionsprinzip berechnete Erweiterung von f.
Seien $\widetilde{A} \subset X, \widetilde{B} \subset Y$ Fuzzy–Zahlen mit $\widetilde{f}(\widetilde{A}) = \widetilde{B}$ sowie x_0 der Modalwert von $\widetilde{A}$.

Dann ist $y_o = f(x_0)$ *der Modalwert von* $\widetilde{B}$.

Beweis 3.2 *Es gilt nach dem Extensionsprinzip:*

$$\mu_{\widetilde{B}}(y_0) = \begin{cases} \sup\limits_{x|y_0=f(x)} \left(\mu_{\widetilde{A}}(x)\right) : \textit{falls } f^{-1}(y_0) \neq \emptyset \\ 0 : \textit{sonst} \end{cases}$$

Wegen $f(x_0) = y_0$ *und* $\mu_{\widetilde{A}}(x_0) = 1$ *muß auch* $\mu_{\widetilde{B}}(y_0) = 1$ *gelten. D.h.* y_0 *ist der Modalwert von* $\widetilde{B}$. □

Beispiel 3.15
Sei $f : \mathbb{R} \longrightarrow \mathbb{R}$ *als* $f(x) = x + 1$ *gegeben,*
sei $\widetilde{A}$ *die Dreieckszahl "ungefähr drei"* $(1, 3, 5)$.

Dann ist nach dem Extensionsprinzip $\widetilde{f}(\widetilde{A}) = \widetilde{B}$ *mit*
$\widetilde{B}$ *der Dreieckszahl "ungefähr vier"* $(2, 4, 6)$*:*

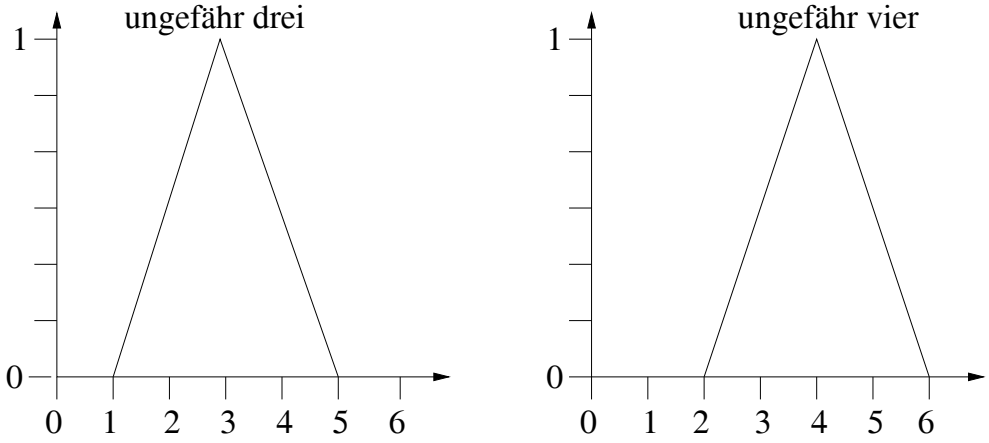

Abbildung 3.5: Ergebnis des Extensionsprinzips

Berechnung einzelner Werte:

$$\mu_{\widetilde{B}}(3) = \begin{cases} \sup\limits_{x|3=f(x)} \left(\mu_{\widetilde{A}}(x)\right) : \textit{falls } f^{-1}(3) \neq \emptyset \\ 0 : \textit{sonst} \end{cases}$$

$$= \mu_{\widetilde{A}}(2) = 0.5$$

Da nur $f(2) = 3$ *gilt.*

$$\mu_{\widetilde{B}}(4) = \begin{cases} \sup\limits_{x|4=f(x)} \left(\mu_{\widetilde{A}}(x)\right) : falls\ f^{-1}(4) \neq \emptyset \\ 0 : sonst \end{cases}$$

$$= \mu_{\widetilde{A}}(3) = 1$$

Da nur $f(3) = 4$ gilt.

$$\mu_{\widetilde{B}}(5) = \begin{cases} \sup\limits_{x|5=f(x)} \left(\mu_{\widetilde{A}}(x)\right) : falls\ f^{-1}(5) \neq \emptyset \\ 0 : sonst \end{cases}$$

$$= \mu_{\widetilde{A}}(4) = 0.5$$

Da nur $f(4) = 5$ gilt.

Die weiteren Werte werden analog berechnet.

Weitere Eigenschaften des Extensionsprinzips und weitere Beispiele werden z.B. in [Feuring, 1996] beschrieben.

3.6 Klassische Aussagenlogik

Die klassische *Aussagenlogik* beschäftigt sich mit Aussagen, die entweder wahr oder falsch sind und Verknüpfungen dieser Aussagen zu neuen Aussagen. Der *Wahrheitswert* einer Aussage wird mit der *Wahrheitswertefunktion* δ berechnet, die nur die Werte 1 für *wahr* und 0 für *falsch* annimmt. Die Variablen einer Aussage heißen *Aussagenvariablen* und können ebenfalls nur die Werte 0 und 1 annehmen.

Definition 3.19 *Eine Aussage ist entweder eine Aussagenvariable x oder eine Verknüpfung von Aussagen. Zur Verknüpfung stehen die bekannten aussagenlogischen Funktionen zur Verfügung:*

- $\neg$ *(Negation)*
- $\wedge$ *(logisches und)*
- $\vee$ *(logisches oder)*
- $\longrightarrow$ *(Implikation)*
- $\longleftrightarrow$ *(Bijunktion)*

Aussagen können demnach beliebig als Bestandteile neuer Aussagen verwendet werden:

Beispiel 3.16
Die Aussagen $A_1(x_1, x_2) = x_1 \vee x_2$ *und* $A_2(x_3, x_4) = x_3 \vee x_4$

werden mit $\wedge$ *zur neuen Aussage*

$A(x_1, x_2, x_3, x_4) = (x_1 \vee x_2) \wedge (x_3 \vee x_4)$

zusammengesetzt.

Der Wahrheitswert wird bei komplexeren Aussagen schrittweise von innen nach außen berechnet, indem jeweils zuerst die Wahrheitswerte der innersten Aussagen berechnet werden, dann die Wahrheitswerte der daraus zusammengesetzten Aussagen usw. Ausgangspunkt für die Berechnung ist in jedem Fall eine *Belegung* der Aussagenvariablen.

Definition 3.20 *Eine aussagenlogische Belegung ist eine Abbildung von der Menge der Aussagenvariablen in die Menge der Wahrheitswerte* $\{0, 1\}$.

Bei einer Belegung wird also für jede Variable festgelegt, welchen Wert sie hat. Dies entspricht dem Einsetzen von Zahlen für die Variablen $x_1, \ldots, x_n$ in eine mathematische Funktion $f(x_1, \ldots, x_n)$. Der Wahrheitswert einer Aussage hängt von der Belegung ab und wird entsprechend der Definition der aussagenlogischen Funktionen berechnet:

Beispiel 3.17 *In Beispiel 3.16 sieht das für* $x_1 = 1$, $x_2 = 0$, $x_3 = 0$ *und* $x_4 = 1$ *wie folgt aus:*

$$\delta\big(A(1,0,0,1)\big) = \delta\big((1 \vee 0) \wedge (0 \vee 1)\big) =$$

$$\delta\big(\delta(1 \vee 0) \wedge \delta(0 \vee 1)\big) = \delta\big(1 \wedge 1\big) = 1$$

Definition 3.21
Eine Aussage $A(x_1, \ldots, x_n)$ *heißt erfüllbar oder konsistent, wenn es eine Belegung für* $x_1 \ldots, x_n$ *gibt mit* $\delta\big(A(x_1, \ldots, x_n)\big) = 1$.

Eine Aussage $A(x_1, \ldots, x_n)$ *heißt nicht erfüllbar oder inkonsistent, wenn es keine Belegung für* $x_1, \ldots, x_n$ *gibt mit* $\delta\big(A(x_1, \ldots, x_n)\big) = 1$.

Eine Aussage $A(x_1, \ldots, x_n)$ *heißt allgemeingültig oder Tautologie, wenn für jede Belegung von* $x_1, \ldots, x_n$ $\delta\big(A(x_1, \ldots, x_n)\big) = 1$ *gilt.*

Beispiel 3.18

- $x_1 \vee x_1$ *ist konsistent*
- $x_1 \wedge \neg x_1$ *ist inkonsistent*
- $x_1 \vee \neg x_1$ *ist eine Tautologie*

Es gibt mehr Verknüpfungsoperatoren als zur Darstellung aller Aussagen notwendig sind. Um jede mögliche Aussage darzustellen würden z.B. $\neg$ und $\vee$ genügen. Alle weiteren Verknüpfungen lassen sich aus diesen beiden Verknüpfungen zusammensetzen. Z.B. hat $\neg x_1 \vee x_2$ für jede Belegung von x_1 und x_2 den gleichen Wahrheitswert wie $x_1 \longrightarrow x_2$, d.h. die Aussagen sind gleichwertig oder *aussagenlogisch äquivalent.*

Verknüpfungs–Symbole wie $\wedge$ oder $\longrightarrow$ dienen der besseren Darstellung und dem Verständnis von Aussagen. Der Wahrheitswert von $x_1 \longrightarrow x_2$ entspricht z.B. der intuitiven Bedeutung der Folgerung "wenn x_1 gilt, dann gilt auch x_2"; ebenso entspricht der Wahrheitswert von $x_1 \wedge x_2$ der intuitiven Bedeutung von "x_1 und x_2". Wertetabellen für die verschiedenen Verknüpfungen und weitere äquivalente Formeln finden sich z.B. in [Heinemann, 1992].

3.7 Aussagenlogischer Schluß

Das Ziel beim aussagenlogischen Schluß ist es, aus Aussagen, deren Gültigkeit bereits bekannt ist, neue gültige Aussagen herzuleiten. Genauer bedeutet das folgendes:

Definition 3.22 *Seien $A_1, \ldots, A_m$ und B Aussagen.*
Wenn für jede Belegung von $x_1, \ldots, x_n$ aus

$$\delta\big(A_1(x_1, \ldots, x_n)\big) = \ldots = \delta\big(A_m(x_1, \ldots, x_n)\big) = 1$$

folgt, daß auch

$$\delta\big(B(x_1, \ldots, x_n)\big) = 1$$

gilt, dann heißt B aussagenlogischer Schluß aus $A_1, \ldots, A_m$.

Schreibweise: $A_1, \ldots, A_m \Longrightarrow B$.

Diese Definition ist gleichbedeutend damit, daß $(A_1 \wedge \ldots \wedge A_m) \longrightarrow B$ allgemeingültig ist. Ein aussagenlogischer Schluß bedeutet also, daß immer, wenn die Formeln der Voraussetzung wahr sind, auch die gefolgerte Formel wahr ist. Aussagenlogische Schlüsse werden verwendet, um aus als gültig bekannten Formeln neue Formeln herzuleiten. Bekannte Schlußregeln sind z.B.:

Definition 3.23
Modus ponens (Abtrennungsregel): $a, a \longrightarrow b \Longrightarrow b$

Modus tollens (Widerspruchsregel): $\neg b, a \longrightarrow b \Longrightarrow \neg a$

Modus barbara (Kettenschluß): $a \longrightarrow b, b \longrightarrow c \Longrightarrow a \rightarrow c$

3.8 Fuzzy–Aussagenlogik

Ebenso wie Fuzzy–Mengen eine Erweiterung von "gewöhnlichen" Mengen sind, ist die Fuzzy–Aussagenlogik eine Erweiterung der "gewöhnlichen" Aussagenlogik. Analog zur Erweiterung des Bildbereichs der Indikatorfunktion $\mathcal{X}$ wird hier zur Definition der fuzzy–Wahrheitswertefunktion $\widetilde{\delta}$ der Bildbereich der klassischen Wahrheitswertefunktion δ auf das ganze Intervall $[0, 1]$ erweitert. Dadurch lassen sich Aussagen genauer bewerten als in der klassischen Logik. Je nach Betrachter sind unterschiedliche Ansichten und Bewertungen möglich, wie folgende Aussage zeigt:

Beispiel 3.19 *Betrachten wir folgende umgangssprachliche Aussage: "Die Mathe–Klausur war schwer." Dieser Aussage wird ein Schüler oder Student entsprechend seiner Kenntnisse und Vorbereitung mehr oder weniger stark zustimmen. Diese Unterschiede bei der realen Bewertung einer Aussage können in der Fuzzy–Aussagenlogik wesentlich besser berücksichtigt werden als in der klassischen Aussagenlogik, die nur zwei Wahrheitswerte kennt.*

In der Fuzzy–Aussagenlogik werden die gleichen Verknüpfungs–Symbole wie in der klassischen Aussagenlogik definiert. Der Wertebereich für die Aussagenvariablen und der Bildbereich der Wahrheitswertefunktion ist das Intervall $[0, 1]$. Eine Aussage kann also jetzt nicht mehr nur wahr oder falsch sein, sondern sie hat einen bestimmten *Wahrheitsgrad* zwischen 0 und 1. Dabei bedeutet 1, daß die Aussage wahr ist, 0 bedeutet, daß die Aussage falsch ist.

Definition 3.24 *Eine fuzzy–aussagenlogische Belegung ist eine Abbildung von der Menge der Aussagenvariablen in das Intervall* $[0, 1]$.

Welchen Wahrheitsgrad eine Aussagenvariable in der Fuzzy–Aussagenlogik bekommt, muß jeweils im Einzelfall festgelegt werden. Dabei spielen auch subjektive Meinungen eine Rolle, vgl. Beispiel 3.19.

Analog zur klassischen Aussagenlogik lassen sich Aussagen beliebig mit den verschiedenen Verknüpfungen zu neuen Aussagen kombinieren. Die Berechnung des Wahrheitswertes erfolgt dabei analog zur klassischen Aussagenlogik von innen nach außen. Folgende fuzzylogische Operationen stehen zur Verfügung:

Definition 3.25

- *Negation:* $\neg$ *mit* $\widetilde{\delta}(\neg x) = 1 - \widetilde{\delta}(x)$
- *Konjunktion:* $\wedge$ *mit* $\widetilde{\delta}(x_1 \wedge x_2) = \min\left\{\widetilde{\delta}(x_1), \widetilde{\delta}(x_2)\right\}$
- *Disjunktion:* $\vee$ *mit* $\widetilde{\delta}(x_1 \vee x_2) = \max\left\{\widetilde{\delta}(x_1), \widetilde{\delta}(x_2)\right\}$
- *Implikation:* $\longrightarrow$ *mit* $\widetilde{\delta}(x_1 \longrightarrow x_2) = \min\left\{1, 1 - \widetilde{\delta}(x_1) + \widetilde{\delta}(x_2)\right\}$
- *Bijunktion:* $\longleftrightarrow$ *mit* $\widetilde{\delta}(x_1 \leftrightarrow x_2) = 1 - \left(\widetilde{\delta}(x_1) - \widetilde{\delta}(x_2)\right)$

Beispiel 3.20 *Betrachten wir folgende zusammengesetzte Aussage A:*
"Diplom–Mathematiker mit Kenntnissen in der Informatik und der Betriebswirtschaftslehre haben gute Berufsaussichten in der Entwicklung oder im Management." Für Diplom–Mathematiker S. werden folgende Einzel–Bewertungen für die Teilaussagen festgelegt:

- A_1*: "S. hat Kenntnisse in der Informatik":* $\widetilde{\delta}(A_1) = 0.9$
- A_2*: "S. hat Kenntnisse in BWL":* $\widetilde{\delta}(A_2) = 0.7$
- A_3*: "S. hat gute Berufsaussichten in der Entwicklung":* $\widetilde{\delta}(A_3) = 0.8$
- A_4*: "S. hat gute Berufsaussichten im Management":* $\widetilde{\delta}(A_4) = 0.4$

Damit kann nun der Wahrheitswert von Aussage A berechnet werden:

$$\begin{aligned}
\widetilde{\delta}(A) &= \widetilde{\delta}\big((A_1 \wedge A_2) \longrightarrow (A_3 \vee A_4)\big) \\
&= \widetilde{\delta}\big((0.9 \wedge 0.7) \longrightarrow (0.8 \vee 0.4)\big) \\
&= \widetilde{\delta}\big(\widetilde{\delta}(0.9 \wedge 0.7) \longrightarrow \widetilde{\delta}(0.8 \vee 0.4)\big) \\
&= \widetilde{\delta}\big(0.7 \longrightarrow 0.8\big) \\
&= \min\left\{1, 1 - 0.7 + 0.8\right\} = 1
\end{aligned}$$

D.h. A ist eine wahre Aussage.

Allgemein wird $\widetilde{\delta}(x_1 \longrightarrow x_2)$ nur kleiner als 1, wenn der Wahrheitsgrad von x_2 kleiner als der von x_1 ist. Die Implikation entspricht also der Vorschrift: "x_2 muß mindestens so wahr sein wie x_1", in Verallgemeinerung der Vorschrift "wenn x_1 wahr ist, dann muß auch x_2 wahr sein", die bei der Implikation in der klassischen Logik gilt.

Bei Wahrheitswerten 0 oder 1 verhalten sich die Operatoren der Fuzzy–Aussagenlogik genau so, wie die entsprechenden Operatoren der klassischen Aussagenlogik. D.h. $\widetilde{\delta}$ ist eine echte Erweiterung von δ. Wie für Fuzzy–Mengen gibt es auch in der Fuzzy–Aussagenlogik noch weitere Möglichkeiten, diese Operatoren zu definieren (s. Abschnitt 3.3). Die für die Durchschnitts– und Vereinigungs–Berechnung verwendeten T–Normen und T–Conormen sind gleichzeitig $\wedge$– bzw. $\vee$–Operatoren. Für die Implikation steht z.B. der Operator von Zadeh zur Verfügung:

Beispiel 3.21 *Zadehs Implikation:*

$$\widetilde{\delta}(x_1 \longrightarrow x_2) = \min\Big\{1, \max\big\{1 - \widetilde{\delta}(x_1), \widetilde{\delta}(x_2)\big\}\Big\}$$

Weitere Beispiele finden sich z.B. in [Feuring, 1996].

Definition 3.26
Eine Aussage $A(x_1, \ldots, x_n)$ heißt fuzzy–logisch erfüllbar, wenn es eine Belegung für $x_1, \ldots, x_n$ gibt mit $\widetilde{\delta}\big(A(x_1, \ldots, x_n)\big) \neq 0$.

Eine Aussage $A(x_1, \ldots, x_n)$ heißt fuzzy–logisch allgemeingültig, wenn für jede Belegung von $x_1, \ldots, x_n$ $\widetilde{\delta}\big(A(x_1, \ldots, x_n)\big) = 1$ gilt.

Eine Aussage $A(x_1, \ldots, x_n)$ heißt fuzzy–logisch unerfüllbar, wenn $\neg A$ allgemeingültig ist.

Eine Aussage wird demnach in der Fuzzy–Aussagenlogik bereits als erfüllbar bezeichnet, wenn es eine Belegung gibt, bei der sie nicht "ganz falsch" ist.

Die bekannten Äquivalenzen von Verknüpfungen aus der klassischen Aussagenlogik gelten in der Fuzzy–Logik i.A. nicht! So läßt sich z.B. $x_1 \longrightarrow x_2$ nicht durch $\neg x_1 \vee x_2$ ersetzen, ebenso ist der Wahrheitswert von $x_1 \wedge \neg x_1$ nicht immer 0 und der Wahrheitswert von $x_1 \vee \neg x_1$ nicht immer 1:

Beispiel 3.22

$$\widetilde{\delta}(0.6 \longrightarrow 0.4) = min\big\{1, 1 - 0.6 + 0.4\big\} = 0.8$$

$$\neq 0.4 = \max\big\{0.4, 0.4\big\} = \widetilde{\delta}(\neg 0.6 \vee 0.4)$$

$$\widetilde{\delta}(0.6 \wedge \neg 0.6) = \min\{0.6, 0.4\} = 0.4 \neq 0$$

$$\widetilde{\delta}(0.6 \vee \neg 0.6) = \max\{0.6, 0.4\} = 0.6 \neq 1$$

Auch die aussagenlogischen Schlüsse lassen sich nicht ohne Modifikationen auf die Fuzzy–Aussagenlogik übertragen. Z.B. ist der Modus ponens hier nicht allgemeingültig:

Beispiel 3.23 *Sei* $\widetilde{\delta}(x_1) = 0.8$ *und* $\widetilde{\delta}(x_2) = 0.3$, *dann gilt:*

$$\widetilde{\delta}\Big(\big(x_1 \wedge (x_1 \longrightarrow x_2)\big) \longrightarrow x_2\Big) = \widetilde{\delta}\Big(\big(0.8 \wedge (0.8 \longrightarrow 0.3)\big) \longrightarrow 0.3\Big)$$
$$= \widetilde{\delta}\Big(\delta\big(0.8 \wedge \widetilde{\delta}(0.8 \longrightarrow 0.3)\big) \longrightarrow 0.3\Big) = \widetilde{\delta}\Big(\widetilde{\delta}\big(0.8 \wedge 0.5\big) \longrightarrow 0.3\Big)$$
$$= \widetilde{\delta}\Big(0.5 \rightarrow 0.3\Big) = 0.8 \neq 1$$

Um dennoch aus fuzzy–Aussagen neue Aussagen herleiten zu können, ist eine neue Definition des Modus ponens erforderlich. Diese wird im folgenden Abschnitt 3.9 beschrieben.

3.9 Unscharfes Schließen

Unscharfes Schließen ist eine der wesentlichen Methoden, die in Fuzzy–Controllern angewendet werden. Das Ziel beim unscharfen Schließen ist es, aus unscharfen (= "fuzzy") Daten aufgrund von gegebenen Regeln Schlüsse zu ziehen. In den Regeln ist dabei das Wissen gespeichert, auf dem die Schlüsse basieren. Zur Formulierung der Regeln werden *linguistische Variablen* verwendet:

Definition 3.27 *Linguistische Variablen oder sprachliche Variablen sind Variablen, deren Werte keine Zahlen sind, sondern linguistische Terme, d.h. sprachliche Ausdrücke.*

Üblicherweise besteht der Wertebereich von linguistischen Variablen aus einer Menge von Adjektiven.

Beispiel 3.24 *Mögliche Wertebereiche einer linguistischen Variable für die Temperatur sind z.B.:*

{*niedrig, mittel, hoch*}

oder

$\{$*wenig heiß, etwas heiß, heiß, sehr heiß*$\}$

Um mit linguistischen Termen rechnen zu können, werden zur Repräsentation dieser Terme Fuzzy–Mengen definiert. D.h. für jeden verwendeten Term wird eine Fuzzy–Menge auf einem geeigneten Grundraum benötigt. Auf diese Weise lassen sich reelle Daten (z.B. Meßwerte) mit den linguistischen Termen verknüpfen.

Ist $\widetilde{A} \subset X$ eine Fuzzy–Menge auf dem Grundraum X und $x \in X$, dann gibt der Zugehörigkeitsgrad von x zu $\widetilde{A}$: $\mu_{\widetilde{A}}(x)$ an, zu welchem Grad x die Eigenschaft erfüllt, die $\widetilde{A}$ repräsentiert. Je höher der Grad ist, desto mehr erfüllt der Meßwert die Bedingung des zugehörigen linguistischen Terms. Grad 0 bedeutet, die Bedingung ist gar nicht erfüllt, Grad 1 bedeutet, die Bedingung ist vollständig erfüllt. Dabei ist es möglich, daß gleichzeitig mehrere Eigenschaften zu einem bestimmten Grad ungleich Null erfüllt werden.

Beispiel 3.25
Liegt die Temperatur eines Zimmers stets zwischen $10°\mathcal{C}$ *und* $30°\mathcal{C}$*, so ist das Intervall* $[10, 30]$ *ein geeigneter Grundraum. Als Fuzzy–Mengen werden dann z.B. folgende Dreiecks–Mengen* (l, m, r) *verwendet:*

$$\begin{aligned} \textit{niedrig} &\mathrel{\widehat{=}} \widetilde{A}_1 = (10, 15, 20) \\ \textit{mittel} &\mathrel{\widehat{=}} \widetilde{A}_2 = (15, 20, 25) \\ \textit{hoch} &\mathrel{\widehat{=}} \widetilde{A}_3 = (20, 25, 30) \end{aligned}$$

Ist die gemessene Temperatur $17.5°\mathcal{C}$*, so gilt:*

- *die Temperatur ist zum Grad 0.5 niedrig (da* $\mu_{\widetilde{A}_1}(17.5) = 0.5$*)*
- *die Temperatur ist zum Grad 0.5 mittel (da* $\mu_{\widetilde{A}_2}(17.5) = 0.5$*)*
- *die Temperatur ist zum Grad 0 hoch (da* $\mu_{\widetilde{A}_3}(17.5) = 0$*)*

Definition 3.28 *Die Einteilung des Grundraums in einzelne Bereiche durch verschiedene Fuzzy–Mengen heißt Partitionierung .*

Soll durch unscharfes Schließen die Leistung eines Heizkörpers in Abhängigkeit von der Temperatur gesteuert werden, so muß auch für die Heizstärke eine linguistische Variable verwendet werden. Analog zur Temperatur müssen geeignete linguistische Terme für die Heizleistung und zugehörige Fuzzy–Mengen definiert werden. Z.B. auf dem Grundraum $Y = [0, 10]$ Dreiecksmengen (l, m, r):

`schwach` $\widehat{=} \widetilde{B}_1 = (0, 2, 4)$
`normal` $\widehat{=} \widetilde{B}_2 = (3, 6, 9)$
`stark` $\widehat{=} \widetilde{B}_3 = (8, 9, 10)$

Das Wissen, auf dem die Schlüsse basieren, wird in Form von *"WENN–DANN–Regeln"* oder *"IF–THEN–Regeln"* repräsentiert:

Definition 3.29 *Seien x, y linguistische Variablen auf Grundräumen X, Y, für die geeignete Partitionierungen definiert sind. WENN–DANN–Regeln oder IF–THEN–Regeln sind Regeln der Gestalt:*

`WENN x = Term 1 DANN y = Term 2`

oder

`IF x = Term 1 THEN y = Term 2`

mit `Term 1` und `Term 2` linguistischen Termen für x bzw. y.

Alternativ lassen sich die Regeln auch direkt mit den Fuzzy–Mengen der Partitionierung statt den linguistischen Termen formulieren:

`IF x =` $\widetilde{A}_i$ `THEN y =` $\widetilde{B}_j$

Beispiel 3.26 *Für die Steuerung eines Heizkörpers sind folgende Regeln geeignet: Sei T die linguistische Variable für Temperatur, H die linguistische Variable für Heizleistung. Seien geeignete Partitionierungen zu den Termen*

{`niedrig, mittel, hoch`} für T

und

{`schwach, normal, stark`} für H

auf den zugehörigen Grundräumen definiert:

`WENN` $T =$ `niedrig DANN` $H =$ `stark`
`WENN` $T =$ `mittel DANN` $H =$ `normal`
`WENN` $T =$ `hoch DANN` $H =$ `schwach`

Die in Beispiel 3.26 definierten Regeln verdeutlichen einen wesentlichen Vorteil der Steuerung mit Methoden der Fuzzy–Logik: die Anschaulichkeit. Sowohl die verwendeten linguistischen Terme als auch die aufgeführten Regeln sind für jeden Anwender auch ohne Kenntnisse der Fuzzy–Logik verständlich. Tatsächlich ist es möglich, das Wissen eines Experten mit Hilfe von IF–THEN–Regeln in Kontrollsystemen zu speichern und zur Steuerung eines Prozesses zu verwenden.

Aufgrund dieser Regeln soll nun bei einer gegebenen Temperatur die geeignete Heizleistung bestimmt werden. Dazu wird zunächst die Temperatur t gemessen. Durch Berechnen der Zugehörigkeitsgrade von t zu den Fuzzy–Mengen der Partitionierung wird bestimmt, zu welchem Grad die einzelnen Bedingungen im WENN–Teil der Regeln erfüllt sind (s. Beispiel 3.25). Je mehr eine Bedingung erfüllt ist, desto stärker soll die Folgerung dieser Regel berücksichtigt werden.

Anstatt den reellen Meßwert direkt zu verwenden, wird er in einem Zwischenschritt auf eine Fuzzy–Menge abgebildet. Dieser Vorgang wird *Fuzzifizierung* genannt, Einzelheiten dazu werden in Abschnitt 3.10 beschrieben. Auf diese Weise ist es möglich, Unsicherheiten der reellen Werte (Meßfehler) zu berücksichtigen. Statt mit dem exakten, aber möglicherweise fehlerhaften reellen Wert 20 wird z.B. mit der Dreiecks–Menge $(18, 20, 22)$ ("ungefähr 20") weiter gerechnet.

Liegt nach der Fuzzifizierung eine Fuzzy–Menge $\widetilde{A}$ vor, so braucht diese Menge nicht eine der bei der Partitionierung definierten Mengen zu sein. D.h. keine der IF–Bedingungen ist genau erfüllt. Dann muß aber auch keine der THEN–Folgerungen genau erfüllt werden. Wie soll in diesem Fall die geeignete Heizleistung berechnet werden?

Das Ziel beim unscharfen Schließen ist es, jeweils aus einer gegebenen Regel und einem als Fuzzy–Menge gegebenen Fakt eine neue Fuzzy–Menge zu berechnen. Dieser Schluß wird als *generalisierter Modus ponens* bezeichnet:

Definition 3.30 *generalisierter Modus ponens*

Seien auf Grundräumen X und Y Partitionierungen definiert.
Sei die Regel **IF** $x = \widetilde{A}_i$ **THEN** $y = \widetilde{B}_j$ *mit $\widetilde{A}_i \subset X$, $\widetilde{B}_j \subset Y$ gegeben.*

Sei weiterhin als fuzzifizierte Menge $\widetilde{A} \subset X$ gegeben, d.h. es gilt $x = \widetilde{A}$.
Dann wird das Ergebnis $y = \widetilde{B}$ berechnet gemäß:

$$\mu_{\widetilde{B}}(y) = \max_{x \in X} \left\{ \mu_{\widetilde{A}}(x) * \mu_{\widetilde{A_i}}(x) * \mu_{\widetilde{B_j}}(y) \right\}$$

wobei $*$ jeweils für eine T–Norm steht, z.B. den Minimum–Operator.

Anschaulicher ist die schematische Darstellung des verallgemeinerten Modus ponens:

$$\begin{array}{l} \texttt{IF } x = \widetilde{A_i} \texttt{ THEN } y = \widetilde{B_j} \\ x = \widetilde{A} \\ \hline y = \widetilde{B} \end{array}$$

Statt dem Minimum–Operator lassen sich auch andere T–Normen verwenden, Möglichkeiten für geeignete T–Normen finden sich z.B. in [Feuring, 1996].

Beispiel 3.27
Seien auf dem Grundraum $X = \{1, 2, 3\}$ *die Fuzzy–Menge* $\widetilde{A}_1$ *gemäß*

$\mu_{\widetilde{A}_1}(1) = 0.5$, $\mu_{\widetilde{A}_1}(2) = 1.0$ *und* $\mu_{\widetilde{A}_1}(3) = 0.6$ *definiert*

und auf dem Grundraum $Y = \{10, 20, 30\}$ *die Fuzzy–Menge* $\widetilde{B}_1$ *gemäß*

$\mu_{\widetilde{B}_1}(10) = 0.5$, $\mu_{\widetilde{B}_1}(20) = 1.0$ *und* $\mu_{\widetilde{B}_1}(30) = 0.5$.

Sei die Regel `IF` $x = \widetilde{A}_1$ `THEN` $y = \widetilde{B}_1$ *gegeben.*

Zur Fuzzy–Menge $\widetilde{A} \subset X$ *mit*

$\mu_{\widetilde{A}}(1) = 0.4$, $\mu_{\widetilde{A}}(2) = 0.9$ *und* $\mu_{\widetilde{A}}(3) = 0.5$

ergibt sich mit dem generalisierten Modus ponens die Fuzzy–Menge $\widetilde{B} \subset Y$ *als:*

$$\mu_{\widetilde{B}}(10) = \max_{x \in X} \Big\{ \min\{\mu_{\widetilde{A}}(x), \mu_{\widetilde{A}_i}(x), \mu_{\widetilde{B}}(10)\} \Big\}$$

$$= \max \Big\{ \min\{\mu_{\widetilde{A}}(1), \mu_{\widetilde{A}_i}(1), \mu_{\widetilde{B}}(10)\}, \min\{\mu_{\widetilde{A}}(2), \mu_{\widetilde{A}_i}(2), \mu_{\widetilde{B}}(10)\}, \min\{\mu_{\widetilde{A}}(3), \mu_{\widetilde{A}_i}(3), \mu_{\widetilde{B}}(10)\} \Big\}$$

$$= \max \Big\{ \min\{0.5, 0.4, 0.5\}, \min\{1.0, 0.9, 0.5\}, \min\{0.6, 0.5, 0.5\} \Big\}$$

$$= \max \big\{0.4, 0.5, 0.5\big\} = 0.5$$

analog ergeben sich:

$$\mu_{\widetilde{B}}(20) = 0.9$$

$$\mu_{\widetilde{B}}(30) = 0.5$$

Mit dem generalisierten Modus ponens ist es möglich, aus einer Regel und einer gegebenen Fuzzy–Menge eine neue Fuzzy–Menge zu berechnen. Die erste $*$–Verknüpfung kombiniert dabei die gegebene Fuzzy–Menge mit der Fuzzy–Menge aus der Regel–Voraussetzung. Dabei gilt für jedes x: je ähnlicher die beiden Fuzzy–Mengen sind, desto größer ist der Wert von $\mu_{\widetilde{A}}(x) * \mu_{\widetilde{A_1}}(x)$.

Die zweite $*$–Verknüpfung bewirkt eine Kombination mit der Fuzzy–Menge aus der Regel–Folgerung. Für das Ergebnis $\widetilde{B}$ gilt: je "ähnlicher" $\widetilde{A}$ der Regel–Voraussetzung $\widetilde{A}_1$ ist, desto "ähnlicher" ist $\widetilde{B}$ der Regel–Folgerung $\widetilde{B}_1$.

Zur Steuerung eines Gerätes oder einer Anlage ist eine Fuzzy–Menge als Ergebnis des verallgemeinerten Modus ponens nicht geeignet. Daher muß diese Fuzzy–Menge auf eine reelle Zahl abgebildet werden. Dieser Vorgang heißt *Defuzzifizierung*. Da i.A. mehrere Regeln definiert sind, müssen die Ergebnisse der Regeln in geeigneter Weise kombiniert werden. Das unscharfe Schließen, die Defuzzifizierung und die Kombination der Regelergebnisse sind Bestandteile eines Fuzzy–Controllers. Daher werden sie im Abschnitt über Fuzzy–Controller 3.10 ausführlich beschrieben.

3.10 Der Fuzzy–Controller nach Mamdani

Fuzzy–Controller sind die wichtigste, auch kommerziell erfolgreiche, Anwendung der Fuzzy–Theorie. Sie werden vorzugsweise zur Steuerung von Prozessen eingesetzt, z.B. für einen Zement–Drehrohrofen. Es gibt grundsätzlich zwei verschiedene Arten von Fuzzy–Controllern, den *Sugeno–Controller* nach [Sugeno, 1985] und den *Mamdani-Controller* nach [Mamdani, 1974]. In diesem Abschnitt wird der Fuzzy–Controller nach Mamdanis Definition vorgestellt. Auf den Sugeno–Controller wird in Abschnitt 3.12 eingegangen.

Definition 3.31
Seien $X_1 \times \ldots \times X_n \subset \mathbb{R}^n$ *der Eingaberaum,*
$Y_1 \times \ldots \times Y_m \subset \mathbb{R}^m$ *der Ausgaberaum eines zu steuernden Systems.*

Ein Fuzzy–Controller (nach Mamdanis Definition) ist eine Steuereinheit, die

Eingaben $(r_1, \ldots, r_n) \in X_1 \times \ldots \times X_n$

auf Ausgaben $(s_1, \ldots, s_m) \in Y_1 \times \ldots \times Y_m$

abbildet.

Im wesentlichen besteht ein Fuzzy–Controller aus vier Komponenten:

- Fuzzifizierer
- Regelbasis
- Entscheidungslogik
- Defuzzifizierer

Definition 3.32
Der Fuzzifizierer berechnet aus den reellen Eingaben Fuzzy–Mengen (s. Abb. 3.6).

Ein Eingabewert $r_i \in X_i$ *wird dabei z.B. in die Dreiecks–Menge* $(r_i - \varepsilon, r_i, r_i + \varepsilon)$ *mit einem* $\varepsilon > 0$ *umgewandelt.*

In den meisten Fällen wird jedoch ein Singleton–Fuzzifizierer verwendet. D.h. ein Eingabewert $r_i \in X_i$ *wird in die Fuzzy–Menge* $\widetilde{R}_i$ *umgewandelt mit:*

$$\mu_{\widetilde{R}_i}(x) = \begin{cases} 1 : x = r_i \\ 0 : x \neq r_i \end{cases}$$

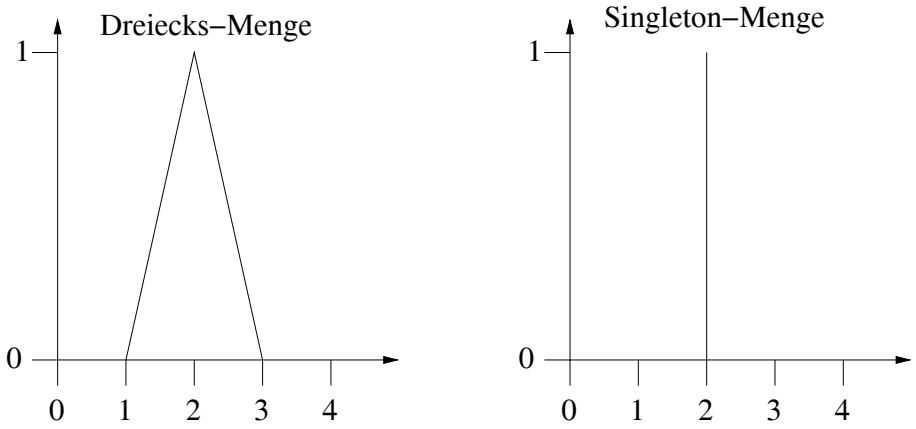

Abbildung 3.6: Dreiecks–Menge und Singleton–Menge zur Eingabe 2.0

Definition 3.33 *Die Regelbasis besteht aus endlich vielen WENN–DANN–Regeln bzw. IF–THEN–Regeln. Dabei wird für jede Dimension des Eingaberaumes $X_1 \times \ldots \times X_n$ eine linguistische Variable x_i verwendet. Die einzelnen Werte werden mit einer UND–Verknüpfung kombiniert.*

Für die Werte von jeder linguistischen Variablen x_i werden geeignete Fuzzy–Mengen auf ihrer Eingabe–Dimension X_i definiert. Die Anzahl der Fuzzy–Mengen können für jede Eingabe–Dimension unterschiedlich gewählt werden, ebenso die Eingabe–Dimensionen selber. Negative Bereiche sind ebenfalls möglich. Analog müssen geeignete Partitionierungen für die linguistischen Variablen y_j der Dimensionen des Ausgaberaumes $Y_1 \times \ldots \times Y_m$ definiert werden.

Beispiel 3.28
Sollen als Eingaben zur Steuerung eines Heizkörpers Meßwerte für die Temperatur und die Luftfeuchtigkeit verwendet werden, so ist $X_1 \times X_2$ mit

$X_1 = [10, 30]$ *für die Temperatur in Grad Celsius und*
$X_2 = [0, 100]$ *für die Luftfeuchtigkeit in Prozent*

ein geeigneter Eingaberaum.

Linguistische Terme für x_1 sind z.B.

$\{$`kalt, mittelwarm, warm`$\}$

und für x_2

$\{$`trocken, normal, feucht, sehr feucht`$\}$

Zu diesen Termen sind jeweils passende Fuzzy–Mengen auf X_1 bzw. X_2 zu definieren. Z.B. Dreiecksmengen (l, m, r):

$$\begin{aligned}
\texttt{kalt} &\mathrel{\widehat{=}} \widetilde{A}_{11} = (10, 14, 18)\\
\texttt{mittelwarm} &\mathrel{\widehat{=}} \widetilde{A}_{12} = (14, 18, 22)\\
\texttt{warm} &\mathrel{\widehat{=}} \widetilde{A}_{13} = (18, 24, 30)
\end{aligned}$$

$$\begin{aligned}
\texttt{trocken} &\mathrel{\widehat{=}} \widetilde{A}_{21} = (0, 15, 30)\\
\texttt{normal} &\mathrel{\widehat{=}} \widetilde{A}_{22} = (15, 35, 55)\\
\texttt{feucht} &\mathrel{\widehat{=}} \widetilde{A}_{23} = (50, 70, 90)\\
\texttt{sehr feucht} &\mathrel{\widehat{=}} \widetilde{A}_{24} = (80, 90, 100)
\end{aligned}$$

Liegt die Einstellung der Heizleistung zwischen 0 und 10, so ist $Y = [0, 10]$ ein geeigneter Ausgaberaum, eine geeignete Partitionierung ist:

$\texttt{schwach} \quad \widehat{=} \widetilde{B}_1 = (0, 2, 4)$
$\texttt{mittelstark} \widehat{=} \widetilde{B}_2 = (3, 6, 9)$
$\texttt{stark} \quad \widehat{=} \widetilde{B}_3 = (8, 9, 10)$

Passende Regeln sind:

R_1: *WENN* x_1 = *kalt* *UND* x_2 = *sehr feucht* *DANN* y = *stark*
R_2: *WENN* x_1 = *kalt* *UND* x_2 = *feucht* *DANN* y = *mittelstark*
R_3: *WENN* x_1 = *mittelwarm* *UND* x_2 = *normal* *DANN* y = *mittelstark*
R_4: *WENN* x_1 = *warm* *UND* x_2 = *trocken* *DANN* y = *schwach*

Die Regelbasis ist das Herzstück eines Fuzzy–Controllers. In den Regeln ist das Wissen eines Experten gespeichert, wie in welcher Situation reagiert werden soll. Die Voraussetzung oder *Prämisse* einer Regel beschreibt eine Situation, die Folgerung oder *Konklusion* beschreibt die korrekte Reaktion in dieser Situation.

Definition 3.34 *Der WENN–Teil bzw. IF–Teil einer Regel heißt Prämisse , der DANN–Teil bzw. THEN–Teil heißt Konklusion.*

Es genügt, Regeln zu verwenden, deren Konklusion sich nur auf die linguistische Variable einer Dimension des Ausgaberaumes bezieht. Folgerungen für mehrere Dimensionen des Ausgaberaumes werden einfach durch mehrere Regeln abgedeckt.

Beispiel 3.29
IF $x_1 = \widetilde{A}_{1*}$ *UND* ...*UND* $x_n = \widetilde{A}_{n*}$ *THEN* $y_1 = \widetilde{B}_{1*}$ *UND* $y_2 = \widetilde{B}_{2*}$

wird in äquivalenter weise ersetzt durch

IF $x_1 = \widetilde{A}_{1*}$ *UND* ...*UND* $x_n = \widetilde{A}_{n*}$ *THEN* $y_1 = \widetilde{B}_{1*}$
IF $x_1 = \widetilde{A}_{1*}$ *UND* ...*UND* $x_n = \widetilde{A}_{n*}$ *THEN* $y_2 = \widetilde{B}_{2*}$

Die in Abschnitt 3.11 vorgeführte schrittweise Berechnung der Ausgabe eines Fuzzy–Controllers macht deutlich, daß die Regel mit beiden Ausgabe–Dimensionen genau dieselben Ergebnisse liefert, wie beide Regeln mit je einer Ausgabe–Dimension. Daher wird im Folgenden o.B.d.A. für jede Regel folgende Gestalt vorausgesetzt:

IF $x_1 = \widetilde{A}_{1*}$ UND ...UND $x_n = \widetilde{A}_{n*}$ THEN $y_j = \widetilde{B}_{j*}$

mit $\widetilde{A}_{i*}$ einer Fuzzy–Menge auf X_i und $\widetilde{B}_{j*}$ einer Fuzzy–Menge auf Y_j.

Definition 3.35
Die Entscheidungslogik berechnet unter Berücksichtigung der Regeln aus den Eingabe–Fuzzy–Mengen, die der Fuzzifizierer liefert, die Ausgabe–Fuzzy–Mengen. Dafür wird für jede Dimension des Ausgaberaumes eine Ausgabe–Fuzzy–Menge ermittelt.

Zur Bestimmung jeder Ausgabe–Fuzzy–Menge werden jeweils alle Regeln berücksichtigt, deren Konklusion die zugehörige Ausgabe–Dimension betreffen.

Der Defuzzifizierer berechnet für jede Ausgabe–Dimension aus der Ausgabe–Fuzzy–Menge einen reellen Ausgabewert. D.h. der Deffuzzifizierer realisiert eine Abbildung von den Fuzzy–Mengen nach $\mathbb{R}$. Alle Ausgabewerte zusammen sind die Ausgabe des Fuzzy–Controllers $(s_1, \ldots, s_m) \in Y_1 \times \ldots \times Y_m$.

Abb. 3.7 zeigt schematisch den Aufbau eines Fuzzy–Controllers nach Mamdani:

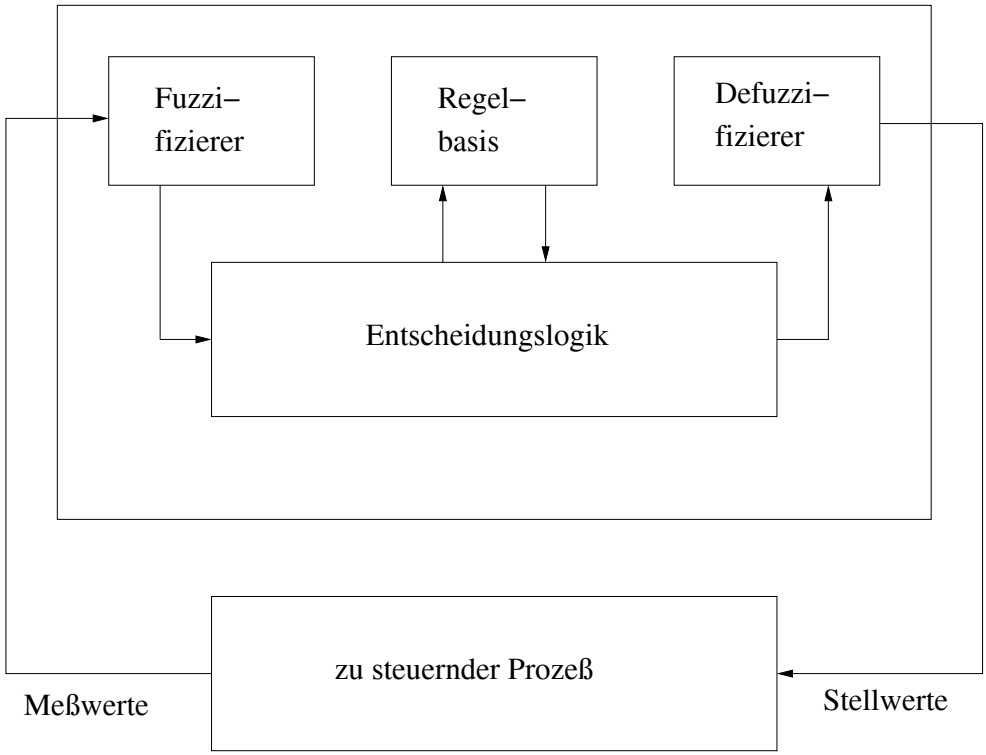

Abbildung 3.7: Aufbau eines Fuzzy–Controllers

Sämtliche Einzelheiten zu den Berechnungen der Entscheidungslogik und des Defuzzifizierers werden im folgenden Abschnitt 3.11 beschrieben.

3.11 Arbeitsweise eine Mamdani–Controllers

Es folgt eine schrittweise Beschreibung, wie ein Fuzzy–Controller aus den Eingaben die Ausgaben berechnet. Dabei wird vorausgesetzt, daß ein Singleton–Fuzzifizierer verwendet wird. Dies ist die in der Praxis am häufigsten gewählte Methode und stellt keine Einschränkung bzgl. der Leistungsfähigkeit des Fuzzy–Controllers dar (s. Abschnitt 3.13).

Seien die Partitionierungen und die Regelbasis eines Fuzzy–Controllers nach Mamdani definiert. Sei die Eingabe $(r_1, \ldots, r_n) \in X_1 \times \ldots \times X_n$ gegeben.

1. Erfüllungsgrade der Prämissen berechnen:

Als erstes wird für jede Regel einzeln der *Erfüllungsgrad der Prämisse* berechnet. Dies ist ein Maßstab dafür, zu welchem Grad die Eingabe mit der Regel–Prämisse übereinstimmt. Da ein Singleton–Fuzzifizierer verwendet wird, entfällt die Berechnung der Eingabe–Fuzzy–Mengen und die vom generalisierten Modus ponens (s. Abschnitt 3.9) bekannte Kombination von Fuzzy–Mengen. Der Erfüllungsgrad der Prämisse wird in zwei Schritten berechnet:

Zuerst werden die Zugehörigkeitsgrade der Eingabe–Werte zu den Fuzzy–Mengen der Regel–Prämisse bestimmt. D.h. für jede Regel R_k werden

$$E_{k1} := \mu_{\widetilde{A}_{1*}}(r_1) \ , \ \ldots \ , \ E_{kn} := \mu_{\widetilde{A}_{n*}}(r_n)$$

berechnet. Dabei sind $\widetilde{A}_{1*}, \ldots, \widetilde{A}_{n*}$ die bei dieser Regel verwendeten Partitions–Fuzzy–Mengen. Falls eine linguistische Variable x_i in einer Regel R_k nicht vorkommt, entfällt die Berechnung von E_{ki}.

Anschließend werden für jede Regel R_k diese Werte mit einer T–Norm verknüpft. Der Erfüllungsgrad E_k für Regel R_k ergibt sich also durch:

$$E_k = \widetilde{\delta}(E_{k1} \wedge \ldots \wedge E_{kn})$$

Wobei $\wedge$ eine T–Norm ist und nicht berechnete Werte von E_{ki} wegfallen.

Definition 3.36
Seien R_k eine linguistische Regel und $E_{k1}, \ldots, E_{k_n}$ die Zugehörigkeitsgrade der reellen Eingaben $r_1, \ldots, r_n$ zu den Fuzzy–Mengen aus der Regel–Prämisse.

$$E_k := \widetilde{\delta}(E_{k1} \wedge \ldots \wedge E_{kn})$$

heißt Erfüllungsgrad der Prämisse von Regel R_k.

Beispiel 3.30 *Seien die Regeln und Partitionierungen aus Beispiel 3.28 gegeben. Sei die Eingabe* $(17.0, 85.0)$ *(d.h. Temperatur* $17°\mathcal{C}$*, Luftfeuchtigkeit* 85%*).*

Der Erfüllungsgrad der Prämisse von Regel 1 ist dann:

$$E_1 = \widetilde{\delta}\big(\mu_{\widetilde{A}_{11}}(17) \wedge \mu_{\widetilde{A}_{24}}(85)\big) = \widetilde{\delta}\big(0.25 \wedge 0,5\big) = 0.25$$

2. Ergebnisse der Regeln berechnen:

Als nächstes wird für jede Regel R_k das Ergebnis der Anwendung dieser Regel bestimmt. Bezieht sich die Konklusion von Regel R_k auf Ausgabe–Dimension Y_j, so ist dieses Ergebnis eine Fuzzy–Menge $\widetilde{C}_k \subset Y_j$. Zur Berechnung von $\widetilde{C}_k$ wird der Erfüllungsgrad der Prämisse von Regel R_k, E_k, mit der Fuzzy–Menge $\widetilde{B}_{k*}$ der Konklusion dieser Regel verknüpft. Zum Verknüpfen wird eine Fuzzy–Implikation verwendet. Mamdani hat hier zur Vereinfachung der Berechnung das Minimum ausgewählt, es sind jedoch auch andere Definitionen möglich. Für $\widetilde{C}_k$ gilt:

$$\mu_{\widetilde{C}_k}(y_j) = \begin{cases} \mu_{\widetilde{B}_{k*}}(y_j) : & \mu_{\widetilde{B}_{k*}}(y_j) < E_k \\ E_k : & \mu_{\widetilde{B}_{k*}}(y_j) \geq E_k \end{cases}$$

Das Ergebnis $\widetilde{C}_k$ der Anwendung von Regel R_k entsteht somit durch "abschneiden" von $\widetilde{B}_{k*}$ in Höhe des Erfüllungsgrades E_k, wie Abb. 3.8 verdeutlicht.

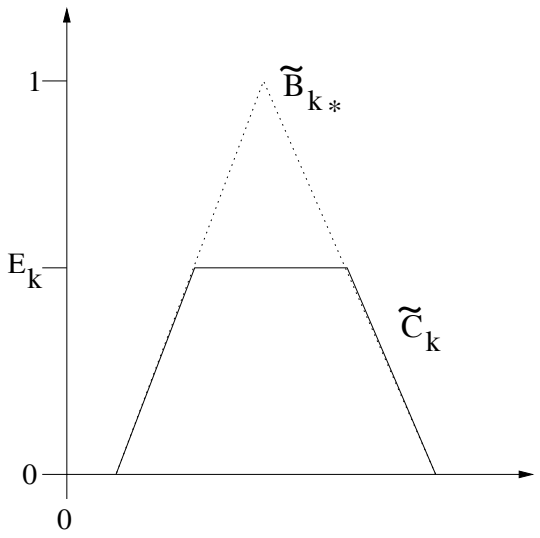

Abbildung 3.8: Ergebnis–Fuzzy–Menge einer Regel

3. Ausgabe–Fuzzy–Mengen berechnen:

Nun wird für jede Ausgabe–Dimension die Ausgabe–Fuzzy–Menge berechnet. Dazu wird jeweils die Vereinigung sämtlicher Ergebnis–Fuzzy–Mengen $\widetilde{C}_k$ gebildet, die in dieser Ausgabe–Dimension liegen. Zur Vereinigung wird eine T–Conorm verwendet. Mamdani wählt dazu das Maximum, es sind jedoch auch andere Definitionen möglich. Für die Ausgabe–Fuzzy–Menge $\widetilde{D}_j$ auf der Ausgabe–Dimension Y_j gilt somit:

$$\widetilde{D}_j = \widetilde{C}_{l_1} \cup \ldots \cup \widetilde{C}_{l_{\#R(j)}}$$

Wobei $\widetilde{C}_{l_1}, \ldots, \widetilde{C}_{l_{\#R(j)}}$ die Ergebnis–Fuzzy–Mengen auf der Ausgabe–Dimension Y_j sind und $\#R(j)$ die Anzahl der Regeln ist, die sich auf Ausgabe–Dimension Y_j beziehen.

Abb. 3.9 zeigt die Vereinigung von drei Ergebniss–Fuzzy–Mengen.

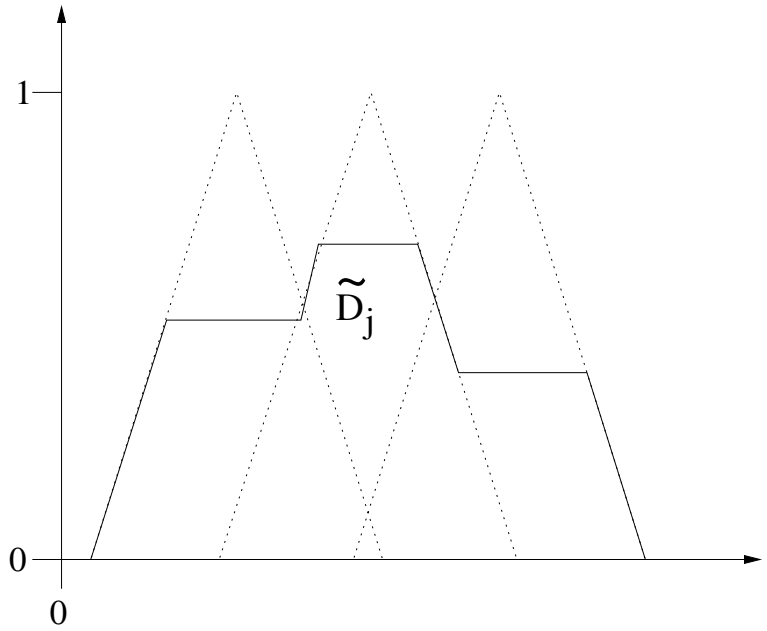

Abbildung 3.9: Vereinigungs–Fuzzy–Menge

4. Defuzzifizierung:

Da der Fuzzy–Controller reell–wertige Ausgaben liefern soll, muß in einem letzten Schritt aus den berechneten Ausgabe–Fuzzy–Mengen $\widetilde{D}_j$, $j = 1, \ldots, m$, jeweils eine reelle Zahl s_j berechnet werden, d.h. die Fuzzy–Mengen $\widetilde{D}_j$ werden *defuzzifiziert.*

Die Defuzzifizierung ist eine Abbildung von den Fuzzy–Mengen nach $\mathbb{R}$. Zur Definition dieser Abbildung gibt es verschiedene Möglichkeiten, die bekanntesten sind die *Maximum–Methode* und die *Schwerpunkt–Methode.*

Definition 3.37 *Sei $\widetilde{D}_j \subset Y_j$ eine Fuzzy–Menge.*

Bei der Maximum–Methode wird eine Stelle s_j aus der Ausgabe–Dimension Y_j bestimmt, für die $\mu_{\widetilde{D}_j}$ maximalen Wert hat. D.h. für das Ergebnis der Defuzzifizierung s_j gilt:

$$s_j = y_j \in Y_j \mid \mu_{\widetilde{D}_j}(y_j) maximal$$

Bei der Schwerpunkt–Methode wird der Schwerpunkt von $\widetilde{D}_j$ gemäß folgender Formel bestimmt:

$$s_j = \frac{1}{\int\limits_{y_j \in Y_j} \mu_{\widetilde{D}_j}(y_j) dy_j} \int\limits_{y_j \in Y_j} y_j \cdot \mu_{\widetilde{D}_j}(y_j) dy_j$$

Ein Vorteil der Maximum–Methode ist, daß sie leicht zu berechnen ist. Ein Nachteil ist, daß das Ergebnis nicht eindeutig bestimmt ist, falls die Zugehörigkeitsfunktion $\mu_{\widetilde{D}_j}$ an mehreren Stellen ihren maximalen Wert annimmt. In diesem Fall muß von den Werten mit maximalem Zugehörigkeitsgrad einer ausgesucht werden, z.B. durch ein Zufallsverfahren. Die Schwerpunkt–Methode ist eindeutig und liefert meistens gute Ergebnisse. Sie wird daher in der Praxis am häufigsten eingesetzt, jedoch ist die Berechnung aufwendiger. Die Ergebnisse der Maximum–Methode und der Schwerpunkt–Methode sind i.A. unterschiedlich. Welche Methode die bessere ist, hängt von der konkreten Anwendung ab und muß bei der Erstellung eines Fuzzy–Controllers im Einzelfall entschieden werden.

Beispiel 3.31 *Es soll ein Fuzzy–Controller zur Lenkung eines automatischen Fahrzeugs (z.B. selbständiger Staubsauger oder Rasenmäher) eingesetzt werden. Die Regeln sehen vor, daß bei einem Hindernis nach links oder nach rechts ausgewichen werden kann. Hierzu werden auf dem Ausgabe–Raum Y (Lenkereinschlag) geeignete Fuzzy–Mengen für* `links` *und* `rechts` *definiert. Fährt das Fahrzeug auf ein Hindernis zu, ist folgende Situation möglich (s. Abb. 3.10):*

Die Regeln zum Ausweichen nach links und nach rechts haben den selben, maximalen Erfüllungsgrad der Prämisse, z.B. 0.9; alle anderen Regeln haben einen Erfüllungsgrad der Prämisse gleich Null. Dadurch hat die Ausgabe–Fuzzy–Menge $\widetilde{D}$ die Gestalt aus Abb. 3.10. Die mit der Schwerpunkt–Methode berechnete Ausgabe (s_schwer) ist 0. Der Schwerpunkt der Möglichkeiten "nach links ausweichen" und "nach rechts ausweichen" ist genau in der Mitte, und das bedeutet in diesem Fall "geradeaus fahren". D.h. das Fahrzeug stößt mit dem Hindernis zusammen! Mit der Maximum–Methode ergibt sich als Ausgabe (s_max) der Wert -30, d.h. das Fahrzeug fährt links am Hindernis vorbei.

Beispiel 3.31 zeigt, wie sich unterschiedliche Defuzzifizierungsmethoden auf die berechnete Ausgabe auswirken. In diesem Extremfall hängt sogar der Erfolg von der Wahl der richtigen Methode ab. Es ist jedoch zu bemerken, daß auch bei Verwendung der Schwerpunkt–Methode eine korrekte Steuerung eines Fahrzeugs möglich ist, falls die

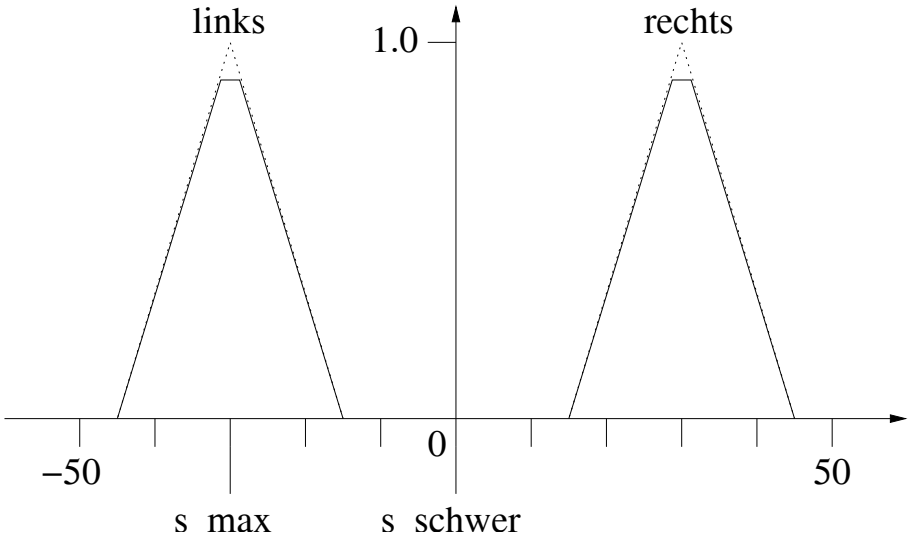

Abbildung 3.10: Schwerpunkt– und Maximum–Methode

Partitionierungen und die Regeln geeignet definiert werden. Für die korrekte Erstellung eines Fuzzy–Controllers sind alle Bestandteile von Bedeutung, insbesondere die Art der Defuzzifizierung, die Partitionierungen und die Regeln.

Die Ergebnisse der Defuzzifizierung sind die Ausgabe des Fuzzy–Controllers $(s_1, \ldots, s_m)$. Bei der Berechnung werden immer alle Regeln ausgewertet, wobei gilt: je mehr die Eingabe–Werte der Situation entsprechen, die die Prämisse einer Regel beschreibt (hoher Erfüllungsgrad), desto größer ist der Einfluß dieser Regel auf die Ausgabe–Werte.

3.12 Der Sugeno–Controller

Der wesentliche Unterschied eines Sugeno–Controllers im Vergleich zum Fuzzy–Controller nach Mamdani liegt in den Konklusionen der Regeln und der Berechnung der Ausgabe. Beim Sugeno–Controller werden als Konklusion der Regeln reellwertige Parameter oder lineare Funktionen an Stelle von Fuzzy–Mengen verwendet. Die Ausgabe ist ein Mittelwert der gewichteten Summe aus Konklusionen und Erfüllungsgraden der Regeln.

Definition 3.38 *Ein Sugeno–Controller besteht aus Eingabe–Partitionierungen und Regeln der Form:*

R_k : `IF` $x_1 = \widetilde{A}_{1*}$ `UND` ... `UND` $x_n = \widetilde{A}_{n*}$ `THEN` $y = f_k(x_1, \ldots, x_n)$

mit $f_k : \mathbb{R}^n \longrightarrow \mathbb{R}$ *linear bzw. konstant.*

Die Erfüllungsgrade E_k jeder Regel R_k werden analog wie beim Fuzzy–Controller nach Mamdani bestimmt.

Die reelle Ausgabe $s_j, j = 1 \ldots m$, des Controllers zur Eingabe $(r_1, \ldots, r_n)$ wird mit folgender Formel berechnet:

$$s_j = \frac{\sum\limits_{R_k \in Reg(j)} E_k \cdot f_k(r_1, \ldots, r_n)}{\sum\limits_{R_k \in Reg(j)} E_k}$$

mit $Reg(j)$ der Menge aller Regeln, die sich in der Konklusion auf Ausgabe–Dimension Y_j beziehen.

Ein Vorteil von Sugeno–Controllern ist die einfachere Berechnung der Ausgaben im Vergleich zum Mamdani–Controller. Die aufwendige Bestimmung der Ausgabe–Fuzzy–Mengen und die Defuzzifizierung entfallen. Ein Nachteil ist die fehlende Anschaulichkeit der Regeln, da bei den Konklusionen statt linguistischer Terme reelle Parameter oder lineare Funktionen verwendet werden. Hierdurch wird auch die Konstruktion des Sugeno–Controllers erschwert. Zur Bestimmung der Konklusions–Funktionen der Regeln müssen mathematische Zusammenhänge zwischen den Eingaben und den korrekten Ausgaben modelliert werden.

Sowohl der Mamdani– als auch der Sugeno–Controller sind universelle Approximatoren. D.h. stetige Funktionen lassen sich mit diesen Modellen beliebig genau approximieren. Im folgenden Abschnitt 3.13 wird dies betrachet.

3.13 Universalität von Fuzzy–Controllern

Mit Fuzzy–Controllern nach Mamdani und mit Sugeno–Controllern lassen sich unter bestimmten Voraussetzungen alle stetigen Funktionen beliebig genau approximieren.

Satz 3.3 *Sei $U \subset \mathbb{R}^n$ kompakt. Dann liegt die Menge aller mit einem Mamdani–Controller berechenbaren Funktionen dicht im Raum der stetigen Funktionen auf U, $\mathcal{C}(U)$, falls folgende Spezifikationen eingehalten werden: Singleton–Fuzzifizierung, Verwendung von Gauß–Mengen, Produkt als T–Norm und Schwerpunkt–Defuzzifizierung.*

Beweis 3.3 *Dieser Satz wird mit Hilfe des Approximationssatzes von Stone–Weierstraß bewiesen, vgl. [Mendel, 1992] und [Wang, 1992].* □

Satz 3.4 *Jede stetige Funktion läßt sich mit einem Sugeno–Controller beliebig genau approximieren, falls als Regel–Konklusion Polynome statt linearer Funktionen verwendet werden und die Ausgabe als Linearkombination dieser Polynome und der Erfüllungsgrade der Regeln berechnet wird:*

$$o_j = \sum_{R_k \in Reg(j)} E_k \cdot f_k(r_1, \ldots, r_n)$$

Beweis 3.4 *Zum Beweis siehe [Buckley, 1993].* □

Die Sätze 3.3 und 3.4 zeigen, daß Fuzzy–Controller nach Mamdani und Sugeno–Controller mit bestimmten Vorgaben bzw. Modifikationen nachweislich universelle Approximatoren sind. Diese Eigenschaft und der relativ einfache, nachvollziehbare Aufbau eines Fuzzy–Controllers haben zur Popularität dieser Modelle beigetragen. Obwohl die Universalität nur unter bestimmten Bedingungen tatsächlich bewiesen ist, gibt es in der Praxis auch andere Variationen, die ebenfalls erfolgreich eingesetzt werden, s. z.B. [Nauck, 1996] und [Tilli, 1993].

3.14 Anwendung von Fuzzy–Controllern

In der klassischen Regelungstechnik wird ein physikalisch–mathematisches Modell des zu regelnden Systems erstellt, meist in Form von Differentialgleichungen. Dies ist aber oftmals sehr aufwendig, wenn nicht gar unmöglich, und selbst wenn alle Gleichungen erstellt wurden, ist es oft schwierig, sie zu lösen. Außerdem muß der Entwickler des Reglers über ein entsprechendes mathematisches und physikalisches Wissen verfügen.

Einen ganz anderen Ansatz bieten nun Fuzzy–Controller: hiermit soll das Verhalten eines Menschen, der ein System regeln kann, simuliert werden. Der Vorteil ist, daß das Wissen des Experten durch einfache linguistische Regeln auf den Fuzzy–Controller übertragen wird. Die Regeln beschreiben, wie bei einer bestimmten Konstellation der Meßgrößen die Stellgrößen zu wählen sind.

Als Beispiel soll nun das Stabbalance–Problem behandelt werden. Beim *Stabbalance–Problem* oder *inversen Pendel* geht es darum, einen Stab wie auf dem Finger zu balancieren. Die Ausgleichsbewegungen, die ein Mensch dabei intuitiv macht, damit der Stab nicht umkippt, sollen nun von einem Fuzzy–Controller gesteuert werden.

Dazu muß zunächst ein Modell dieses Vorgangs entwickelt werden. Zur Vereinfachung wird vorausgesetzt, daß der Stab so auf einem Wagen befestigt ist, daß er nur in zwei Richtungen kippen kann. Der Wagen soll auf einer Schiene zum Ausgleich in die gleichen Richtungen bewegt werden (s. Abb. 3.11). Das Ziel ist es, den Stab so zu balancieren, daß er möglichst immer senkrecht steht und nicht umkippt.

Das System wird nun durch den aktuellen Wert vom Winkel des Stabes bezüglich der senkrechten Achse ϑ und der aktuellen Winkelgeschwindigkeit $\dot{\vartheta}$ vollständig beschrieben. Die Aufgabe ist es, in Abhängigkeit von ϑ und $\dot{\vartheta}$ die Kraft F zu bestimmen, die

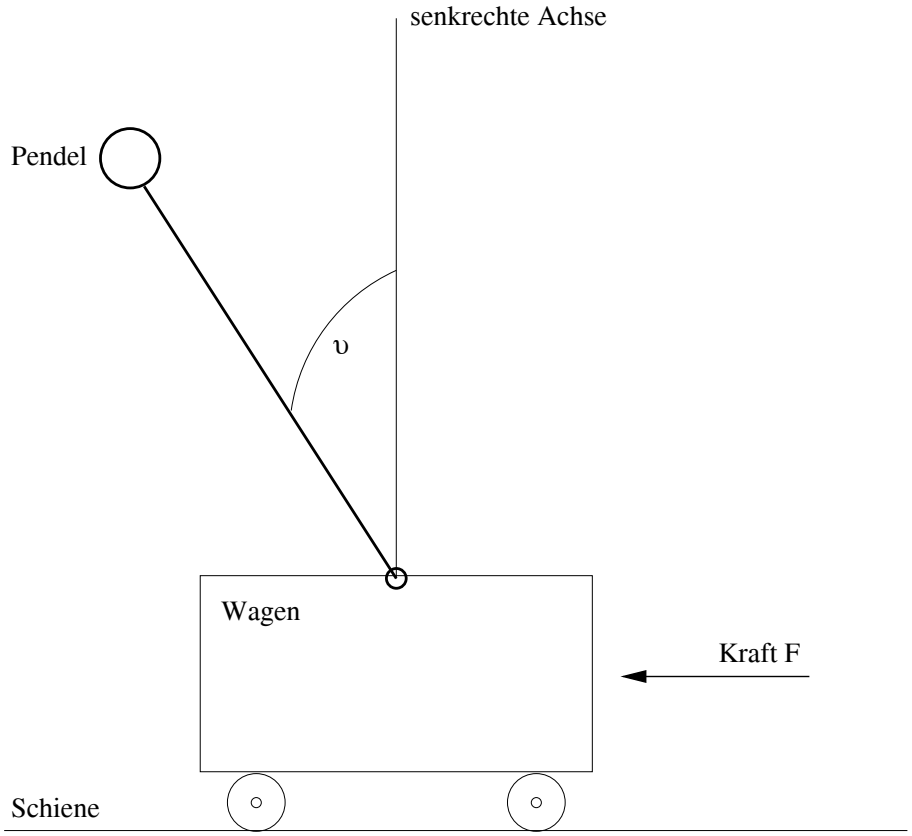

Abbildung 3.11: Modell des inversen Pendels

auf den Wagen einwirken soll, damit das Pendel nicht umkippt. Geeigneter Wertebereich für den Winkel ist etwa $-90° \leq \vartheta \leq 90°$, und für die Winkelgeschwindigkeit $-45\frac{°}{s} \leq \dot{\vartheta} \leq 45\frac{°}{s}$. Für die Kraft wird $-10N \leq F \leq 10N$ (Newton) angenommen.

Für die Konstruktion des Fuzzy–Controllers sind nun geeignete Fuzzy–Mengen und eine passende Regelbasis zu erstellen. Im Prinzip müssen dazu einfach die verschiedenen möglichen Situationen durch geeignete Fuzzy–Mengen repräsentiert werden, um dann jeweils mit einer Regel die richtige Reaktion anzugeben.

Beim Balancieren auf dem Finger bewegt man die Hand nach links, wenn der Stab nach links kippt. Bewegt der Stab sich besonders schnell, oder ist er schon weit gekippt, erfolgt die Bewegung der Hand entsprechend schneller. Diese "Regeln" und "Werte" müssen nun konkret formuliert werden, außerdem müssen geeignete Fuzzy–Mengen definiert werden.

Diese Beschreibungen verdeutlichen einen Vorteil von Fuzzy–Controllern: es ist auf einfache Weise möglich, das Wissen eines Experten zu repräsentieren. Dazu ist es lediglich nötig, die Vorgänge beim Steuerungs–Prozeß genau zu analysieren und zu beschreiben. Zur Beschreibung können simple WENN–DANN–Regeln unter Verwendung von geeig-

neten Adjektiven direkt übernommen werden. Danach müssen noch zu den verwendeten Adjektiven (d.h. linguistischen Termen) passende Fuzzy–Mengen auf geeigneten Eingabe– und Ausgabe–Dimensionen definiert werden. Die Wertebereiche ergeben sich zumeist aus den Skalen der Meßinstrumente bzw. der Regler. Die korrekte Anzahl der Fuzzy–Mengen und die Formulierung der Regeln ist Aufgabe des Experten.

Beispiel 3.32 *Für das Stabbalance–Problem sind die geeigneten Grundräume:*

$X_1 = [-90, 90]$
$X_2 = [-45, 45]$
$Y = [-10, 10]$

Dabei werden für alle linguistischen Variablen (x_1 für Winkel, x_2 für Winkelgeschwindigkeit und y für Kraft) die gleichen linguistischen Terme verwendet:

negativ groß (ng)
negativ mittel (nm)
negativ klein (nk)

ungefähr null (un)

positiv klein (pk)
positiv mittel (pm)
positiv groß (pg)

Jedoch sind für jede linguistische Variable jeweils zu diesen Termen eigene, unterschiedliche Fuzzy–Mengen auf ihrem Grundraum zu definieren. In diesem Beispiel werden Dreiecks–Mengen verwendet:

auf X_1:

ng $\widehat{=} \widetilde{A}_{11} = (-120, -90, -60)$
nm $\widehat{=} \widetilde{A}_{12} = (-90, -60, -30)$
nk $\widehat{=} \widetilde{A}_{13} = (-60, -30, 0)$
un $\widehat{=} \widetilde{A}_{14} = (-30, 0, 30)$
pk $\widehat{=} \widetilde{A}_{15} = (0, 30, 60)$
pm $\widehat{=} \widetilde{A}_{16} = (30, 60, 90)$
pg $\widehat{=} \widetilde{A}_{17} = (60, 90, 120)$

auf X_2:

ng $\widehat{=} \widetilde{A}_{21} = (-60, -45, -30)$
nm $\widehat{=} \widetilde{A}_{22} = (-45, -30, -15)$

$nk \mathrel{\widehat{=}} \widetilde{A}_{23} = (-30, -15, 0)$
$un \mathrel{\widehat{=}} \widetilde{A}_{24} = (-15, 0, 15)$
$pk \mathrel{\widehat{=}} \widetilde{A}_{25} = (0, 15, 30)$
$pm \mathrel{\widehat{=}} \widetilde{A}_{26} = (15, 30, 45)$
$pg \mathrel{\widehat{=}} \widetilde{A}_{27} = (30, 45, 60)$

auf Y:

$ng \mathrel{\widehat{=}} \widetilde{B}_1 = (-13.2, -9.9, -6.6)$
$nm \mathrel{\widehat{=}} \widetilde{B}_2 = (-9.9, -6.6, -3.3)$
$nk \mathrel{\widehat{=}} \widetilde{B}_3 = (-6.6, -3.3, 0)$
$un \mathrel{\widehat{=}} \widetilde{B}_4 = (-3.3, 0, 3.3)$
$pk \mathrel{\widehat{=}} \widetilde{B}_5 = (0, 3.3, 6.6)$
$pm \mathrel{\widehat{=}} \widetilde{B}_6 = (3.3, 6.6, 9.9)$
$pg \mathrel{\widehat{=}} \widetilde{B}_7 = (6.6, 9.9, 13.2)$

Folgende Regelbasis wird gewählt:

R_{01} : IF $x_1 =$ nk UND $x_2 =$ ng THEN $y =$ pk
R_{02} : IF $x_1 =$ un UND $x_2 =$ ng THEN $y =$ pg
R_{03} : IF $x_1 =$ un UND $x_2 =$ nm THEN $y =$ pm
R_{04} : IF $x_1 =$ ng UND $x_2 =$ nk THEN $y =$ nm
R_{05} : IF $x_1 =$ nk UND $x_2 =$ nk THEN $y =$ pk
R_{06} : IF $x_1 =$ un UND $x_2 =$ nk THEN $y =$ pk
R_{07} : IF $x_1 =$ ng UND $x_2 =$ un THEN $y =$ ng
R_{08} : IF $x_1 =$ nm UND $x_2 =$ un THEN $y =$ nm
R_{09} : IF $x_1 =$ nk UND $x_2 =$ un THEN $y =$ nk
R_{10} : IF $x_1 =$ un UND $x_2 =$ un THEN $y =$ un
R_{11} : IF $x_1 =$ pk UND $x_2 =$ un THEN $y =$ pk
R_{12} : IF $x_1 =$ pm UND $x_2 =$ un THEN $y =$ pm
R_{13} : IF $x_1 =$ pg UND $x_2 =$ un THEN $y =$ pg
R_{14} : IF $x_1 =$ un UND $x_2 =$ pk THEN $y =$ nk
R_{15} : IF $x_1 =$ pk UND $x_2 =$ pk THEN $y =$ nk
R_{16} : IF $x_1 =$ pg UND $x_2 =$ pk THEN $y =$ pm
R_{17} : IF $x_1 =$ un UND $x_2 =$ pm THEN $y =$ nm
R_{18} : IF $x_1 =$ un UND $x_2 =$ pg THEN $y =$ ng
R_{19} : IF $x_1 =$ pk UND $x_2 =$ pg THEN $y =$ nk

Diese Regeln wurden mit Hilfe eines Experten erstellt und ermöglichen die korrekte Steuerung des inversen Pendels (s. [Nauck, 1996]). Sie decken zwar nicht jede theoretisch mögliche Kombination von Eingabe–Werten ab, jedoch repräsentieren sie alle in der Praxis vorkommenden Situationen. Deshalb ist mit diesen Regeln eine geeignete Regelbasis definiert.

Damit ist die wesentliche Arbeit für den Einsatz eines Fuzzy–Controllers zur Steuerung des inversen Pendels getan. Es müssen lediglich noch die UND–Verknüpfung (nach Mamdani: Minimum) und der Vereinigungs–Operator (nach Mamdani: Maximum) sowie die Defuzzifizierungsmethode festgelegt werden.

Ein Nachteil von Fuzzy–Controllern ist, daß für die Berechnung der Ausgabe jedesmal alle Regeln der Reihe nach ausgewertet werden müssen. Dies ist mit einem hohen Zeit– und Rechenaufwand verbunden. Ein weiterer Nachteil ist, daß Fuzzy–Controller sich im Gegensatz zu neuronalen Netzen nicht selber anpassen können. Vor dem Einsatz zur Steuerung müssen sämtliche Regeln, die Fuzzy–Mengen, die benötigten Verknüpfungen und die Methode zur Defuzzifizierung festgelegt werden.

Dies bringt Probleme mit sich, wenn z.B. kein Experten–Wissen für die Regeln zur Verfügung steht. Oder wenn einzelne Regeln fehlerhaft sind oder die Fuzzy–Mengen ungünstig gewählt wurden etc. Um diesen Nachteil zu kompensieren, wurden verschiedene Methoden entwickelt, Fuzzy–Controller mit neuronalen Netzen zu kombinieren, denn diese können sich selber ändern und anpassen. D.h. es werden die adaptiven Fähigkeiten von neuronalen Netzen genutzt, um Bestandteile eines Fuzzy–Controllers zu verbessern.

Bekannte Methoden, dies zu realisieren, sind z.B. das Verfahren von Lin und Lee (s. [LinLee, 1991]), das NEFCON–Modell (s. [Nauck, 1996]) oder das ANFIS–System (s. [Jang, 1993]). Diese Modelle werden in den Kapiteln 4, 5 und 8 vorgestellt. Schließlich wird in den Kapiteln 6 und 9 mit dem MFOS–System ein eigenes Modell als Alternative zu den bekannten Systemen entwickelt.

Teil II

Optimierungsmethoden für Mamdani–Controller

Kapitel 4

Das Verfahren von Lin und Lee

Von C.T. Lin und C.S.G. Lee [LinLee, 1991] wurde 1991 ein neuronales Netz entwickelt, mit dem ein Fuzzy–Controller nach Mamdani repräsentiert werden kann. Dabei wird die Regelbasis des Fuzzy–Controllers durch die Struktur des Netzes gespeichert. Als Gewichte werden statt reeller Zahlen Fuzzy–Mengen verwendet. Die Funktionen der einzelnen Neuronen sind so definiert, daß das Netz exakt die gleichen Berechnungen durchführt wie der Fuzzy–Controller, den es repräsentiert. Somit entspricht die Netzausgabe der Ausgabe dieses Fuzzy–Controllers.

Der Vorteil dieser Übertragung eines Fuzzy–Controllers auf ein neuronales Netz ist, daß nun die bei neuronalen Netzen üblichen Lernmethoden zur Verbesserung des Fuzzy–Controllers anwendbar sind. Beim Verfahren von Lin und Lee werden die Gewichte (und somit die Fuzzy–Mengen) mit Hilfe einer hybriden Methode unter Verwendung des Backpropagation–Verfahrens angepaßt. Außerdem ist es möglich, neue Ausgabe–Partitions–Mengen zu erzeugen und die Konklusionen von Regeln zu ändern.

4.1 Aufbau und Arbeitsweise

Das verwendete fünfschichtige Netz (Abb. 4.1) ist wie folgt aufgebaut: die Eingabeschicht enthält für jeden Eingabewert ein Neuron. In Schicht 2 gibt es für jeden vorhandenen linguistischen Term der Eingabe–Partitionierungen ein Neuron. Dieses ist jeweils mit dem Eingabe–Neuron verbunden, das der gleichen Eingabe–Dimension zugeordnet ist. Als Gewicht wird dabei die Eingabe–Partitions–Menge verwendet, die den jeweiligen linguistischen Term repräsentiert. Die verwendeten Fuzzy–Mengen sind Gauß– bzw. Dreiecks–Mengen.

Schicht 3 enthält für jede Regel ein Neuron. Dieses ist genau mit den Neuronen aus Schicht 2 verbunden, die für die linguistischen Terme der Prämisse dieser Regel stehen. Dabei werden keine Gewichte verwendet. In Schicht 4 gibt es analog zu Schicht 2 für jeden vorhandenen linguistischen Term der Ausgabe–Partitionierungen ein Neuron. Dieses ist jeweils mit allen Neuronen aus Schicht 3 verbunden, deren zugehörige

Regel diesen Term als Konklusion hat. Dabei werden keine Gewichte verwendet. In der Ausgabeschicht gibt es für jede Ausgabe–Dimension ein Neuron. Dieses ist mit allen Neuronen aus Schicht 4 verbunden, deren linguistischer Term zu dieser Ausgabe–Dimension gehört. Als Gewicht wird jeweils die dem linguistischen Term zugeordnete Ausgabe–Partitions–Menge verwendet.

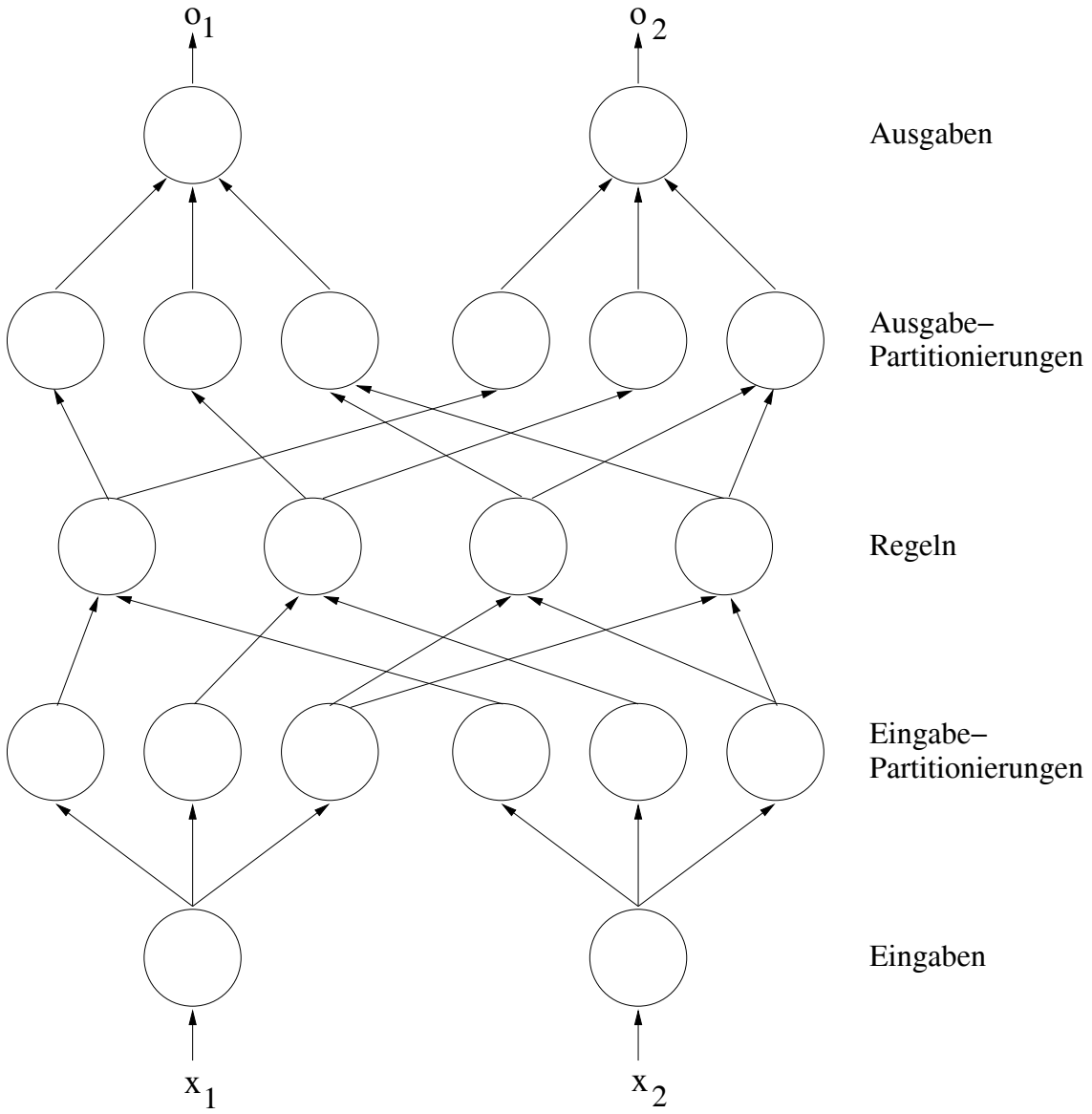

Abbildung 4.1: Aufbau eines Lin und Lee–Netzes

Die Regelbasis des Fuzzy–Controllers wird somit durch die Verbindungen zwischen den Schichten zwei, drei und vier gespeichert; die Partitionierungen werden in den Gewichten gespeichert.

Beispiel 4.1 *Hat Regel 1 die Gestalt:*

IF $x_1 =$ `mittel` *UND* $x_2 =$ `hoch` *THEN* $y_1 =$ `schwach`,

so gibt es folgende Verbindungen mit Neuron 1 aus Schicht 3 (s. Abb. 4.2):

- *aus der zweiten Schicht empfängt es Verbindungen von den Neuronen für* `mittel` *aus Eingabe–Partitionierung 1 und* `hoch` *aus Eingabe–Partitionierung 2*
- *zu Schicht 4 gibt es eine Verbindung mit dem Neuron für* `schwach` *aus Ausgabe–Partitionierung 1.*

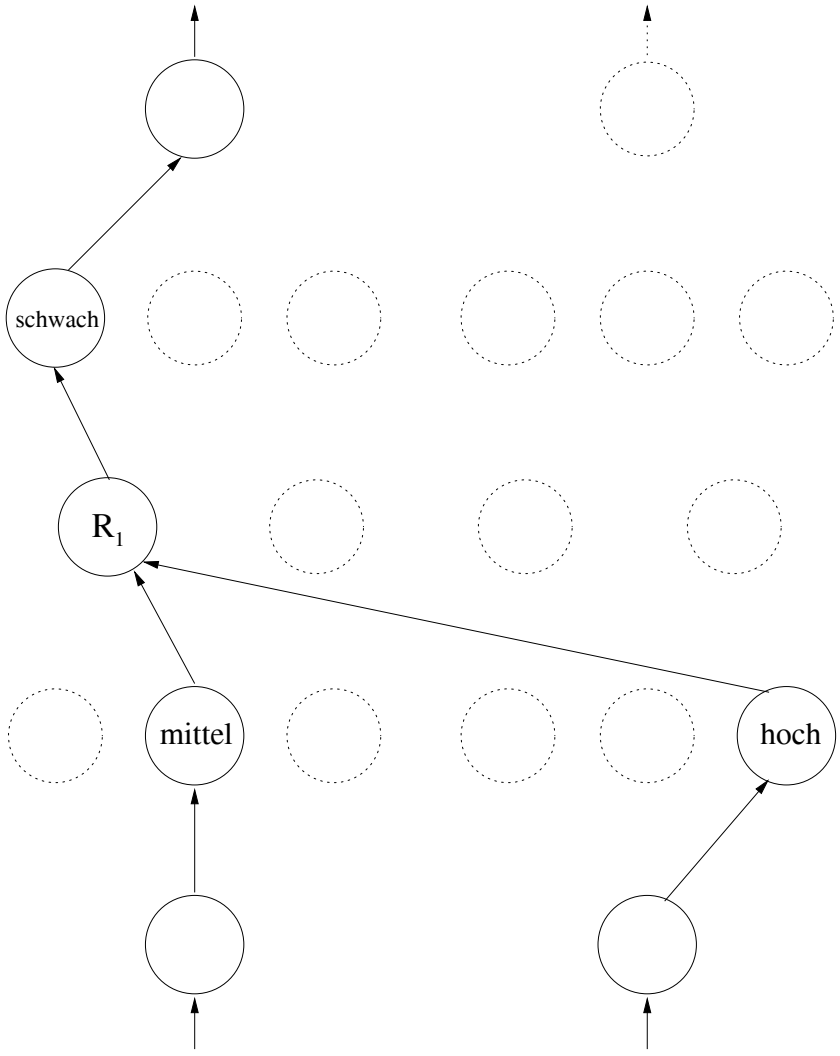

Abbildung 4.2: Regel 1 repräsentierende Verbindungen

Die Berechnung der Netzausgabe entspricht der Berechnung der Ausgabe eines Fuzzy–Controllers. In Schicht 1 gibt jedes Neuron seine Eingabe unverändert weiter. Jedes Neuron aus Schicht 2 berechnet den Zugehörigkeitsgrad seines Eingabe–Wertes zu der Fuzzy–Menge, die als Gewicht der Verbindung mit Schicht 1 verwendet wird. In Schicht 3 berechnet jedes Neuron das Minimum seiner Eingabe–Werte. Dies ergibt den Erfüllungsgrad der Prämisse von der Regel, die es repräsentiert.

In Schicht 4 berechnet jedes Neuron das Minimum zwischen 1 und der Summe seiner Eingabe–Werte. Damit wird eine ODER–Verknüpfung realisiert, um die Schnitthöhe der zugehörigen Fuzzy–Menge zu bestimmen. In Schicht 5 berechnet jedes Neuron j seine Ausgabe wie folgt:

$$o_j = \frac{\sum\limits_{i \in S_4} m_{5,i,j} \cdot w_{5,i,j} \cdot z_{4i}}{\sum\limits_{i \in S_4} w_{5,i,j} \cdot z_{4i}}$$

wobei die Summe jeweils nur über die Neuronen i aus Schicht 4 (S_4) läuft, mit denen das aktuelle Ausgabe–Neuron verbunden ist, z_{4i} die jeweilige Ausgabe ist und $m_{5,i,j}$ der Modalwert, $w_{5,i,j}$ die Weite der Fuzzy–Menge der Verbindungen mit Schicht 4 ist. Auf diese Weise wird eine Schwerpunkt–Defuzzifizierung approximiert.

Das Netz verhält sich nun so, wie der Fuzzy–Controller, den es repräsentiert. D.h. zu gleichen Eingaben berechnet das Netz dieselben Ausgaben wie dieser Fuzzy–Controller. Im Gegensatz zu einem Fuzzy–Controller gibt es jedoch die Möglichkeit, Lernverfahren anzuwenden.

4.2 Das hybride Lernverfahren

Beim Verfahren von Lin und Lee kommt eine hybride Lernmethode basierend auf dem Backpropagation–Algorithmus zum Einsatz. Die Gewichte des Netzes, d.h. in diesem Fall die Fuzzy–Mengen, werden mit dem Backpropagation–Algorithmus trainiert (s. Kapitel 1 und [Zinth, 1996]).

Zusätzlich werden bei Bedarf neue Neuronen in Schicht 4 erzeugt, wobei gleichzeitig geeignete Verbindungen mit Schicht 3 und Schicht 5 generiert werden. Diese strukturelle Änderung des Netzes entspricht der Erzeugung neuer Ausgabe–Partitions–Mengen sowie der Bildung neuer Konklusionen für einige Regeln.

Zur Entscheidung, ob neue Ausgabe–Fuzzy–Mengen benötigt werden, wird ein *Fuzzy–Ähnlichkeitsmaß* E mit Werten zwischen 0 und 1 verwendet:

Definition 4.1 *Fuzzy–Ähnlichkeitsmaß E*

seien $\widetilde{A}_1 \subset X$ *und* $\widetilde{A}_2 \subset X$ *Fuzzy–Mengen auf einer Grundmenge X. Dann gilt:*

$$E(\widetilde{A}_1, \widetilde{A}_2) := \frac{M(\widetilde{A}_1 \cap \widetilde{A}_2)}{M(\widetilde{A}_1 \cup \widetilde{A}_2)}$$

mit

$$M(\widetilde{A}) := \sum_{x \in X} \mu_{\widetilde{A}}(x)$$

der Mächtigkeit einer Fuzzy–Menge $\widetilde{A} \subset X$.

Mit diesem Fuzzy–Ähnlichkeitsmaß wird der Grad der Gleichheit von zwei Fuzzy–Mengen bestimmt. Dabei gilt, je ähnlicher die Mengen sind, desto größer ist der Wert des Ähnlichkeitsmaßes. Zwei gleiche Mengen haben Ähnlichkeitsmaß 1, d.h. es gilt:

Satz 4.1 *Sei* $\widetilde{A} \subset X$ *eine Fuzzy–Menge auf einem Grundraum X. Dann gilt bei Verwendung des Minimums als T–Norm und des Maximums als T–Conorm:*

$$E(\widetilde{A}, \widetilde{A}) = 1.0$$

Beweis 4.1 *Dieser Satz ist eine direkte Folgerung aus der Definition des Ähnlichkeitsmaßes.* □

Beispiel 4.2 *Seien auf dem Grundraum* $X = \{1, 2, 3, 4\}$ *die Fuzzy–Mengen* $\widetilde{A}_1$ *gemäß*

$\mu_{\widetilde{A}_1}(1) = 0.2$, $\mu_{\widetilde{A}_1}(2) = 0.4$, $\mu_{\widetilde{A}_1}(3) = 0.9$ $\mu_{\widetilde{A}_1}(4) = 0.7$

und $\widetilde{A}_2$ *gemäß*

$\mu_{\widetilde{A}_2}(1) = 0.4$, $\mu_{\widetilde{A}_2}(2) = 0.5$, $\mu_{\widetilde{A}_2}(3) = 0.8$, $\mu_{\widetilde{A}_2}(4) = 0.6$

definiert. Dann gilt:

$$E(\widetilde{A}_1, \widetilde{A}_2) = \frac{M(\widetilde{A}_1 \cap \widetilde{A}_2)}{M(\widetilde{A}_1 \cup \widetilde{A}_2)} = \frac{\sum\limits_{x \in X} \mu_{\widetilde{A}_1 \cap \widetilde{A}_2}(x)}{\sum\limits_{x \in X} \mu_{\widetilde{A}_1 \cup \widetilde{A}_2}(x)}$$

$$= \frac{\sum\limits_{x \in X} \min\left\{\mu_{\widetilde{A}_1}(x), \mu_{\widetilde{A}_2}(x)\right\}}{\sum\limits_{x \in X} \max\left\{\mu_{\widetilde{A}_1}(x), \mu_{\widetilde{A}_2}(x)\right\}}$$

$$= \frac{\min\left\{0.2, 0.4\right\} + \min\left\{0.4, 0.5\right\} + \min\left\{0.9, 0.8\right\} + \min\left\{0.7, 0.6\right\}}{\max\left\{0.2, 0.4\right\} + \max\left\{0.4, 0.5\right\} + \max\left\{0.9, 0.8\right\} + \max\left\{0.7, 0.6\right\}}$$

$$= \frac{0.2 + 0.4 + 0.8 + 0.6}{0.4 + 0.5 + 0.9 + 0.7} = \frac{2.0}{2.5} = 0.8$$

Das hybride Lernverfahren basiert auf der Annahme, daß eine aus der Ausgabe–Partitions–Menge $\widetilde{B}_{j*}$ mit dem Backpropagation–Algorithmus berechnete Fuzzy–Menge $\widetilde{B}_{j*}^{neu}$ die im Vergleich zu $\widetilde{B}_{j*}$ bessere Konklusion einer Regel mit hohem Erfüllungsgrad ist. Daher soll bei Regeln mit hohem Erfüllungsgrad der Prämisse jeweils die Konklusion $\widetilde{B}_{j*}$ durch $\widetilde{B}_{j*}^{neu}$ ersetzt werden.

Dies geschieht jedoch nicht analog zum Backpropagation–Verfahren durch einfaches Ändern der Fuzzy–Mengen. Stattdessen wird eine der folgenden drei Möglichkeiten durchgeführt:

1. ist $\widetilde{B}_{j*}^{neu}$ ähnlich zu $\widetilde{B}_{j*}$, wird die $\widetilde{B}_{j*}$ geändert zu $\widetilde{B}_{j*}^{neu}$
2. ist $\widetilde{B}_{j*}^{neu}$ ähnlich zu einer anderen Fuzzy–Menge $\widetilde{B}_{j*}^{x}$, wird diese als neue Konklusion der betroffenen Regeln eingesetzt
3. ist $\widetilde{B}_{j*}^{neu}$ zu keiner vorhandenen Ausgabe–Partitions–Menge ähnlich genug, wird $\widetilde{B}_{j*}^{neu}$ als neue Ausgabe–Partitions–Menge eingefügt und als Konklusion betroffener Regeln eingesetzt

Auf diese Weise erhält jede vorhandene Regel die optimale Konklusion. Fall 1 entspricht dem gewöhnlichen Backpropagation–Verfahren, Fall 2 ändert die Konklusion einer vorhandenen Regel und Fall 3 erzeugt eine zusätzliche Ausgabe–Partitions–Menge. Die

Eingabe–Partitions–Mengen werden direkt mit dem Backpropagation–Verfahren adaptiert. Die neuen Modalwerte und Weiten der Ausgabe–Partitions–Mengen werden gemäß dem Backpropagation–Algorithmus nach folgenden Formeln berechnet (zur Herleitung s. [Zinth, 1996]):

$$\Delta m_{5,i,j} = \eta \cdot \left((w_{5,i,j} \cdot z_{4i}) \cdot \frac{y_j - o_j}{\sum\limits_{k \in S_4} w_{5,k,j} \cdot z_{4,k}} \right)$$

$$\Delta w_{5,i,j} = \eta \cdot \left(\frac{m_{5,i,j} \cdot z_{4i} \cdot \left(\sum\limits_{k \in S_4} w_{5,k,j} \cdot z_{4k} \right) - \left(\sum\limits_{k \in S_4} m_{5,k,j} \cdot w_{5,k,j} \cdot z_{4,k} \right) \cdot z_{4i}}{\left(\sum\limits_{k \in S_4} w_{5,k,j} \cdot z_{4,k} \right)^2} \cdot (y_j - o_j) \right)$$

mit

- $m_{5,i,j}$ und $w_{5,i,j}$ Modalwert und Weite der Ausgabe–Partitions–Menge, die Verbindungsgewicht zwischen Neuron i aus Schicht 4 und Neuron j aus der Ausgabeschicht ist
- z_{4i} bzw. z_{4k} den Ausgaben von Neuron i bzw. k aus Schicht 4
- o_j der berechneten Ausgabe, y_j der gewünschten Ausgabe von Ausgabeneuron j
- η der üblichen Lernrate

Beispiel 4.3 *Sei folgende Situation gegeben:*

$$m_{5,1,1} = 10, w_{5,1,1} = 5, m_{5,2,1} = 20, w_{5,2,1} = 5, z_{41} = 0.2, z_{42} = 0.8, y_1 = 20, \eta = 0.1$$

Durch Einsetzen der Werte ergibt sich:

$$o_1 = \frac{m_{5,1,1} \cdot w_{5,1,1} \cdot z_{41} + m_{5,2,1} \cdot w_{5,2,1} \cdot z_{42}}{w_{5,1,1} \cdot z_{41} + w_{5,2,1} \cdot z_{42}} = \frac{10 \cdot 5 \cdot 0.2 + 20 \cdot 5 \cdot 0.8}{5 \cdot 0.2 + 5 \cdot 0.8} = 18$$

Somit ist der Fehler der berechneten Ausgabe $|y_1 - o_1| = |20 - 18| = 2$

Und es gilt:

$$\Delta m_{5,1,1} = 0.1 \cdot \left((5 \cdot 0.2) \cdot \frac{20 - 18}{5 \cdot 0.2 + 5 \cdot 0.8} \right) = 0.1 \cdot \frac{2}{5} = \frac{1}{25}$$

$$\Delta w_{5,1,1} = 0.1 \cdot \left(\frac{10 \cdot 0.2 \cdot \Big(5 \cdot 0.2 + 5 \cdot 0.8\Big) - \Big(10 \cdot 5 \cdot 0.2 + 20 \cdot 5 \cdot 0.8\Big) \cdot 0.2}{\Big(5 \cdot 0.2 + 5 \cdot 0.8\Big)^2} \cdot \big(20 - 18\big) \right)$$

$$= 0.1 \cdot \frac{10 - 18}{25} \cdot 2 = 0.1 \cdot -\frac{16}{25} = -\frac{16}{250}$$

$$\Delta m_{5,2,1} = 0.1 \cdot \left((5 \cdot 0.8) \cdot \frac{20 - 18}{5 \cdot 0.2 + 5 \cdot 0.8} \right) = 0.1 \cdot \frac{8}{5} = \frac{4}{25}$$

$$\Delta w_{5,2,1} = 0.1 \cdot \left(\frac{20 \cdot 0.8 \cdot \Big(5 \cdot 0.2 + 5 \cdot 0.8\Big) - \Big(10 \cdot 5 \cdot 0.2 + 20 \cdot 5 \cdot 0.8\Big) \cdot 0.8}{\Big(5 \cdot 0.2 + 5 \cdot 0.8\Big)^2} \cdot \big(20 - 18\big) \right)$$

$$= 0.1 \cdot \frac{80 - 72}{25} \cdot 2 = 0.1 \cdot \frac{16}{25} = \frac{16}{250}$$

Die neuen Modalwerte und Weiten sind somit:

$$m_{5,1,1}^{neu} = 10 + \frac{1}{25} = 10.04$$

$$w_{5,1,1}^{neu} = 5 - \frac{16}{250} = 4.936$$

$$m_{5,2,1}^{neu} = 20 + \frac{4}{25} = 20.16$$

$$w_{5,2,1}^{neu} = 5 + \frac{16}{250} = 5.064$$

Damit ergibt sich als neue Ausgabe:

$$o_1^{neu} = \frac{10.04 \cdot 4.936 \cdot 0.2 + 20.16 \cdot 5.064 \cdot 0.8}{4.936 \cdot 0.2 + 5.064 \cdot 0.8}$$

$$= \frac{9.911488 + 81.672192}{0.9872 + 4.0512} = \frac{91.58368}{5.0384} = 18.177135$$

Somit ist der Fehler der neuen berechneten Ausgabe $|y_1 - o_1| = |20 - 18.177135|$ $= 1.822865$

Dieses Beispiel zeigt, daß die in einem Schritt durchgeführten Änderungen der Modalwerte und Weiten eine Verbesserung der berechneten Ausgabe bewirken. Beim gewöhnlichen Backpropagation werden diese Änderungen wiederholt durchgeführt, bis der Fehler der berechneten Ausgabe unter einer Schranke liegt. Beim hybriden Lernverfahren werden diese Änderungen jedoch nicht direkt durchgeführt. Der Ablauf des hybriden Lernverfahrens ist stattdessen wie folgt:

Es wird der Reihe nach für jede Ausgabe–Partitions–Menge $\widetilde{B}_{j*}$ gemäß dem Backpropagation–Algorithmus eine modifizierte Fuzzy–Menge $\widetilde{B}_{j*}^{neu}$ berechnet, ohne die ursprüngliche Menge zu verändern. Anschließend wird jeweils mit Hilfe des Ähnlichkeitsmaßes E bestimmt, zu welcher der vorhandenen Ausgabe–Partitions–Mengen der zugehörigen Ausgabe–Dimension die neue Menge am ähnlichsten ist.

Falls bei keiner der Ausgabe–Partitions–Mengen der Ähnlichkeitsgrad über einem Mindestwert liegt, wird ein neues Neuron in Schicht 4 eingefügt, das nun $\widetilde{B}_{j*}^{neu}$ als neue

Ausgabe–Fuzzy–Menge repräsentiert. Dieses Neuron wird verbunden mit allen Neuronen aus Schicht 3, die vorher mit dem Neuron aus Schicht 4 verbunden waren, das die gerade veränderte Fuzzy–Menge $\widetilde{B}_{j*}$ repräsentiert. Als Gewicht wird dabei die neue Fuzzy–Menge verwendet. Gleichzeitig werden die ursprünglichen Verbindungen der betroffenen Neuronen aus Schicht 3 mit dem Neuron, das die alte Fuzzy–Menge $\widetilde{B}_{j*}$ repräsentiert, gelöscht. Allerdings werden bei diesen Änderungen nur diejenigen Neuronen aus Schicht 3 berücksichtigt, deren Ausgabe über einem Schwellenwert liegt.

Im anderen Fall gibt es zwei Möglichkeiten:

a) die ähnlichste Menge ist nicht die aktuelle Menge $\widetilde{B}_{j*}$, sondern eine andere Menge $\widetilde{B}^x_{j*}$. In diesem Fall werden die Verbindungen von Schicht 3 zu dem Neuron, das die Menge $\widetilde{B}_{j*}$ repräsentiert, umgehängt zu dem Neuron, das $\widetilde{B}^x_{j*}$ repräsentiert. Auch hierfür werden nur die Neuronen aus Schicht 3 berücksichtigt, deren Ausgabe über einem Schwellenwert liegt.

b) wenn die ähnlichste Menge die aktuelle Menge $\widetilde{B}_{j*}$ ist, wird diese Menge verändert zu $\widetilde{B}^{neu}_{j*}$. Ein Umhängen von Verbindungen findet nicht statt.

Durch dieses Verfahren können bei Bedarf beliebig viele neue Ausgabe–Fuzzy–Mengen erzeugt werden, ohne die vorhandenen Fuzzy–Mengen zu verändern. Dagegen würde der Backpropagation–Algorithmus als einziges Verfahren, falls die vor dem Training festgelegte Anzahl der Ausgabe–Fuzzy–Mengen nicht ausreichend ist, nur deren Werte wiederholt ändern. Ein erfolgreiches Training wäre so nicht möglich.

Ein Nachteil dieses Verfahrens ist, daß keine Fuzzy–Mengen gelöscht werden können. Da die neu erzeugten Fuzzy–Mengen $\widetilde{B}^{neu}_{j*}$ nur mit den bereits im Netz vorhandenen Fuzzy–Mengen, nicht jedoch untereinander, verglichen werden, steigt die Anzahl der Mengen in einem Durchgang oftmals stark an. Eventuell werden dabei Fuzzy–Mengen erzeugt, die eigentlich für eine korrekte Fuzzy–Regelung nicht notwendig sind. Ein weiterer Nachteil ist, daß es keine Möglichkeit gibt, auch neue Eingabe–Fuzzy–Mengen zu generieren oder neue Regeln zu erstellen. Falls bei der Initialisierung des Netzes weniger Eingabe–Fuzzy–Mengen oder Regeln verwendet werden, als unbedingt erforderlich sind, ist das System daher nicht in der Lage, einen korrekt funktionierenden Fuzzy–Controller zu erstellen.

Eine erweiterte Darstellung dieses Verfahrens sowie ein ausführliches Beispiel befindet sich in [Zinth, 1996]. Eine genauere Bewertung dieses Verfahrens sowie ein Vergleich zu den anderen vorgestellten Optimierungs–Systemen und ein weiteres Beispiel zu diesem Verfahren (Beispiel 7.1) werden in Kapitel 7 vorgestellt.

Kapitel 5

Das NEFCON–Modell

Beim NEFCON–Modell (NEural Fuzzy CONtroller) (s. [Nauck, 1996]) handelt es sich um einen *hybriden neuronalen Fuzzy–Regler*. D.h. es wird ein spezielles neuronales Netz zur Repräsentation eines Fuzzy–Controllers nach Mamdani verwendet. Dabei kommen zwei unterschiedliche Lernverfahren zum Einsatz, um die verwendeten Fuzzy–Mengen bzw. die Regeln zu trainieren. Für beide Verfahren wird ein *Fuzzy–Fehlermaß* verwendet.

5.1 Ein Fuzzy–Fehlermaß

Ein zu regelndes System $\mathcal{S}$ ist im optimalen Zustand, wenn alle Meßgrößen genau den gewünschten Wert haben. Normalerweise ist ein System aber auch in einem "guten" Zustand, wenn die gewünschten Werte ungefähr angenommen werden. Daher bietet sich die Verwendung von Fuzzy–Mengen zur Bewertung der Güte eines Systems und zur Bestimmung des Fehlers an.

Für ein System gibt es zwei unterschiedliche Arten von "guten" Zuständen:

- Sämtliche Meßwerte haben ungefähr die gewünschten Werte angenommen, d.h. das System ist insgesamt im optimalen Zustand. Am System muß nichts verändert werden.
- Es haben nicht sämtliche Meßwerte ungefähr die gewünschten Werte, jedoch ist aufgrund der gemessenen Abweichungen von den gewünschten Werten eine Entwicklung in Richtung dieser Werte zu erwarten. Dies ist z.B. der Fall, wenn in einem Ofen die Temperatur zu gering ist, gleichzeitig aber die Gaszufuhr sehr hoch ist. In diesem Zustand ist zu erwarten, daß der Ofen sich aufheizen wird. D.h. die im Vergleich zu den gewünschten Werten geringere Temperatur und höhere Gaszufuhr gleichen sich aus. Dieses wird als kompensatorische Situation bezeichnet.

Definition 5.1
Sei $\mathcal{S}$ ein zu regelndes System mit n Meßgrößen $z_1 \in X_1, \ldots, z_n \in X_n$.

Seien die optimalen Werte der Meßgrößen durch die Fuzzy–Mengen

$\widetilde{A}_1^{opt}$ auf X_1, ..., $\widetilde{A}_n^{opt}$ auf X_n repräsentiert.

Weiterhin seien für das System s kompensatorische Situationen bekannt, die durch Fuzzy–Relationen

$\widetilde{R}_i^{komp} \subset X_1 \times \ldots \times X_n$, $i = 1, \ldots, s$, repräsentiert werden.

Dann ist die Fuzzy–Güte G des Systems bei gegebenen Meßwerten $r_1, \ldots, r_n$ definiert durch:

$$G : X_1 \times \ldots \times X_n \longrightarrow [0,1]$$

$$G(r_1, \ldots, r_n) = g\big(G_{opt}(r_1, \ldots, r_n), G_{komp}(r_1, \ldots, r_n)\big)$$

mit:

$$G_{opt}(r_1, \ldots, r_n) = \widetilde{\delta}\big(\mu_{\widetilde{A}_1^{opt}}(r_1) \wedge \ldots \wedge \mu_{\widetilde{A}_n^{opt}}(r_n)\big)$$

$$G_{komp}(r_1, \ldots, r_n) = \widetilde{\delta}\big(\mu_{\widetilde{R}_1^{komp}}(r_1, \ldots, r_n) \wedge \ldots \wedge \mu_{\widetilde{R}_s^{komp}}(r_1, \ldots, r_n)\big)$$

Wobei $\wedge$ eine T–Norm ist, z.B. das Minimum, und g eine Verknüpfung der einzelnen Gütemaße G_{opt} (Güte der Meßwerte) und G_{komp} (Güte der Kompensation) durchführt, z.B. Auswahl eines der Werte.

Der *Fuzzy–Fehler E* wird nun definiert durch:

Definition 5.2 *Fuzzy–Fehler*

$$E(r_1, \ldots, r_n) = 1 - G(r_1, \ldots, r_n)$$

Ein Lernverfahren, das auf diesem Fuzzy–Fehlermaß basiert, heißt *Fuzzy–Fehler–Propagierung*. Der Ablauf ist ähnlich dem Backpropagation–Verfahren. Es wird ein verstärkendes Lernen angewendet, um den Einfluß "guter" Regeln zu verstärken und den Einfluß "schlechter" Regeln zu schwächen.

5.2 Aufbau und Arbeitsweise

Ein *NEFCON–System* ist ein dreischichtiges neuronales Netz, welches einen Fuzzy–Controller repräsentieren kann (s. Abb. 5.1). Der wesentliche Unterschied zu einem herkömmlichen neuronalen Netz ist, daß als Gewichte für die Verbindungen statt reeller Zahlen Fuzzy–Mengen verwendet werden. Das Netz ist wie folgt aufgebaut:

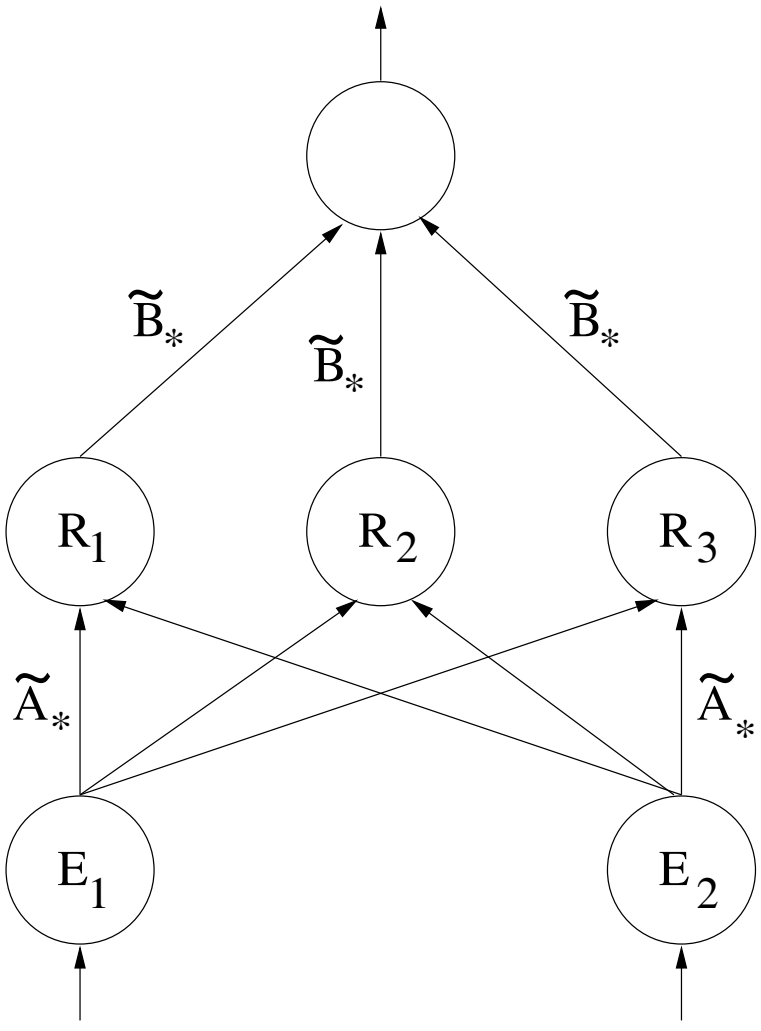

Abbildung 5.1: Aufbau eines NEFCON–Systems

Sei $\mathcal{S}$ ein zu regelndes System mit n Meßgrößen und einer Stellgröße (ein System für mehrere Stellgrößen ist nicht vorgesehen). Zur Regelung seien r linguistische Regeln formuliert. Dann hat das Netz folgende Struktur:

Die Eingabeschicht besteht aus n Neuronen, die nur ihren Eingabewert weiterleiten. In Schicht 2, der *Regelschicht*, gibt es für jede Regel ein Neuron. Die Verbindungen von Schicht 1 sind jeweils mit einem linguistischem Term benannt. Dabei werden für die Verbindungen der Neuronen $E_1, \ldots, E_n$ aus Schicht 1 zu Neuron R_k aus Schicht 2 gerade die in der Prämisse von Regel R_k verwendeten linguistischen Terme genommen.

Schicht 3 besteht nur aus einem Ausgabe–Neuron. Die Verbindung von jedem Neuron R_k aus Schicht 2 ist mit dem linguistischen Term der Konklusion von Regel R_k benannt. Die Struktur des Netzes repräsentiert somit die Regelbasis eines Fuzzy–Controllers.

Beispiel 5.1 *Hat Regel R_3 die Gestalt*

IF $x_1 =$ *mittel UND* $x_2 =$ *hoch THEN* $y =$ *schwach*,

gibt es folgende Verbindungen zum Neuron R_3 aus Schicht 2 (s. Abb. 5.2):

- *die Verbindung von Neuron E_1 aus Schicht 1 wird mit* `mittel` *benannt, die Verbindung von Neuron E_2 wird mit* `hoch` *benannt. Als Gewicht wird die Fuzzy–Menge für* `mittel` *bzw.* `hoch` *verwendet.*
- *Die Verbindung zum Ausgabe–Neuron wird mit* `schwach` *benannt und bekommt als Gewicht die Fuzzy–Menge, die den linguistischen Term* `schwach` *repräsentiert.*

Es gibt in einer bestimmten Konstellation der Meßgrößen nur einen richtigen Wert für die Stellgröße. Da die Prämisse einer Regel für eine bestimmte Situation steht, gibt es keine zwei verschiedenen Regeln mit gleicher Prämisse. Voraussetzung für die Anwendung der Lernverfahren ist, daß die Zugehörigkeitsfunktionen der Fuzzy–Mengen des Ausgabe–Raumes über ihrem Träger monoton sind. In diesem Fall existiert die Umkehrfunktion, die für die Verfahren benötigt wird. Daher werden ausschließlich Fuzzy–Mengen mit folgender Gestalt verwendet:

Auf den Eingabe–Dimensionen: Dreiecks–Mengen $\widetilde{A} = (l, m, r)$ mit:

$$\mu_{\widetilde{A}}(x) = \begin{cases} \frac{x-l}{m-l} : & x \in [l, m] \\ \frac{r-x}{r-m} : & x \in]m, r] \\ 0 : & \text{sonst} \end{cases}$$

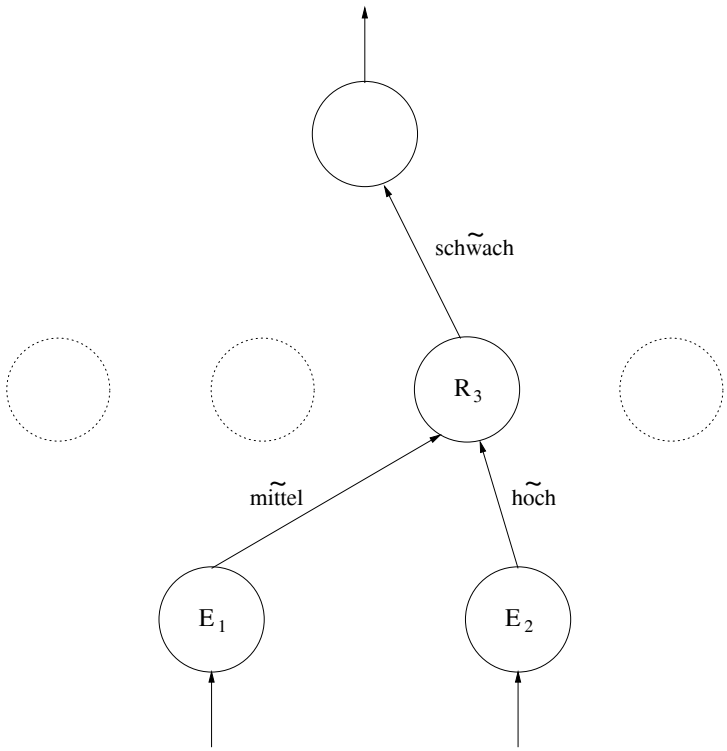

Abbildung 5.2: Regel 3 repräsentierende Verbindungen

Auf dem Ausgaberaum: *Zacken–Mengen* ("halbiertes Dreieck"):

Definition 5.3 *Eine Zacken–Menge ist eine Fuzzy–Menge* $\widetilde{B} = (m, b)$ *mit:*

$$\mu_{\widetilde{B}}(x) = \begin{cases} \frac{x-b}{m-b} : & x \in [b, m],\ falls\ b < m \\ \frac{b-x}{b-m} : & x \in [m, b],\ falls\ m < b \\ 0 : & sonst \end{cases}$$

Beispiel 5.2

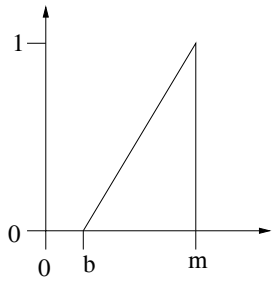

Abbildung 5.3: Eine Zacken–Menge

Die Berechnung der Netzausgabe entspricht der Berechnung der Ausgabe eines Fuzzy–Controllers (vgl. Kapitel 3). In Schicht 1 gibt jedes Neuron seine (reelle) Eingabe wieder aus. In Schicht 2 berechnet jedes Neuron R_k den Erfüllungsgrad der Prämisse von Regel R_k. D.h. die Ausgabe o_k wird berechnet durch

$$o_k = \widetilde{\delta}\big(\mu_{\widetilde{A}_{1*}}(o_{E_1}) \wedge \ldots \wedge \mu_{\widetilde{A}_{n*}}(o_{E_n})\big)$$

mit $o_{E_1}, \ldots, o_{E_n}$ den Ausgaben der Neuronen aus Schicht 1 und $\widetilde{A}_{1*}, \ldots, \widetilde{A}_{n*}$ den Gewichten der Verbindungen von $E_1, \ldots, E_n$ zu R_k, sowie $\wedge$ einer T–Norm.

Im Ausgabe–Neuron wird nun die Ausgabe o wie folgt berechnet:

$$o = \frac{\sum\limits_{R_k \in S_2} o_k \cdot \mu^{-1}_{\widetilde{B}_{k*}}(o_k)}{\sum\limits_{R_k \in S_2} o_k}$$

mit $\widetilde{B}_{k*}$ dem Gewicht der Verbindung von Neuron R_k aus Schicht 2 (S_2) zum Ausgabe-Neuron.

Das Netz verhält sich nun so, wie der Fuzzy–Controller, den es repräsentiert. Im Gegensatz zu einem Fuzzy–Controller gibt es jedoch die Möglichkeit, Lernverfahren anzuwenden. Beim NEFCON–Modell werden zwei unterschiedliche Lernverfahren eingesetzt, von denen eines die vorhandenen Fuzzy–Mengen optimiert, und das andere unter Verwendung vorhandener Fuzzy–Mengen eine Regelbasis erzeugt.

5.3 Anpassung der Fuzzy–Mengen

Die Anpassung der Fuzzy–Mengen erfolgt durch das Verfahren der Fuzzy–Fehler–Propagierung. Dabei wird vorausgesetzt, daß die Regelbasis ausreichend gut erstellt wurde. Wenn die Regelbasis korrekt ist, sind die Fuzzy–Mengen für ein fehlerhaftes Verhalten des Fuzzy–Controllers verantwortlich.

Die Fuzzy–Mengen sollen nun in Abhängigkeit vom Fuzzy–Fehlermaß verändert werden. Da der Einfluß einer Regel auf das Ergebnis vom Erfüllungsgrad ihrer Prämisse abhängt, wird dieser bei der Änderung berücksichtigt. Fuzzy–Mengen aus Regeln mit Erfüllungsgrad 0 werden gar nicht geändert. Es findet ein verstärkendes Lernen statt, das “gute“ Regeln für ihren Einfluß belohnen und “schlechte“ Regeln bestrafen soll.

Zur Bewertung der Regeln muß zunächst einmal festgestellt werden, welchen Beitrag t_k eine Regel R_k zum Ergebnis beisteuert. Aufgrund der Existenz der Umkehrfunktion

bei den verwendeten Ausgabe–Fuzzy–Mengen und der Berechnung der Ausgabe o läßt sich dieser wie folgt berechnen:

Sei o_k die Ausgabe von Neuron R_k (entsprechend dem Erfüllungsgrad der Prämisse von Regel R_k). Sei $\widetilde{B}_{k*}$ das Gewicht der Verbindung von R_k zum Ausgabe–Neuron (entsprechend der Fuzzy–Menge der Konklusion von Regel R_k). Dann gilt:

$$t_k = \mu^{-1}_{\widetilde{B}_{k*}}(o_k)$$

Der Fuzzy–Fehler bewertet den gesamten Zustand des zu steuernden Systems (vgl. Abschnitt 5.1). Um den Einfluß einer einzelnen Regel auf das System als "gut" oder "schlecht" bewerten zu können, müssen daher zusätzlich zum Fuzzy–Fehler weitere Informationen über das richtige Ergebnis bekannt sein. Unter der Voraussetzung, daß (evtl. nach Normierung) die Stellgröße im optimalen Systemzustand den Wert 0 hat, kann für eine gegebene Situation entschieden werden, welches Vorzeichen die Stellgröße haben muß. Der Beitrag t_k von Regel R_k wird dann als "gut" bewertet, wenn t_k das richtige Vorzeichen hat, andernfalls als "schlecht".

Nun soll der Einfluß der "guten" Regeln auf das Ergebnis verstärkt werden, der Einfluß "schlechter" Regeln soll abgeschwächt werden. Der Einfluß einer Regel erhöht sich, wenn der Erfüllungsgrad ihrer Prämisse größer wird. Dies wird durch Verbreiterung des Trägers der in der Prämisse verwendeten Fuzzy–Mengen erreicht. Außerdem erhöht sich der Einfluß einer Regel R_k, wenn der Betrag ihres Beitrags $|t_k|$ vergrößert wird. Dies wird durch Verschmälerung des Trägers der Fuzzy–Menge der Konklusion erreicht, da t_k mit der Umkehrfunktion berechnet wird. Für eine Verringerung des Beitrags einer Regel werden die gegenteiligen Aktionen durchgeführt.

Der Lernalgorithmus zur Anpassung der Fuzzy–Mengen besteht aus folgenden Schritten, die solange wiederholt werden, bis ein Abbruchkriterium erfüllt ist. Dieses ist z.B., daß der Fuzzy–Fehler E für eine bestimmte Anzahl von Durchgängen unterhalb einer Schranke liegt. Sei $\mathcal{S}$ ein zu regelndes System mit n Meßgrößen und einer Stellgröße, für das r Regeln formuliert sind:

NEFCON–Lernalgorithmus 1:

1. Ausgabe o zu aktuellen Meßwerten berechnen, o auf das System anwenden, neue Meßwerte berechnen

2. Fuzzy–Fehler E aus dem neuen Systemzustand berechnen

3. Vorzeichen des richtigen Stellwerts im neuen Systemzustand bestimmen

4. Für jede Regel R_k Beitrag t_k zur Ausgabe berechnen. Dann das *Fehlersignal* F_k jeder Regel R_k berechnen durch:

$$F_k := \begin{cases} -o_k \cdot E: & \text{Vorzeichen von } t_k \text{ richtig} \\ o_k \cdot E: & \text{Vorzeichen von } t_k \text{ falsch} \end{cases}$$

5. Für alle Regeln: ändern der Eingabe–Fuzzy–Mengen $\widetilde{A}_{i*}^{(k)} = (l_{i*}^{(k)}, m_{i*}^{(k)}, r_{i*}^{(k)})$, die bei Regel R_k verwendet werden, nach folgender Formel:

$$\Delta l_{i*}^{(k)} = -\eta \cdot F_k \cdot (m_{i*}^{(k)} - l_{i*}^{(k)})$$

$$\Delta r_{i*}^{(k)} = \eta \cdot F_k \cdot (r_{i*}^{(k)} - m_{i*}^{(k)})$$

6. Für alle Regeln: ändern der Ausgabe–Fuzzy–Mengen $\widetilde{B}_{j*}^{(k)} = (m_{j*}^{(k)}, b_{j*}^{(k)})$, die bei Regel R_k verwendet werden, nach folgender Formel:

$$\Delta b_{j*}^{(k)} = \begin{cases} \eta \cdot F_k \cdot (b_{j*}^{(k)} - m_{j*}^{(k)}): & b_{j*}^{(k)} > m_{j*}^{(k)} \\ \eta \cdot F_k \cdot (m_{j*}^{(k)} - b_{j*}^{(k)}): & b_{j*}^{(k)} < m_{j*}^{(k)} \end{cases}$$

mit einer Lernrate $\eta > 0$.

Beispiel 5.3 *Lernalgorithmus 1 wurde am Beispiel des inversen Pendels (s. Abschnitt 3.14) getestet. Als Wertebereich für den Winkel ϑ wurde $[-90, 90]$ festgelegt, für die Winkelgeschwindigkeit $\dot{\vartheta}$ $[-200, 200]$ und für die Kraft F $[-25, 25]$.*

Da die Ausgabe–Fuzzy–Mengen für die Anwendung des Lernalgorithmus monoton sein müssen, wird aus Symmetriegründen bei jeder Partitionierung die Fuzzy–Menge **ungefähr null (un)** *durch die Mengen* **negativ null (nn)** *und* **positiv null (pn)** *ersetzt. Folgende Regelbasis wird verwendet:*

R_{01} : **IF** $x_1 =$ **ng** **UND** $x_2 =$ **ng** **THEN** $y =$ **ng**
R_{02} : **IF** $x_1 =$ **nm** **UND** $x_2 =$ **ng** **THEN** $y =$ **ng**
R_{03} : **IF** $x_1 =$ **ng** **UND** $x_2 =$ **nm** **THEN** $y =$ **ng**
R_{04} : **IF** $x_1 =$ **nm** **UND** $x_2 =$ **nm** **THEN** $y =$ **nm**

R_{05} : *IF* $x_1 =$ *nk UND* $x_2 =$ *nm THEN* $y =$ *nm*
R_{06} : *IF* $x_1 =$ *nn UND* $x_2 =$ *nm THEN* $y =$ *nk*
R_{07} : *IF* $x_1 =$ *ng UND* $x_2 =$ *nk THEN* $y =$ *ng*
R_{08} : *IF* $x_1 =$ *nm UND* $x_2 =$ *nk THEN* $y =$ *nm*
R_{09} : *IF* $x_1 =$ *nk UND* $x_2 =$ *nk THEN* $y =$ *pk*
R_{10} : *IF* $x_1 =$ *nn UND* $x_2 =$ *nk THEN* $y =$ *nn*

R_{11} : *IF* $x_1 =$ *nm UND* $x_2 =$ *nn THEN* $y =$ *nk*
R_{12} : *IF* $x_1 =$ *nk UND* $x_2 =$ *nn THEN* $y =$ *nk*
R_{13} : *IF* $x_1 =$ *nn UND* $x_2 =$ *nn THEN* $y =$ *nn*
R_{14} : *IF* $x_1 =$ *pn UND* $x_2 =$ *pn THEN* $y =$ *pn*
R_{15} : *IF* $x_1 =$ *pk UND* $x_2 =$ *pn THEN* $y =$ *pk*
R_{16} : *IF* $x_1 =$ *pm UND* $x_2 =$ *pn THEN* $y =$ *pk*
R_{17} : *IF* $x_1 =$ *pn UND* $x_2 =$ *pk THEN* $y =$ *pn*
R_{18} : *IF* $x_1 =$ *pk UND* $x_2 =$ *pk THEN* $y =$ *nk*
R_{19} : *IF* $x_1 =$ *pm UND* $x_2 =$ *pk THEN* $y =$ *pm*
R_{20} : *IF* $x_1 =$ *pg UND* $x_2 =$ *pk THEN* $y =$ *pg*

R_{21} : *IF* $x_1 =$ *pn UND* $x_2 =$ *pm THEN* $y =$ *pk*
R_{22} : *IF* $x_1 =$ *pk UND* $x_2 =$ *pm THEN* $y =$ *pm*
R_{23} : *IF* $x_1 =$ *pm UND* $x_2 =$ *pm THEN* $y =$ *pm*
R_{24} : *IF* $x_1 =$ *pg UND* $x_2 =$ *pm THEN* $y =$ *pg*
R_{25} : *IF* $x_1 =$ *pm UND* $x_2 =$ *pg THEN* $y =$ *pg*
R_{26} : *IF* $x_1 =$ *pg UND* $x_2 =$ *pg THEN* $y =$ *pg*

Folgende Partitionierungen, mit denen die Steuerung versagt, wurden definiert (Dreiecks–Mengen auf X_1 und X_2, Zacken–Mengen auf Y):

auf X_1:

ng $\widehat{=}\widetilde{A}_{11} = (-90, -90, -70)$
nm $\widehat{=}\widetilde{A}_{12} = (-60, -50, -40)$
nk $\widehat{=}\widetilde{A}_{13} = (-30, -20, -10)$
nn $\widehat{=}\widetilde{A}_{14} = (-5, 0, 0)$
pn $\widehat{=}\widetilde{A}_{15} = (0, 0, 5)$
pk $\widehat{=}\widetilde{A}_{16} = (10, 20, 30)$
pm $\widehat{=}\widetilde{A}_{17} = (40, 50, 60)$
pg $\widehat{=}\widetilde{A}_{18} = (70, 90, 90)$

auf X_2:

ng $\widehat{=}\widetilde{A}_{21} = (-200, -200, -150)$
nm $\widehat{=}\widetilde{A}_{22} = (-140, -120, -100)$
nk $\widehat{=}\widetilde{A}_{23} = (-90, -70, -50)$

$\mathit{nn} \mathrel{\widehat{=}} \widetilde{A}_{24} = (-40, 0, 0)$
$\mathit{pn} \mathrel{\widehat{=}} \widetilde{A}_{25} = (0, 0, 40)$
$\mathit{pk} \mathrel{\widehat{=}} \widetilde{A}_{26} = (50, 70, 90)$
$\mathit{pm} \mathrel{\widehat{=}} \widetilde{A}_{27} = (100, 120, 140)$
$\mathit{pg} \mathrel{\widehat{=}} \widetilde{A}_{28} = (150, 200, 200)$

auf Y:

$\mathit{ng} \mathrel{\widehat{=}} \widetilde{B}_1 = (-25, 0)$
$\mathit{nm} \mathrel{\widehat{=}} \widetilde{B}_2 = (-20, 0)$
$\mathit{nk} \mathrel{\widehat{=}} \widetilde{B}_3 = (-15, 0)$
$\mathit{nn} \mathrel{\widehat{=}} \widetilde{B}_4 = (0, -15)$
$\mathit{pn} \mathrel{\widehat{=}} \widetilde{B}_5 = (0, 15)$
$\mathit{pk} \mathrel{\widehat{=}} \widetilde{B}_6 = (15, 0)$
$\mathit{pm} \mathrel{\widehat{=}} \widetilde{B}_7 = (20, 0)$
$\mathit{pg} \mathrel{\widehat{=}} \widetilde{B}_8 = (25, 0)$

Nach dem Training ergaben sich folgende Partitionierungen, mit denen die Steuerung in jeder Situation funktioniert:

auf X_1:

$\mathit{ng} \mathrel{\widehat{=}} \widetilde{A}_{11} = (-90, -90, -70)$
$\mathit{nm} \mathrel{\widehat{=}} \widetilde{A}_{12} = (-90, -50, 0)$
$\mathit{nk} \mathrel{\widehat{=}} \widetilde{A}_{13} = (-90, -20, 0)$
$\mathit{nn} \mathrel{\widehat{=}} \widetilde{A}_{14} = (-10, 0, 0)$
$\mathit{pn} \mathrel{\widehat{=}} \widetilde{A}_{15} = (0, 0, 10)$
$\mathit{pk} \mathrel{\widehat{=}} \widetilde{A}_{16} = (0, 20, 90)$
$\mathit{pm} \mathrel{\widehat{=}} \widetilde{A}_{17} = (0, 50, 90)$
$\mathit{pg} \mathrel{\widehat{=}} \widetilde{A}_{18} = (70, 90, 90)$

auf X_2:

$\mathit{ng} \mathrel{\widehat{=}} \widetilde{A}_{21} = (-200, -200, -150)$
$\mathit{nm} \mathrel{\widehat{=}} \widetilde{A}_{22} = (-170, -120, -70)$
$\mathit{nk} \mathrel{\widehat{=}} \widetilde{A}_{23} = (-200, -70, 0)$
$\mathit{nn} \mathrel{\widehat{=}} \widetilde{A}_{24} = (-200, 0, 0)$
$\mathit{pn} \mathrel{\widehat{=}} \widetilde{A}_{25} = (0, 0, 200)$
$\mathit{pk} \mathrel{\widehat{=}} \widetilde{A}_{26} = (0, 70, 200)$
$\mathit{pm} \mathrel{\widehat{=}} \widetilde{A}_{27} = (45, 120, 200)$
$\mathit{pg} \mathrel{\widehat{=}} \widetilde{A}_{28} = (150, 200, 200)$

auf Y:

$ng \mathrel{\widehat{=}} \widetilde{B}_1 = (-25, -4)$
$nm \mathrel{\widehat{=}} \widetilde{B}_2 = (-20, -15)$
$nk \mathrel{\widehat{=}} \widetilde{B}_3 = (-15, -10)$
$nn \mathrel{\widehat{=}} \widetilde{B}_4 = (0, -10)$
$pn \mathrel{\widehat{=}} \widetilde{B}_5 = (0, 10)$
$pk \mathrel{\widehat{=}} \widetilde{B}_6 = (15, 10)$
$pm \mathrel{\widehat{=}} \widetilde{B}_7 = (20, 15)$
$pg \mathrel{\widehat{=}} \widetilde{B}_8 = (25, 2)$

Mit dem NEFCON–Lernalgorithmus 1 lassen sich die vorhandenen Fuzzy–Mengen korrekt einstellen, vorausgesetzt, eine korrekte Regelbasis wurde zuvor definiert. Die Erstellung der Regelbasis unter Verwendung der vorhandenen (noch nicht optimierten) Fuzzy–Mengen erfolgt entweder vom Anwender, oder —falls dies nicht möglich oder nicht gewünscht ist— mit Hilfe des im folgenden Abschnitt vorgestellten NEFCON–Lernalgorithmus 2.

5.4 Erlernen einer Regelbasis

Wenn kein ausreichendes Expertenwissen für ein zu regelndes System verfügbar ist, läßt sich vom Anwender keine geeignete Regelbasis für einen Fuzzy–Controller erstellen. Für das NEFCON–System gibt es ein Lernverfahren, das in diesem Fall automatisch eine vollständige Regelbasis erzeugt.

Voraussetzung für die Anwendung dieses Verfahrens ist, daß zumindest einigermaßen geeignete Fuzzy–Mengen für die Eingaberäume und den Ausgaberaum definiert sind. Dabei muß insbesondere die Anzahl der definierten Fuzzy–Mengen korrekt sein. Eine genaue Anpassung der Fuzzy–Mengen (d.h. Feinabstimmung) erfolgt dann im Anschluß an das Erlernen der Regelbasis mit der in Abschnitt 5.3 beschriebenen Methode. Zusätzlich muß, wie beim Lernverfahren zur Anpassung der Fuzzy–Mengen, das richtige Vorzeichen der Stellgröße bekannt sein.

Die Idee bei dem verwendeten Verfahren ist es, zunächst alle (!) Regeln zu erstellen, die mit den zuvor vom Anwender definierten Fuzzy–Mengen erzeugt werden können. Sind für die Eingabe–Dimensionen $X_1, \ldots, X_n$ jeweils $p_i, i = 1, \ldots, n$, Fuzzy–Mengen gegeben, und für den Ausgaberaum Y q Fuzzy–Mengen, so lassen sich damit $N = q \cdot \prod_{i=1}^{n} p_i$ verschiedene Regeln bilden. Für die Prämisse wird jede mögliche Kombination von Fuzzy–Mengen auf den Eingabe–Dimensionen verwendet, als Konklusion wird dazu jeweils jede gegebene Fuzzy–Menge auf dem Ausgaberaum eingesetzt. Von diesen Regeln werden nun iterativ die falschen und überflüssigen entfernt, bis eine geeignete Regelbasis übrig bleibt.

Dieser Vorgang wird in zwei Phasen durchgeführt. In der ersten Phase werden jeweils alle Regeln entfernt, deren Beitrag zum Ergebnis das falsche Vorzeichen hat. In der zweiten Phase werden von den verbliebenen Regeln jeweils alle mit gleicher Prämisse zu einer Menge von Regeln zusammengefaßt. Aus jeder dieser Mengen wird dann pro Durchgang eine Regel ausgewählt, die zur Berechnung des Ergebnisses verwendet wird. Danach wird der Fehleranteil jeder verwendeten Regel gespeichert und aufsummiert. Anschließend wird aus jeder Menge die Regel mit dem geringsten Fehleranteil ausgewählt. Die anderen Regeln werden gelöscht, ebenso Regeln, die nur selten *aktiv* sind.

Definition 5.4 *Eine Regel heißt aktiv, wenn der Erfüllungsgrad ihrer Prämisse, bzw. die Ausgabe o_k des entsprechenden Neurons R_k, größer als null ist.*

Im einzelnen besteht dieses Verfahren der *erweiterten Fuzzy–Fehler–Propagierung* aus folgenden Schritten. Sei $\mathcal{S}$ ein zu regelndes System mit n Meßgrößen und einer Stellgröße. Seien bei gegebenen Eingabe– und Ausgabe–Partitionierungen alle N möglichen Regeln erstellt:

NEFCON–Lernalgorithmus 2:

Phase 1:

Für jede Regel R_k, $k = 1, \ldots, N$, wird ein Zähler C_k definiert, der zählt, wie oft eine Regel aktiv ist. Folgende Schritte werden m_1 mal wiederholt:

1. Ausgabe o zu aktuellen Meßwerten berechnen
2. Für jede Regel R_k Beitrag t_k zur Ausgabe berechnen
3. Vorzeichen des richtigen Stellwerts im aktuellen Zustand bestimmen
4. Alle Regeln R_k mit falschem Vorzeichen entfernen, N entsprechend verringern
5. Für jede Regel R_k mit Ausgabe $o_k > 0$ den Zähler C_k um 1 erhöhen
6. Ergebnis o auf das System anwenden und neue Meßwerte bestimmen

Phase 2:

Die noch verbleibenden Regeln werden in Klassen $\mathcal{R}_p$ von Regeln mit gleicher Prämisse aufgeteilt. Für jede Regel R_k wird ein Zähler Z_k für den Fehleranteil definiert. Die Zähler C_k werden unverändert übernommen. Folgende Schritte werden m_2 mal wiederholt:

1. aus jeder Klasse $\mathcal{R}_p$ eine beliebige Regel R_{k_p} auswählen
2. mit den ausgewählten Regeln und den aktuellen Meßwerten das Ergebnis o berechnen
3. o auf das System anwenden und danach die neuen Meßwerte ermitteln
4. für jede verwendete Regel R_{k_p} Beitrag t_{k_p} zur Ausgabe berechnen
5. Vorzeichen des richtigen Stellwerts im neuen Systemzustand bestimmen
6. Fehlersignal F_{k_p} jeder verwendeten Regel R_{k_p} berechnen (s. Lernalgorithmus 1) und zu ihrem Zähler Z_{k_p} dazu addieren
7. für jede verwendete Regel R_{k_p} mit Neuron–Ausgabe $o_{k_p} > 0$ den Zähler C_{k_p} um 1 erhöhen

Anschließend aus jeder Klasse $\mathcal{R}_p$ eine Regel R_{k_p} auswählen, für die Z_{k_p} minimal ist. Alle anderen Regeln dieser Klasse löschen. Außerdem alle Regeln R_k löschen, für die $C_k < \frac{m_1+m_2}{\beta}$ ist, mit einem $\beta \geq 1$ und $m_1, m_2 \in \mathbb{N}$. Zuletzt N auf den aktuellen Stand bringen.

Beispiel 5.4 *Lernalgorithmus 2 wurde ebenfalls am Beispiel des inversen Pendels getestet. Dabei wurden als geeignete Partitionierungen die in Beispiel 1 optimierten Fuzzy–Mengen verwendet. Nach dem Training ergab sich folgende Regelbasis, mit der die Steuerung in jeder Situation funktioniert:*

R_{01} : *IF* $x_1 =$ *ng UND* $x_2 =$ *ng THEN* $y =$ *ng*
R_{02} : *IF* $x_1 =$ *nm UND* $x_2 =$ *ng THEN* $y =$ *ng*
R_{03} : *IF* $x_1 =$ *nk UND* $x_2 =$ *ng THEN* $y =$ *ng*
R_{04} : *IF* $x_1 =$ *nm UND* $x_2 =$ *nm THEN* $y =$ *nk*
R_{05} : *IF* $x_1 =$ *nk UND* $x_2 =$ *nm THEN* $y =$ *nk*
R_{06} : *IF* $x_1 =$ *nn UND* $x_2 =$ *nm THEN* $y =$ *nm*
R_{07} : *IF* $x_1 =$ *pn UND* $x_2 =$ *nm THEN* $y =$ *nk*
R_{08} : *IF* $x_1 =$ *pk UND* $x_2 =$ *nm THEN* $y =$ *ng*
R_{09} : *IF* $x_1 =$ *pm UND* $x_2 =$ *nm THEN* $y =$ *ng*
R_{10} : *IF* $x_1 =$ *ng UND* $x_2 =$ *nk THEN* $y =$ *ng*

R_{11} : *IF* $x_1 =$ *nm UND* $x_2 =$ *nk THEN* $y =$ *nm*
R_{12} : *IF* $x_1 =$ *nk UND* $x_2 =$ *nk THEN* $y =$ *nn*
R_{13} : *IF* $x_1 =$ *nn UND* $x_2 =$ *nk THEN* $y =$ *nk*
R_{14} : *IF* $x_1 =$ *ng UND* $x_2 =$ *nn THEN* $y =$ *ng*
R_{15} : *IF* $x_1 =$ *nm UND* $x_2 =$ *nn THEN* $y =$ *ng*
R_{16} : *IF* $x_1 =$ *nk UND* $x_2 =$ *nn THEN* $y =$ *ng*
R_{17} : *IF* $x_1 =$ *nn UND* $x_2 =$ *nn THEN* $y =$ *nn*
R_{18} : *IF* $x_1 =$ *pn UND* $x_2 =$ *pn THEN* $y =$ *pn*
R_{19} : *IF* $x_1 =$ *pk UND* $x_2 =$ *pn THEN* $y =$ *pm*

R_{20} : *IF* $x_1 =$ *pm UND* $x_2 =$ *pn THEN* $y =$ *pk*
R_{21} : *IF* $x_1 =$ *pn UND* $x_2 =$ *pk THEN* $y =$ *pn*
R_{22} : *IF* $x_1 =$ *pk UND* $x_2 =$ *pk THEN* $y =$ *pk*
R_{23} : *IF* $x_1 =$ *pm UND* $x_2 =$ *pk THEN* $y =$ *pg*
R_{24} : *IF* $x_1 =$ *pn UND* $x_2 =$ *pm THEN* $y =$ *pg*
R_{25} : *IF* $x_1 =$ *pk UND* $x_2 =$ *pm THEN* $y =$ *pk*
R_{26} : *IF* $x_1 =$ *pm UND* $x_2 =$ *pm THEN* $y =$ *pg*

Für die Güte der mit diesem Verfahren erzeugten Regelbasis sind verschiedene Faktoren verantwortlich. Die Parameter m_1, m_2 und β müssen geeignet gewählt werden. Es ist wichtig, die beiden Phasen ausreichend lange laufen zu lassen, je nach Situation z.B. in 2000 bis 3000 Durchgängen. Für β haben sich Werte zwischen 1.00 und 1.03 bewährt. Bei zu hohen Werten werden sonst eventuell selten gebrauchte, aber für Ausnahmesituationen wichtige Regeln gelöscht. Außerdem sollte das System während des Lernvorgangs alle möglichen typischen Zustände durchlaufen. In jedem Fall ist eine automatisch erzeugte Regelbasis nur als ein erster Vorschlag zu betrachten, der noch überprüft und verbessert werden sollte. Wenn z.B. eine generierte Regel intuitiv als ungünstig erscheint, ist sie nachträglich durch eine geeignetere Regel zu ersetzen.

Es ist auch möglich, diesen Lernalgorithmus mit vorhandenem Teil–Wissen zu kombinieren. Wenn etwa für eine bestimmte Situation schon eine passende Regel bekannt ist, kann sie im voraus erzeugt werden. Dem Lernalgorithmus wird dann verboten, diese Regel zu entfernen, andere Regeln mit der gleichen Prämisse werden gar nicht erst erzeugt. Wenn dagegen bekannt ist, daß bestimmte Folgerungen für eine Situation falsch sind, wird die Regelmenge $\mathcal{R}_p$ für diese Situation (Prämisse) ohne Regeln mit Konklusionen angelegt, die für diese Folgerungen stehen.

Beide Lernverfahren wurden mit verschiedenen Einstellungen erfolgreich am Beispiel des inversen Pendels getestet. Lernalgorithmus 1 ist in der Lage, ungünstige Fuzzy–Mengen, bei denen eine Steuerung nicht möglich ist, in geeignete Fuzzy–Mengen zu transformieren. Lernalgorithmus 2 erzeugt eine Regelbasis, mit der das Pendel auch in künstlich erzeugten Extremsituationen sicher zu steuern ist. Eine genauere Darstellung dieser Tests findet sich in [Nauck, 1996].

Ein Nachteil des NEFCON–Systems ist die Voraussetzung, daß die Zugehörigkeitsfunktionen der Fuzzy–Mengen des Ausgaberaumes auf ihrem Träger monoton sein müssen, wodurch die Auswahl der verwendeten Fuzzy–Mengen eingeschränkt wird. Das ist ungünstig, da speziell die nicht monotonen Dreiecks–Mengen und Gauß–Mengen häufig für Fuzzy–Controller verwendet werden. Fuzzy–Controller, die Mengen dieser Typen verwenden, sind grundsätzlich für Optimierungen mit dem NEFCON–System nicht geeignet. Ein weiterer Nachteil ist die Festlegung auf nur einen Ausgabewert. Daher ist es nicht ohne weiteres möglich, einen beliebigen, bereits erstellten Fuzzy–Controller auf das NEFCON–System zu übertragen und optimieren zu lassen.

Weiterhin fehlt z.B. die Möglichkeit, vorhandene Regeln zu überprüfen und gegebenenfalls zu korrigieren, oder fehlende Fuzzy–Mengen erzeugen zu lassen. Daher ist ein sinnvoller Einsatz nur für Fuzzy–Controller möglich, die die genannten Voraussetzungen erfüllen (Monotonie, ein Ausgabewert), wobei zumindest die Anzahl der benötigten Fuzzy–Mengen genau bekannt sein muß. Falls eine für die korrekte Steuerung notwendige Fuzzy–Menge vergessen wurde, kann das NEFCON–System sie nicht erzeugen und daher mit den vorgestellten Verfahren keinen in jeder Situation funktionierenden Fuzzy–Controller erstellen.

Eine genauere Bewertung dieses Verfahrens sowie ein Vergleich zu den anderen vorgestellten Optimierungs–Systemen und ein weiteres Beispiel zu diesem Verfahren (Beispiel 7.2) werden in Kapitel 7 vorgestellt.

Kapitel 6

MFOS: Münsteraner–Fuzzy–Optimierungs–System

In diesem Kapitel wird ein Modell beschrieben, mit dem es möglich ist, einen potentiell beliebigen Fuzzy–Controller nach Mamdani in ein funktional äquivalentes neuronales Netz zu transformieren. Die Komponenten des Fuzzy–Controllers lassen sich danach mit unterschiedlichen Lernverfahren individuell entsprechend den Forderungen des Users optimieren. Anschließend ist eine Rücktransformation in einen separaten, optimierten Fuzzy–Controller möglich. Eine Version dieses Modells für Sugeno–Controller wird in Abschnitt 9 beschrieben.

Bei der Entwicklung des Modells und der Lernverfahren wurde darauf geachtet, die von anderen Systemen dieser Art (s. Kapitel 4 und 5) bekannten Einschränkungen für den verwendeten Fuzzy–Controller und bei den Optimierungen zu vermeiden (s. Beispiel 6.10). Das Ziel war es, dem Anwender ein spezielles "Werkzeug" zur Verfügung zu stellen, um einen gegebenen Fuzzy–Controller in ein neuronales Netz umzuwandeln, ohne sich beim Entwurf etwa am NEFCON–Modell orientieren zu müssen, und dann einstellen zu können, welche Komponenten angepaßt werden sollen. Es ist z.B. möglich, fehlerhafte Regeln korrigieren zu lassen, nach Bedarf zusätzliche Fuzzy–Mengen zu erzeugen oder überflüssige Fuzzy–Mengen zu löschen etc. Damit gezielt die Bestandteile angepaßt werden können, die noch nicht genau bestimmt sind, gibt es bei der Auswahl der einzusetzenden Lernverfahren keine Vorgaben. Außerdem kann so festgelegt werden, daß keine neuen Regeln oder neue Partitions–Mengen erzeugt werden dürfen, falls deren Anzahl z.B. durch eine Spezialhardware o.ä. eingeschränkt ist. Die Optimierung gliedert sich in zwei Phasen, dem Pre–Tuning und dem Fine–Tuning.

6.1 Spezifikation des Fuzzy–Controllers

In Kapitel 3 wurde bereits der Nachteil von Fuzzy–Controllern beschrieben, daß sie keine adaptiven Systeme sind. Um diesen Nachteil zu beheben, gibt es verschiedene Ansätze, Fuzzy–Controller mit neuronalen Netzen zu kombinieren, z.B. das Verfah-

ren von Lin und Lee (s. Kapitel 4) und das NEFCON–Modell (s. Kaptitel 5). Die meisten dieser Systeme machen allerdings mehr oder weniger starke Voraussetzungen an den verwendeten Fuzzy–Controller. So wird etwa beim NEFCON–Modell verlangt, daß die Ausgabe–Partitions–Mengen auf ihrem Träger monoton sind. Aufgrund solcher Voraussetzungen und Vorgaben ist es nur eingeschränkt möglich, einen bereits vorhandenen Fuzzy–Controller mit einem dieser Systeme zu optimieren. Außerdem werden zumeist nur bestimmte, vorgegebene Bestandteile des Fuzzy–Controllers angepaßt. So ist z.B. das NEFCON–Modell nicht in der Lage, bei Bedarf neue Fuzzy–Mengen zu erzeugen (s. Kapitel 5). Das Verfahren von Lin und Lee (s. Kapitel 4) führt eine Anpassung der Fuzzy–Mengen und der Regeln durch. Dabei werden jedoch nur zusätzlich benötigte Fuzzy–Mengen auf dem *Ausgabe*–Raum generiert. D.h. schon das Vergessen einer benötigten *Eingabe*–Partitions–Menge führt dazu, daß ein korrektes Einstellen des Fuzzy–Controllers nicht möglich ist (s. Kapitel 4).

Um diese Einschränkungen zu vermeiden, wurde ein neuronales Netz entwickelt, welches einen Fuzzy–Controller mit endlicher Eingabe– und Ausgabe–Dimension in äquivalenter Weise simuliert. Die berechnete Netzausgabe entspricht exakt der Ausgabe des Fuzzy–Controllers. Weiterhin wurden diverse Lernverfahren entwickelt, mit denen sich die verschiedenen Bestandteile des Fuzzy–Controllers anpassen lassen. Dabei ist es möglich, nach dem Training des Netzes die optimierte Regelbasis und die optimierten Partitionierungen zu extrahieren und separat für einen unabhängigen Fuzzy–Controller zu nutzen. Alternativ läßt sich das System nach dem Training selber als Fuzzy–Controller einsetzen. Das Netz und sämtliche Lernverfahren wurden objektorientiert in C++ implementiert, wobei stets auf zukünftige Erweiterungsmöglichkeiten geachtet wurde.

Bemerkung 6.1 *Das vorgestellte System heißt MFOS–M:*
Münsteraner–Fuzzy–Optimierungs–System für Mamdani–Controller.

Die Fähigkeiten des MFOS–M beinhalten die Transformation des Fuzzy–Controllers in das funktional äquivalente neuronale Netz, die Optimierung des Netzes in zwei Phasen (Pre–Tuning und Fine–Tuning) und die Rücktransformation des optimierten Netzes in einen funktional äquivalenten Fuzzy–Controller.

Folgende Voraussetzungen an den Fuzzy–Controller wurden gemacht:

Die Prämisse jeder Regel wird mit UND verknüpft, und jede Regel bezieht sich in der Konklusion nur auf eine Ausgabe–Dimension. Aufgrund äquivalenter Umformungsmöglichkeiten stellt dies keine Beschränkung der Allgemeinheit dar (s. [Feuring, 1996]). Folgerungen für mehrere Ausgabe–Dimensionen werden durch mehrere Regeln mit der gleichen Prämisse realisiert. Für die Fuzzifizierung wird ein Singleton–Fuzzifizierer verwendet, wodurch sich die Berechnung des Erfüllungsgrades der Prämissen sowie die für das Fine–Tuning nötigen Berechnungen wesentlich vereinfachen. Da in Anwendungen üblicherweise Singleton–Fuzzifizierer verwendet werden, stellt dies keine Ein-

schränkung bezüglich des Einsatzes dar. Insbesondere lassen sich auch bei Verwendung von Singleton–Fuzzifizierern alle stetigen Funktionen beliebig genau approximieren, so daß die Leistungsfähigkeit nicht beeinträchtigt wird (s. Abschnitt 3.13).

Für die Partitionierungen der Ein– und Ausgabe–Dimensionen stehen Dreiecks– und Gauß–Mengen zur Verfügung, da sie in der Praxis häufig verwendet werden. Eine Erweiterung um zusätzliche Mengen–Typen ist auf einfache Weise möglich. Als Fuzzy–Implikation wird entsprechend der Definition des Mamdani–Controllers das Minimum verwendet. Bei der Defuzzifizierung stehen die Schwerpunkt–Methode und die Maximum–Methode zur Wahl. Eine Erweiterung um zusätzliche Methoden ist auf einfache Weise möglich.

6.2 Aufbau und Arbeitsweise des MFOS–M–Netzes

Zur Repräsentation eines Fuzzy–Controllers wurde ein vierschichtiges neuronales Netz entworfen (s. Abb. 6.1), welches eine beliebige, endliche Anzahl von Eingabe– und Ausgabe–Neuronen besitzen kann. Außer den in Abschnitt 6.1 beschriebenen Voraussetzungen gibt es keine Einschränkungen für die verwendeten Regeln und Partitionierungen. Als Gewichte werden Fuzzy–Mengen verwendet.

Definition 6.1 *Abbildung–1: FC $\longrightarrow$ MFOS–M ist die im Folgenden beschriebene Portierung eines Fuzzy–Controllers auf das MFOS–M–System.*

Das Netz ist wie folgt aufgebaut:

In der Eingabeschicht gibt es für jede Eingabe–Dimension ein Neuron, welches nur seine Eingabe weiterleitet. In Schicht 2, der *Prämissenschicht*, gibt es für jede Regel R_k ein Neuron, welches ebenfalls mit R_k bezeichnet wird. Dieses ist mit allen Neuronen aus Schicht 1 verbunden, deren zugehörige Eingaben bei dieser Regel verwendet werden. Somit lassen sich auch linguistische Regeln einsetzen, die nicht alle Eingabe–Werte berücksichtigen. Als Gewicht wird für jede Verbindung die Fuzzy–Menge genommen, die den entsprechenden linguistischen Term aus der Prämisse von Regel R_k für die zugehörige Eingabe–Dimension repräsentiert.

Jedes Neuron R_k in Schicht 2 berechnet den Erfüllungsgrad der Prämisse von Regel R_k. Dazu werden zunächst die Zugehörigkeitsgrade der Eingabewerte zu den jeweiligen Fuzzy–Mengen der Verbindungen mit Schicht 1 berechnet. Anschließend werden diese Werte rekursiv mit einem UND–Operator zum Erfüllungsgrad verknüpft. Dieser ist dann die Ausgabe des Neurons.

In Schicht 3, der *Konklusionsschicht*, gibt es für *jede* verwendete Ausgabe–Partitions–Menge ein Neuron. Dieses ist jeweils mit allen Neuronen aus Schicht 2 verbunden, deren Regel diese Menge als Konklusion hat. Da sich nach Voraussetzung jede Re-

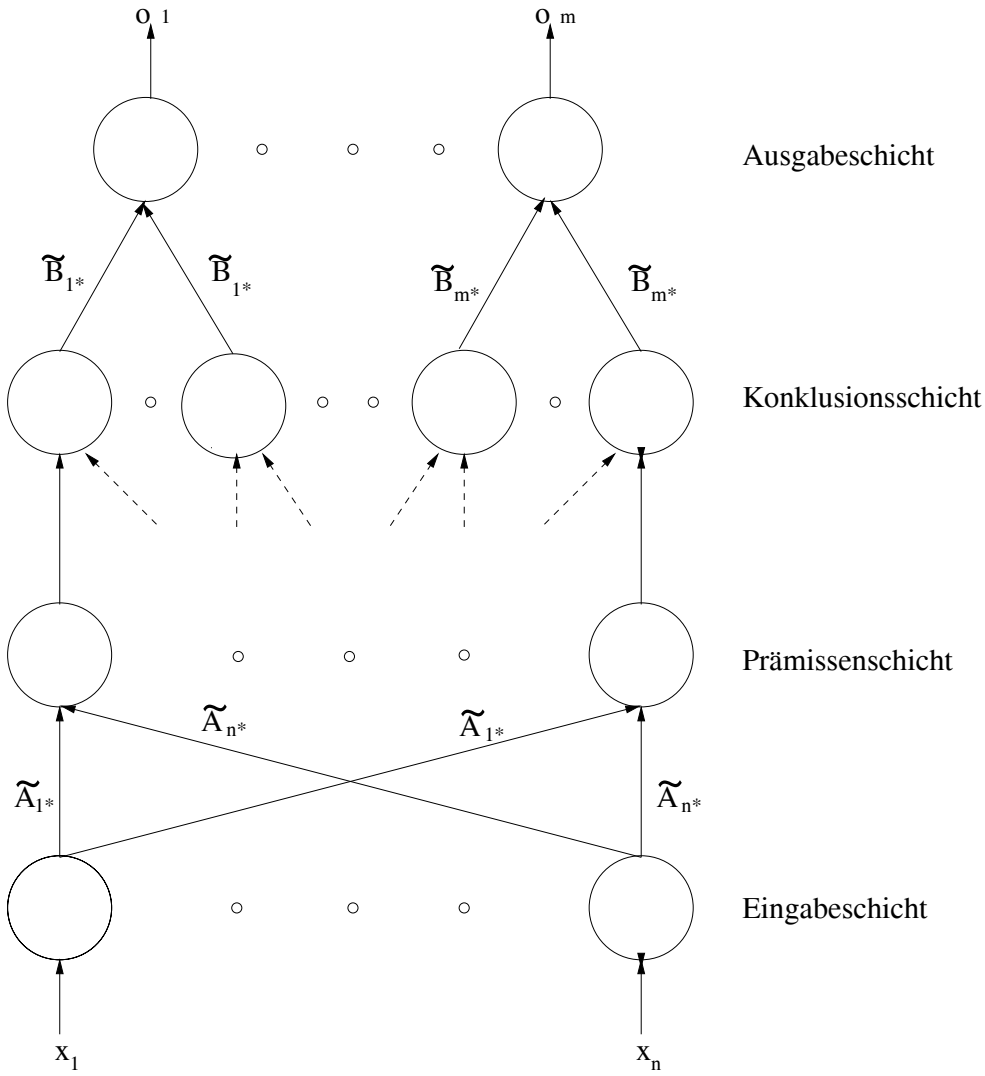

Abbildung 6.1: Aufbau eines MFOS–M–Netzes

gel nur auf eine Ausgabe–Dimension bezieht, läßt sich die Konklusion eindeutig einer Ausgabe–Partitions–Menge zuordnen, woraus sich die Verbindung zu Schicht 3 ergibt. Diese Verbindungen haben keine Gewichte bzw. 1 als unveränderliches Gewicht (die Ausgabe–Partitions–Mengen sind die Gewichte der Verbindungen zwischen Schicht 3 und Schicht 4).

Jedes Neuron in Schicht 3 berechnet das Maximum seiner Eingaben und gibt es aus. Dieser Wert wird als Schnitthöhe für die Ausgabe–Partitions–Menge genommen, die das Neuron repräsentiert. Das Berechnen des Maximums entspricht dem Vereinigen von gleichen Partitions–Mengen, die in verschiedenen Höhen abgeschnitten wurden. Da die Vereinigung assoziativ ist und als Maximum berechnet wird, lassen sich die Schnittmengen der selben Partitions–Menge auf diese Weise vorab vereinigen, ohne das Ergebnis der Vereinigung aller Schnittmengen zu verändern. Dadurch wird zur

Berechnung der Ausgabe–Fuzzy–Mengen nur eine Schnittmenge pro Partitions–Menge erzeugt, wodurch sich die Berechnung der Ausgabe–Fuzzy–Mengen wesentlich vereinfacht.

In der Ausgabeschicht gibt es für jede Dimension des Ausgaberaumes ein Neuron. Dieses ist mit allen Neuronen aus Schicht 3 verbunden, die eine Ausgabe–Partitions–Menge aus dieser Ausgabe–Dimension repräsentieren. Als Gewicht wird jeweils die entsprechende Fuzzy–Menge genommen. Jedes Neuron in der Ausgabeschicht berechnet den Stellwert für seine Ausgabe–Dimension. Dazu werden zunächst alle Ausgabe–Partitions–Mengen dieser Ausgabe–Dimension in Höhe der zugehörigen Eingabe–Werte abgeschnitten, um danach diese Schnittmengen mit einer ODER–Verknüpfung zu vereinigen. Anschließend wird die Vereinigungs–Fuzzy–Menge defuzzifiziert. Das Ergebnis, der Stellwert, wird ausgegeben.

Die Struktur des Netzes repräsentiert somit die Regelbasis des Fuzzy–Controllers vollständig. In den Gewichten werden die Partitionierungen ebenfalls vollständig gespeichert. D.h. bei der Übertragung eines Fuzzy–Controllers auf das MFOS–M–Netz gehen keine Informationen verloren.

Beispiel 6.1 *Hat Regel R_4 die Gestalt*

IF $x_1 = \widetilde{A}_{13}$ *UND* $x_2 = \widetilde{A}_{21}$ *THEN* $y_1 = \widetilde{B}_{12}$

so hat das Neuron R_4 aus Schicht 2 folgende Verbindungen (s. Abb. 6.2):

- *es gibt je eine Verbindung mit Neuron 1 und Neuron 2 aus Schicht 1, mit den Eingabe–Partitions–Mengen $\widetilde{A}_{13}$ bzw. $\widetilde{A}_{21}$ als Gewicht*
- *es gibt eine Verbindung mit dem Neuron aus Schicht 3, das die Ausgabe–Partitions–Menge $\widetilde{B}_{12}$ repräsentiert*
- *(die Fuzzy–Menge $\widetilde{B}_{1*}$ ist das Gewicht der von dem betroffenen Neuron aus Schicht 3 weitergehenden Verbindung zu Ausgabe–Neuron 1)*

Das Netz verhält sich nun so, wie der Fuzzy–Controller, den es repräsentiert. D.h. es werden exakt dieselben Berechnungen durchgeführt, wie im ursprünglichen Fuzzy–Controller. Durch das MFOS–M–Netz steht somit eine Methode zur Verfügung, einen potentiell beliebigen Fuzzy–Controller auf ein funktional äquivalentes neuronales Netz zu portieren.

Die Korrektheit von Abbildung–1 wurde in [Tenhagen, 2000] bewiesen. D.h. zu gleichen Eingaben berechnen der Fuzzy–Controller und das zugehörige MFOS–M–System dieselben Ausgaben. Der Sinn der Übertragung eines Fuzzy–Controllers auf ein neuronales Netz besteht darin, dieses Netz mit bekannten und neu entwickelten Lernverfahren zu

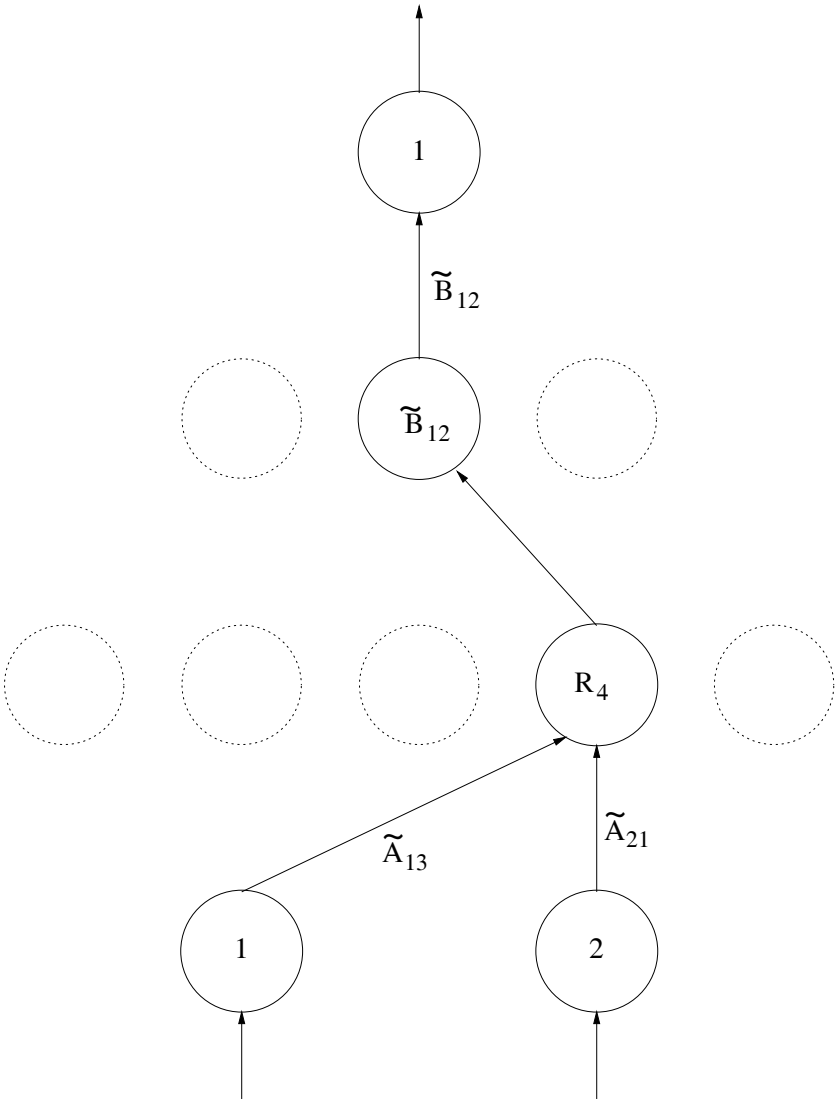

Abbildung 6.2: Regel 4 repräsentierende Verbindungen

trainieren und somit den Fuzzy–Controller zu optimieren. Nach dem Training ist eine Rück–Portierung des MFOS–M–Netzes auf einen gewöhnlichen Fuzzy–Controller möglich. Diese Abbildung wird im Anschluß an die Vorstellung der eingesetzten Lernverfahren in Abschnitt 6.4.5 beschrieben. In den folgenden Abschnitten sollen sämtliche Lernverfahren vorgestellt und untersucht werden.

6.3 Die Pre–Tuning–Verfahren

Vor der Entwicklung von Lernverfahren steht die Frage, welche Möglichkeiten zum Optimieren es prinzipiell gibt. Nach Festlegung der Spezifikationen des Fuzzy–Controllers

(s. Abschnitt 6.1) sind die Regelbasis und die Partitionierungen die entscheidenden Komponenten. Die Regelbasis ist das Herzstück eines Fuzzy–Controllers. In den Regeln ist das Wissen eines Experten gespeichert, wie in welcher Situation reagiert werden soll. Die Partitions–Fuzzy–Mengen repräsentieren die verschiedenen Situationen bzw. die möglichen Folgerungen (s. auch Kapitel 3). Daher ist es wesentlich für die Leistungsfähigkeit eines Fuzzy–Controllers, daß die Regeln und die Partitions–Fuzzy–Mengen "richtig" definiert sind.

In der Erstellung der Regelbasis und der Partitionierungen liegt die Haupt–Schwierigkeit beim Konfigurieren eines Fuzzy–Controllers. Ein Experte mit jahrelanger Erfahrung weiß ganz genau, wie er die Meßwerte seines Systems zu interpretieren hat und welche Steueraktionen er ausführen muß. Dieses Wissen in Worte zu fassen oder gar als Regeln zu formulieren ist dennoch sehr schwierig und scheitert zumeist an mangelndem Verständnis zwischen Experten für das zu steuernde System und z.B. Informatikern. Eine Aussage wie "Man hört doch, in welchem Zustand das System ist" ist leider wenig hilfreich, eine exakte Beschreibung oft nicht möglich.

Aus diesem Grund war es das Ziel bei der Entwicklung der nachfolgend beschriebenen Lernverfahren, die Regeln und die Partitions–Fuzzy–Mengen zu optimieren. Dazu wurde analysiert, welche Art von Fehlern auftreten kann, und wie diese Fehler zu korrigieren sind. Wenn z.B. eine Regel die falsche Konklusion hat, wird in der Situation, die die Prämisse der Regel repräsentiert, die falsche Folgerung gezogen. Somit muß die richtige Konklusion für diese Regel bestimmt werden. Falls in einem Bereich einer Dimension des Eingaberaumes keine Fuzzy–Menge liegt, kann dieser Bereich von den Regeln nicht berücksichtigt werden. Sollen Eingaben aus diesem Bereich verarbeitet werden, muß eine passende Fuzzy–Menge erzeugt werden, etc. Die Bedeutung der Regeln und Partitionierungen sowie die Universalität von Fuzzy–Controllern mit den gewählten Spezifikationen rechtfertigen die Konzentration auf Regeln und Partitions–Mengen bei den Lernverfahren. Folgende Modifikationen sind prinzipiell möglich:

- Modifikation bestehender Regeln
- Erzeugen neuer Regeln
- Löschen vorhandener Regeln
- Modifikation bestehender Fuzzy–Mengen
- Erzeugen neuer Fuzzy–Mengen
- Löschen vorhandener Fuzzy–Mengen

Im Gegensatz zu den bereits bekannten Systemen dieser Art wie dem Verfahren von Lin und Lee (s. Kapitel 4) oder dem NEFCON–System (s. Kapitel 5) bietet das MFOS–M die Möglichkeit, alle diese Modifikationen auf verschiedene Arten durchzuführen. Dabei

bleibt die Entscheidung, welche Modifikationen in welcher Reihenfolge angewendet werden, dem User überlassen. So ist es z.B. möglich, einen bestehenden Fuzzy–Controller zu überprüfen und nur die fehlerhaften Regeln korrigieren zu lassen. Im Extremfall kann auch mit einem vollständig leeren System gestartet werden (kein Fuzzy–Controller zum Verbessern vorhanden). In diesem Fall erzeugt das MFOS–M alle benötigten Regeln und Partitions–Mengen selber. Untersuchungen bezüglich der optimalen Reihenfolge und Anwendung der Lernverfahren werden in Abschnitt 6.6 vorgestellt.

Sämtliche Verfahren basieren auf der Repräsentation von Trainingsbeispielen. Diese sollen möglichst vielseitig sein und alle typischen Situationen abdecken, um eine optimale Anpassung zu ermöglichen. Für fast alle Verfahren werden zusätzlich zu den Eingaben die gewünschten Ausgaben benötigt (überwachtes Lernen), dies wird jeweils erwähnt. Die Beschreibung der Verfahren erfolgt getrennt nach Verfahren zur Anpassung der Regeln und Verfahren zur Anpassung der Partitions–Fuzzy–Mengen. Einfache Beispiele verdeutlichen die Anwendung der Verfahren.

6.3.1 Anpassung der Regeln

Zur Anpassung der Regeln werden alle prinzipiellen Möglichkeiten eingesetzt. Falls eine vorab eingeführte Regel fehlerhaft ist, ergeben Eingaben, die bei dieser Regel einen genügend hohen Erfüllungsgrad bewirken, eine fehlerhafte Ausgabe (ist der Erfüllungsgrad einer Regel null, hat sie keinen Einfluß auf die Ausgabe). Vorhandene Regeln werden überprüft und korrigiert, falls die Konklusion falsch gewählt wurde. Ist für eine Situation noch keine Regel definiert, wird in dieser Situation ebenfalls ein fehlerhafter Ausgabewert berechnet. Regeln, die andere Situationen repräsentieren, bestimmen in diesem Fall trotz ihres geringen Erfüllungsgrades maßgeblich die Ausgabe. Stellt das MFOS–M–System in einer Situation das Fehlen einer geeigneten Regel fest, so erzeugt es automatisch eine für diese Situation optimierte Regel.

Falls vom User versehentlich Regeln eingeführt wurden, die keinen Nutzen für den Fuzzy–Controller haben und deren Korrektur sinnlos wäre, werden diese Regeln entfernt. Da bei der Berechnung der Ausgabe immer *jede* Regel ausgewertet wird (s. Kapitel 3.11), erhöht die Entfernung nutzloser Regeln die Effektivität des Systems auch dann, wenn sie keine Fehler verursachen. Die Anwendung dieser Verfahren sowie deren Reihenfolge sind grundsätzlich frei wählbar. Empfehlungen zum optimalen Einsatz werden in Abschnitt 6.6 gegeben.

Korrigieren von Regeln (überwacht):

Die Prämisse einer Regel repräsentiert eine bestimmte Situation des zu steuernden Systems. Die Konklusion entspricht einer Steueraktion bzw. einem Stellwert für das System. Daher gilt eine Regel als falsch, falls ihre Konklusion einen für die entsprechende Situation falschen Stellwert repräsentiert. In diesem Fall muß die richtige Konklusion bestimmt werden. Die richtige Konklusion entspricht der in der gegebenen Situation

richtigen Ausgabe. Bei der Bewertung der Regeln muß unterschieden werden, ob eine Regel *alle* Eingabe–Variablen verwendet oder nicht: die richtige Ausgabe hängt immer vom Gesamtzustand des zu steuernden Systems ab, und dieser Zustand wird von *allen* Meßwerten bzw. durch *alle* Eingabewerte beschrieben. Daher ist eine direkte Beurteilung der Konklusion einer Regel, die nicht alle Eingabe–Variablen berücksichtigt, nicht möglich. Dennoch sind solche Regeln in bestimmten Fällen sinnvoll, um *Tendenzen* für die Ausgabe vorzugeben.

Beispiel 6.2 *Bei einem Brennofen ist die Gaszufuhr grundsätzlich eher hoch zu wählen, wenn die Temperatur niedrig ist. Daher wird die Regel*

R_1 : *IF* $x_1 =$ *niedrig THEN* $y_1 =$ *hoch*

verwendet, mit x_1 *der Temperatur und* y_1 *der Gaszufuhr. Damit ist die Tendenz gegeben, bei niedriger Temperatur die Gaszufuhr zu erhöhen. Der richtige Wert für die Gaszufuhr hängt noch von weiteren Faktoren wie Kohlendioxidgehalt und Brennzeit ab. Deshalb sind für die optimale Steuerung weitere Regeln mit anderen Eingaben nötig.*

Da der Einfluß einer Regel auf das Ergebnis vom Erfüllungsgrad ihrer Prämisse abhängt, werden zur Beurteilung einer Regel nur Trainingsbeispiele verwendet, die einen hohen Erfüllungsgrad ergeben. Denn für ein Beispiel, das bei einer Regel einen niedrigen Erfüllungsgrad bewirkt, ist die Konklusion dieser Regel irrelevant.

1. Regeln, die alle Eingabe–Variablen verwenden:

Hat die Prämisse einer Regel, die alle Eingabe–Variablen verwendet, einen hohen Erfüllungsgrad, so ist das zu steuernde System zu einem genau so hohen Grad in dem Zustand, den diese Regel repräsentiert. Die Regel mit dem höchsten Erfüllungsgrad der Prämisse beschreibt die entsprechende Situation am besten, andere Regeln sind für andere Situationen vorgesehen. Deshalb werden für jedes Trainingsbeispiel nur die Regeln mit dem höchsten Erfüllungsgrad der Prämisse überprüft. Aufgrund der Form der Regeln (eine Ausgabe–Dimension pro Regel, s. Abschnitt 6.1) können mehrere Regeln die gleiche Prämisse und somit den gleichen Erfüllungsgrad haben. Von den Regeln mit maximalem Erfüllungsgrad der Prämisse wird jede separat getestet.

Falls das einzeln mit einer solchen Regel berechnete Ergebnis einen hohen Fehler aufweist, besitzt die Regel eine falsche Konklusion. Dies wird korrigiert durch Auswahl der optimal geeigneten Ausgabe–Partitions–Menge aus der zugehörigen Ausgabe–Dimension. Dazu werden alle vorhandenen Ausgabe–Partitions–Mengen der zugehörigen Ausgabe–Dimension nacheinander als Konklusion eingesetzt und jeweils nur mit Berücksichtigung der zu korrigierenden Regel die Ausgabe berechnet. Die Partitions–Menge, die dabei den geringsten Fehler verursacht, ist die optimale Konklusion dieser Regel.

Diese Vorgehensweise berücksichtigt die gewählte Defuzzifizierungsmethode. Das ist notwendig, da je nach Methode eventuell unterschiedliche Mengen optimal sind. Zur Durchführung dieses Verfahrens werden für jede Regel, die alle Eingabe–Variablen verwendet, folgende Schritte ausgeführt:

- bestimme das Trainingsbeispiel, welches den maximalen Erfüllungsgrad bewirkt
- falls der maximale Erfüllungsgrad über einer Schranke liegt: berechne nur mit der zu überprüfenden Regel die Ausgabe zu den Eingaben des gewählten Trainingsbeispiels
- bestimme den Fehler der berechneten Ausgabe
- falls der Fehler über einer Schranke liegt: bestimme die neue Konklusion:
  - gehe alle Partitions–Mengen der zugehörigen Ausgabe–Dimension durch
  - berechne jeweils mit der zu überprüfenden Regel die Ausgabe unter Verwendung dieser Partitions–Menge, sowie den Fehler dieser Ausgabe
  - bestimme die Partitions–Menge, die zum minimalen Fehler führt
  - neue Konklusion wird diese Partitions–Menge

Beispiel 6.3 *Die folgenden Fuzzy–Mengen und Regeln ergeben die Partitionierungen und die Regelbasis für einen einfachen Fuzzy–Controller zur Steuerung eines Heizgeräts:*

Eingabe (Temperatur in °C): Dreiecks–Mengen auf [13, 23]:

$$\begin{aligned} \texttt{sehr kalt} &\mathrel{\widehat{=}} \widetilde{A}_1 = (13, 15, 17) \\ \texttt{kalt} &\mathrel{\widehat{=}} \widetilde{A}_2 = (16, 18, 20) \\ \texttt{warm} &\mathrel{\widehat{=}} \widetilde{A}_3 = (19, 21, 23) \end{aligned}$$

Ausgabe (Heizleistung): Dreiecks–Mengen auf [0, 10]:

$$\begin{aligned} \texttt{schwach} &\mathrel{\widehat{=}} \widetilde{B}_1 = (0, 2, 4) \\ \texttt{mittel} &\mathrel{\widehat{=}} \widetilde{B}_2 = (3, 5, 7) \\ \texttt{hoch} &\mathrel{\widehat{=}} \widetilde{B}_3 = (6, 8, 10) \end{aligned}$$

Die linguistischen Variablen sind x für die Temperatur und y für die Heizleistung.

Die richtigen Regeln sind:

R_1 : IF $x =$ sehr kalt THEN $y =$ hoch
R_2 : IF $x =$ kalt THEN $y =$ mittel
R_3 : IF $x =$ warm THEN $y =$ schwach

Durch versehentliches Vertauschen von hoch *und* schwach *bei Erstellung der Regelbasis ergeben sich folgende Regeln:*

R_1 : IF $x =$ sehr kalt THEN $y =$ schwach
R_2 : IF $x =$ kalt THEN $y =$ mittel
R_3 : IF $x =$ warm THEN $y =$ hoch

Das oben beschriebene Verfahren erkennt diese Fehler und korrigiert sie, so daß genau die richtigen Regeln wieder hergestellt werden.

2. Regeln, die nicht alle Eingabe–Variablen verwenden:

Die richtige Ausgabe eines Fuzzy–Controllers hängt vom Gesamtzustand des zu steuernden Systems und damit von *allen* Eingaben ab. Eine Regel, die nicht alle Eingaben verwendet, berücksichtigt nicht den gesamten Zustand des Systems. Eine derartige Regel wird verwendet, um Tendenzen für die Ausgabe vorzugeben (s. Bsp. 6.2). Eine Regel, die nicht alle Eingaben verwendet, hat in verschiedenen Situationen, in denen verschiedene Ausgaben korrekt sind, einen hohen Erfüllungsgrad der Prämisse. Hierfür genügt es, daß die verwendeten Eingaben im von der Regel–Prämisse beschriebenen Bereich liegen. Alle anderen Eingaben haben keinen Einfluß auf den Erfüllungsgrad, wohl aber auf die richtige Ausgabe. Daher muß bei der Korrektur einer solchen Regel überprüft werden, ob die *Tendenz* ihrer Konklusion richtig ist.

Falls das einzeln mit einer Regel, die nicht alle Eingabe–Variablen verwendet, berechnete Ergebnis bei *jedem* Beispiel, das einen hohen Erfüllungsgrad der Prämisse bewirkt, einen hohen Fehler hat, dann liegt die Konklusion dieser Regel in einem falschen Bereich. Dies wird korrigiert durch Auswahl der optimal geeigneten Ausgabe–Partitions–Menge aus der zugehörigen Ausgabe–Dimension als neue Konklusion. Dazu werden alle vorhandenen Ausgabe–Partitions–Mengen der zugehörigen Ausgabe–Dimension nacheinander als Konklusion eingesetzt und jeweils nur mit Berücksichtigung der zu korrigierenden Regel die Ausgabe berechnet. Die Partitions–Menge, die am häufigsten einen Fehler unterhalb einer vorgegebenen Schranke verursacht, ist die optimale Konklusion.

Auch bei diesem Verfahren wird die gewählte Defuzzifizierungsmethode berücksichtigt. Zur Durchführung dieses Verfahrens werden für jede Regel, die nicht alle Eingabe–Variablen verwendet, folgende Schritte ausgeführt:

- gehe alle Trainingsbeispiele durch, die einen Erfüllungsgrad über einer Schranke bewirken
- berechne jeweils mit der zu überprüfenden Regel die Ausgabe zu den Eingaben dieses Beispiels und den Fehler
- falls der Fehler jedesmal über einer Schranke liegt: bestimme die neue Konklusion:
  - bestimme für jedes verwendete Beispiel die Ausgabe–Partitions–Menge, die bei Berechnung der Ausgabe nur mit der zu korrigierenden Regel den minimalen Fehler ergibt
  - bestimme die Ausgabe–Partitions–Menge, die bei den verwendeten Beispielen am häufigsten einen Fehler unter einer Schranke ergibt
  - neue Konklusion wird diese Partitions–Menge

Beispiel 6.4 *Die folgenden Fuzzy–Mengen und Regeln ergeben die Partitionierungen und die Regelbasis für einen einfachen Fuzzy–Controller zur Steuerung eines Heizgeräts. Im Gegensatz zu Beispiel 6.3 wird berücksichtigt, ob es Nacht ist oder nicht, wobei gilt, daß nachts grundsätzlich nur* schwach *geheizt werden soll.*

Für Eingabe 1 (Temperatur in °C) und die Ausgabe werden die gleichen Dreiecks–Mengen wie für Beispiel 6.3 verwendet. Eingabe 2 gibt an, ob es Nacht ist (kodiert durch "3") oder Tag (kodiert durch "1"), dies wird repräsentiert durch Dreiecks–Mengen auf [0, 4]:

$$\texttt{Tag} \quad \widehat{=}\ \widetilde{A}_{21} = (0,1,2)$$
$$\texttt{Nacht} \ \widehat{=}\ \widetilde{A}_{22} = (2,3,4)$$

Die linguistischen Variablen sind x_1 für die Temperatur, x_2 für `Tag/Nacht` *und y für die Heizleistung.*

Die richtigen Regeln sind:

R_1 : *IF* $x_1 =$ `sehr kalt` *UND* $x_2 =$ `Tag` *THEN* $y =$ `hoch`
R_2 : *IF* $x_1 =$ `kalt` *UND* $x_2 =$ `Tag` *THEN* $y =$ `mittel`
R_3 : *IF* $x_1 =$ `warm` *UND* x_2 `=` `Tag` *THEN* $y =$ `schwach`
R_4 : *IF* $x_2 =$ `Nacht` *THEN* $y =$ `schwach`

Ist Regel 4 versehentlich als

IF $x_2 =$ `Nacht` *THEN* $y =$ `hoch`

erzeugt worden, so erkennt das oben beschriebene Verfahren diesen Fehler und stellt die korrekte Regel wieder her.

Erzeugen von Regeln (überwacht):

Es ist notwendig, für *jede* Situation des Systems, die von einem Fuzzy–Controller geregelt werden soll, geeignete Regeln zu erzeugen. Falls für eine konkrete Situation gar keine Regel definiert wurde, übernehmen andere Regeln, die für andere Situationen vorgesehen sind, maßgeblich die Bestimmung der Ausgabe. Dabei sind Fehler zu erwarten, da üblicherweise in verschiedenen Situationen verschiedene Ausgaben korrekt sind. Generell gilt, je größer der Erfüllungsgrad der Prämisse einer Regel ist, desto höher ist der Einfluß dieser Regel auf das Ergebnis. Der Erfüllungsgrad ist ein Maßstab für die Übereinstimmung der Eingabe–Werte mit der Situation, die die Prämisse einer Regel repräsentiert. Daher ist der Erfüllungsgrad der Prämisse das geeignete Kriterium, um zu überprüfen, ob in einer gegebenen Situation eine passende Regel vorhanden ist.

Das Verfahren zur Erzeugung neuer Regeln basiert auf folgender Heuristik: hat bei einem Trainingsbeispiel die Prämisse von *jeder* Regel einen geringen Erfüllungsgrad, dann gibt es keine passende Regel für diese Situation. Also muß eine neue Regel erzeugt werden. Für die Prämisse und für die Konklusion werden die Partitions–Mengen bestimmt, die am besten die Eingabe– bzw. die korrekten Ausgabe–Werte repräsentieren. Zur Durchführung dieses Verfahrens werden für jedes Trainingsbeispiel die folgenden Schritte ausgeführt:

- berechne für jede Regel den Erfüllungsgrad der Prämisse
- ist der Erfüllungsgrad bei jeder Regel unterhalb einer Schranke, erzeuge eine neue Regel
  - bestimme für jeden Eingabe–Wert die Partitions–Menge aus der zugehörigen Eingabe–Dimension, die den höchsten Zugehörigkeitsgrad ergibt
  - die so bestimmten Eingabe–Partitions–Mengen ergeben die Prämisse der zu erzeugenden Regel(n)
  - bestimme für jeden Ausgabe–Wert die Partitions–Menge aus der zugehörigen Ausgabe–Dimension, die den höchsten Zugehörigkeitsgrad ergibt
  - die so bestimmten Ausgabe–Partitions–Mengen ergeben die Konklusion der zu erzeugenden Regel(n) (pro Ausgabe–Dimension wird eine Regel erzeugt)

Beispiel 6.5 *Seien die richtigen Partitions–Mengen und Regeln die in Beispiel 6.3 verwendeten. Wird nun bei der Erzeugung der Regelbasis Regel 1:*

IF $x_1 =$ `sehr kalt` *THEN* $y =$ `hoch`

vergessen, so erzeugt das oben beschriebene Verfahren genau diese Regel, und die Regelbasis ist vervollständigt.

Löschen von Regeln:

Bei der Berechnung der Ausgabe eines Fuzzy–Controllers wird jedesmal *jede* Regel ausgewertet, d.h. es wird ihr Erfüllungsgrad berechnet, die zugehörige Schnitt–Menge gebildet und mit den anderen Schnitt–Mengen der selben Ausgabe–Dimension vereinigt (s. Kapitel 3.11). Diese Berechnungen werden auch dann durchgeführt, wenn die Regel keinen Einfluß auf das Ergebnis hat. Die Entfernung nutzloser Regeln bewirkt keine Verringerung des Ausgabe–Fehlers (hat eine Regel keinen Einfluß auf das Ergebnis, verursacht sie auch keinen Fehler), erhöht aber die Effektivität des Fuzzy–Controllers und ist somit sinnvoll.

Die Identifizierung nutzloser Regeln basiert auf folgender Heuristik: von versehentlich mehrfach definierten Regeln wird sofort jedes überzählige Exemplar gelöscht. Aufgrund der Arbeistweis des MFOS–M–Systems (s. Abschnitt 6.2) gibt es zwei weitere Möglichkeiten, um überflüssige Regeln zu finden: der Erfüllungsgrad der Prämisse einer Regel ergibt die Schnitthöhe für die Ausgabe–Partitions–Menge der Regel–Konklusion. Falls in verschiedenen Situationen die gleiche Folgerung korrekt ist, gibt es verschiedene Regeln mit der selben Konklusion. In diesem Fall erhält das zu der Konklusion gehörige Neuron in Schicht 3 die Erfüllungsgrade der Prämissen sämtlicher dieser Regeln als Eingaben und berechnet davon das Maximum. Das heißt, es bestimmt bei jedem Beispiel von allen Regeln mit der selben Konklusion diejenige die verwendete Schnitthöhe, deren Prämisse den höchsten Erfüllungsgrad hat.

Beispiel 6.6 *Die Ausgabe–Partitions–Menge $\widetilde{B}$ wird in Höhe E_1, E_2 und E_3 abgeschnitten. Entscheidend ist nur die Schnitthöhe E_3 (s. Abb. 6.3).*

Daraus ergeben sich zwei Folgerungen:

a) Wenn bei jedem Trainingsbeispiel von allen Regeln mit der gleichen Konklusion immer dieselbe Regel den maximalen Erfüllungsgrad der Prämissen hat, bestimmt diese Regel alleine die Schnitthöhe der zugehörigen Ausgabe–Partitions–Menge. Alle anderen Regeln mit dieser Konklusion haben niemals Einfluß auf das Ergebnis. Also können sie gelöscht werden. Zur Durchführung dieses Verfahrens werden jeweils alle Regeln mit der gleichen Konklusion gemeinsam betrachtet und die folgenden Schritte ausgeführt:

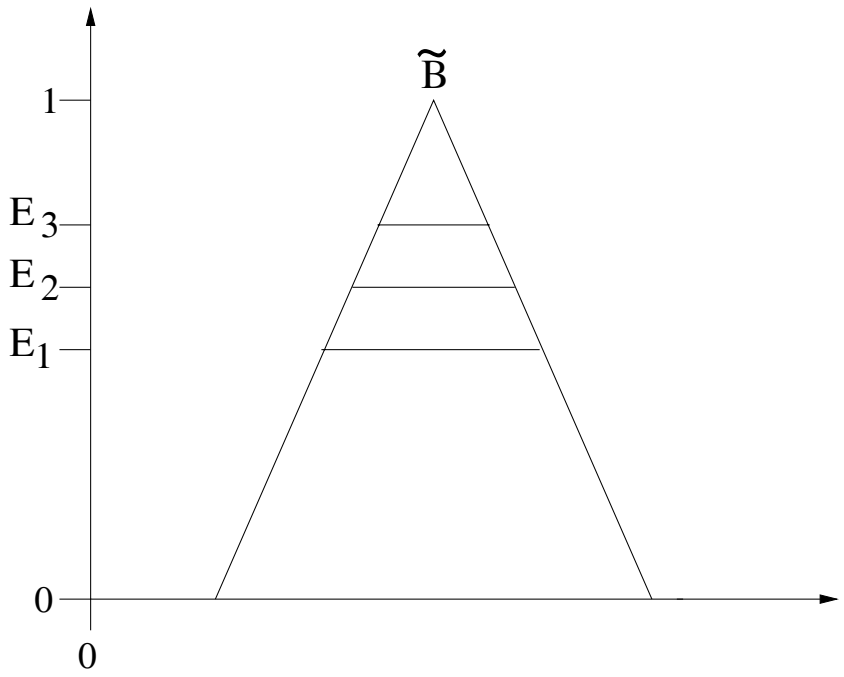

Abbildung 6.3: Verschiedene Schnitthöhen einer Partitions–Menge

- gehe alle Trainingsbeispiele durch
- bestimme die Regel, deren Prämisse den maximalen Erfüllungsgrad hat
- falls jedesmal die Prämisse derselben Regel den maximalen Erfüllungsgrad hat, werden die anderen betrachteten Regeln gelöscht

b) ist ein Umkehrschluß zu a), falls a) nicht erfolgreich war: wenn bei jedem Beispiel von allen Regeln mit der gleichen Konklusion immer dieselbe Regel den minimalen Erfüllungsgrad der Prämissen hat, bestimmt diese Regel niemals die Schnitthöhe und hat niemals Einfluß auf das Ergebnis. Also kann sie gelöscht werden. Zur Durchführung dieses Verfahrens werden jeweils alle Regeln mit der gleichen Konklusion gemeinsam betrachtet und die folgenden Schritte ausgeführt:

- wiederhole, bis nur eine Regel übrig bleibt oder bis sich nichts mehr ändert
- gehe alle Trainingsbeispiele durch
- bestimme die Regel, deren Prämisse den minimalen Erfüllungsgrad hat
- falls jedesmal die Prämisse derselben Regel den minimalen Erfüllungsgrad hat, wird diese Regel gelöscht

Beispiel 6.7 *Seien die folgenden Fuzzy–Mengen und Regeln definiert:*

Eingabe 1: Dreiecks–Mengen auf [1, 11]:

$\texttt{klein} \;\widehat{=}\; \widetilde{A}_{11} = (1, 3, 5)$
$\texttt{mittel} \;\widehat{=}\; \widetilde{A}_{12} = (4, 6, 8)$
$\texttt{groß} \;\widehat{=}\; \widetilde{A}_{13} = (7, 9, 11)$

Eingabe 2: Dreiecks–Mengen auf [2, 9]:

$\texttt{niedrig} \;\widehat{=}\; \widetilde{A}_{21} = (2, 4, 6)$
$\texttt{hoch} \;\widehat{=}\; \widetilde{A}_{22} = (5, 7, 9)$

Ausgabe: Dreiecks–Mengen auf [5, 31]:

$\texttt{klein} \;\widehat{=}\; \widetilde{B}_1 = (5, 10, 15)$
$\texttt{mittel} \;\widehat{=}\; \widetilde{B}_2 = (13, 18, 23)$
$\texttt{groß} \;\widehat{=}\; \widetilde{B}_3 = (21, 26, 31)$

Regeln:

R_1 : IF $x_1 =$ klein UND $x_2 =$ niedrig THEN $y =$ klein
R_2 : IF $x_1 =$ mittel UND $x_2 =$ niedrig THEN $y =$ klein
R_3 : IF $x_1 =$ groß UND $x_2 =$ niedrig THEN $y =$ klein
R_4 : IF $x_1 =$ klein UND $x_2 =$ hoch THEN $y =$ mittel
R_5 : IF $x_1 =$ mittel UND $x_2 =$ hoch THEN $y =$ mittel
R_6 : IF $x_1 =$ groß UND $x_2 =$ hoch THEN $y =$ groß
R_7 : IF $x_2 =$ niedrig THEN $y =$ klein

Durch das Verfahren zum Löschen von Regeln werden die ersten drei Regeln entfernt. Die übrigen Regeln

R_4 :IF $x_1 =$ klein UND $x_2 =$ hoch THEN $y =$ mittel
R_5 :IF $x_1 =$ mittel UND $x_2 =$ hoch THEN $y =$ mittel
R_6 :IF $x_1 =$ groß UND $x_2 =$ hoch THEN $y =$ groß
R_7 :IF $x_2 =$ niedrig THEN $y =$ klein

ergeben genau dieselben Ausgaben wie alle Regeln zusammen.

6.3.2 Anpassung der Partitions–Fuzzy–Mengen

Die Verfahren zum Korrigieren und Erzeugen von Regeln (Abschnitt 6.3.1) berücksichtigen jeweils bereits vorhandene Partitions–Fuzzy–Mengen. Genau wie bei den Regeln besteht auch bei den Fuzzy–Mengen die Möglichkeit, daß bei der Erstellung der Partitionierungen notwendige Partitions–Mengen vergessen wurden oder unpassend definiert wurden. Falls in einem Bereich einer Eingabe–Dimension gar keine Fuzzy–Menge liegt, können Eingaben aus diesem Bereich nicht sinnvoll verarbeitet werden. Der Zugehörigkeitsgrad von Werten außerhalb des Trägers einer Fuzzy–Menge ist immer null (s. Kapitel 3), dementsprechend wäre der Erfüllungsgrad der Regel–Prämissen bei diesen Eingaben ebenfalls null.

Liegt eine bestimmte Fuzzy–Menge an der falschen Position (Modalwert ungeeignet), so führt dies zu falschen Ergebnissen. Es ist auch möglich, zu wenig Fuzzy–Mengen vorzugeben. Wenn z.B. im Bereich zu Eingabe–Werten 1 und 2 verschiedene Ausgaben korrekt sind, jedoch der ganze Bereich von 1 bis 2 nur von einer Fuzzy–Menge abgedeckt ist, ist hier keine Unterscheidung mit Hilfe der Regeln möglich. Daher lassen sich ohne zusätzliche Eingabe–Partitions–Mengen keine verschiedenen Ausgaben zu den Eingaben 1 und 2 erreichen. Umgekehrt gibt es auch zu feine Partitionierungen mit unnötigen Unterteilungen von Bereichen, in denen dieselbe Ausgabe korrekt ist.

In diesen Fällen ist eine Anpassung der Partitions–Fuzzy–Mengen erforderlich. Bei Bedarf werden vom MFOS–M–System neue Fuzzy–Mengen erzeugt und überflüssige Fuzzy–Mengen gelöscht. Die Modifikation bestehender Partitions–Mengen, d.h. die Änderung des Modalwertes (verschieben) und der Weite (verbreitern, verschmälern) wird als *Fine–Tuning* durchgeführt und in Abschnitt 6.4 beschrieben.

Erzeugen von Fuzzy–Mengen (überwacht):

Die Regeln des Fuzzy–Controllers verwenden in der Prämisse und der Konklusion die Partitions–Mengen. Daher ist es notwendig, das Erzeugen von Fuzzy–Mengen wie auch das Löschen von Fuzzy–Mengen unter Berücksichtigung der Regeln durchzuführen. Eine neue Fuzzy–Menge hat nur einen Sinn, wenn sie auch von mindestens einer Regel berücksichtigt wird. Aus diesem Grund sind die Verfahren zur Erzeugung neuer Fuzzy–Mengen in die Verfahren zur Korrektur bzw. Erzeugung von Regeln (s. Abschnitt 6.3.1) integriert.

Grundsätzlich ist es auch möglich zu verhindern, daß das MFOS–M–System neue Fuzzy–Mengen erzeugt. Dies ist erforderlich, falls der Fuzzy–Controller für eine spezielle Hardware vorgesehen ist, die nur eine vorgegebene Anzahl von Fuzzy–Mengen speichern kann. In diesem Fall muß der Fuzzy–Controller soweit wie möglich mit den vorhandenen Mengen optimiert werden.

Das MFOS–M–System erzeugt bei Bedarf Eingabe– und Ausgabe–Partitions–Mengen. Der Modalwert wird jeweils auf geeignete Weise aus den Eingabe– bzw. Ausgabe–

Werten der Trainingsbeispiele bestimmt. Als Weite wird das 1.2–fache des Abstandes zum Modalwert der nächsten Nachbar–Menge festgelegt. Auf diese Weise wird eine gute Überlappung der Fuzzy–Mengen erreicht, die in den meisten Fällen von Vorteil ist. Sollten die erzeugten Fuzzy–Mengen dennoch nicht zu hinreichend guten Ergebnissen führen, ist eine anschließende Optimierung im Rahmen des Fine–Tuning (s. Abschnitt 6.4) möglich. Da die Verfahren zur Erzeugung von Fuzzy–Mengen in die Verfahren zur Korrektur bzw. Erzeugung von Regeln integriert sind, folgt eine Beschreibung getrennt nach diesen Methoden:

1. Beim Korrigieren von Regeln mit allen Eingabe–Variablen:

Bei der Korrektur einer Regel, die alle Eingabe–Variablen verwendet, wird die Partitions–Menge aus der zugehörigen Ausgabe–Dimension bestimmt, mit der das einzeln mit dieser Regel berechnete Ergebnis den minimalen Fehler aufweist. Falls dieser minimale Fehler über einer Schranke liegt, ist keine passende Ausgabe–Partitions–Menge für diese Regel vorhanden. Daher wird während der Anwendung des Korrektur–Verfahrens eine geeignete Fuzzy–Menge erzeugt und als Konklusion der Regel festgelegt:

Neue Partitions–Menge:

- Modalwert: gewünschte Ausgabe
- Breite: 1.2 mal Abstand zum Modalwert der nächsten Nachbar–Menge

Beispiel 6.8 *Seien die richtigen Partitions–Mengen und Regeln die in Beispiel 6.3 verwendeten. Wurde die Ausgabe–Partitions–Menge für* `mittel` *nicht erzeugt und statt der Regel*

IF $x =$ `kalt` *THEN* $y =$ `mittel`

die Regel

IF $x =$ `kalt` *THEN* $y =$ `hoch`

verwendet, so wird durch das oben beschriebene Verfahren bei der Korrektur dieser Regel die Partitions–Menge $(3.2, 5, 6.8)$ *erzeugt, die nun für* `mittel` *steht. Gleichzeitig wird die Regel*

IF $x =$ `kalt` *THEN* $y =$ `hoch`

korrigiert zu

IF $x =$ `kalt` *THEN* $y =$ `mittel`

Mit der erzeugten Fuzzy–Menge berechnet der Fuzzy–Controller genauso gute Ergebnisse wie mit der ursprünglichen Fuzzy–Menge $(3, 5, 7)$.

2. Beim Korrigieren von Regeln ohne alle Eingabe–Variablen:

Bei der Korrektur einer Regel, die nicht alle Eingabe–Variablen verwendet, wird die Partitions–Menge aus der zugehörigen Ausgabe–Dimension bestimmt, bei der das einzeln mit dieser Regel berechnete Ergebnis am häufigsten einen Fehler unter einer Schranke ergibt. Wenn der Fehler keinmal unter dieser Schranke liegt, ist keine passende Ausgabe–Partitions–Menge für diese Regel vorhanden. Daher wird während der Anwendung des Korrektur–Verfahrens eine geeignete Fuzzy–Menge erzeugt und als Konklusion festgelegt:

Neue Partitions–Menge:

- Modalwert: Durchschnitt der gewünschten Ausgaben von Trainingsbeispielen, die einen Erfüllungsgrad der Prämisse über einer Schranke bewirken
- Breite: 1.2 mal Abstand zum Modalwert der nächsten Nachbar–Menge

Beispiel 6.9 *In Anlehnung an Beispiel 6.4 ergeben die folgenden Fuzzy–Mengen und Regeln einen einfachen Fuzzy–Controller zur Steuerung eines Heizgeräts. Dabei wird auch in diesem Fall berücksichtigt, ob es Nacht ist oder nicht. Für Eingabe 1 (Temperatur in °C), Eingabe 2 (Tag/Nacht) und die Ausgabe (Heizleistung) werden dieselben Partitions–Mengen wie in Beispiel 6.4 verwendet. Im Gegensatz zu Beispiel 6.4 soll die Heizleistung tagsüber entweder* `mittel` *oder* `hoch` *sein, nachts grundsätzlich nur* `schwach`.

Die richtigen Regeln sind daher:

R_1 : *IF* $x_1 =$ `sehr kalt` *UND* $x_2 =$ `Tag` *THEN* $y =$ `hoch`
R_2 : *IF* $x_1 =$ `kalt` *UND* $x_2 =$ `Tag` *THEN* $y =$ `hoch`
R_3 : *IF* x_1 `= warm` *UND* x_2 `= Tag` *THEN* $y =$ `mittel`
R_4 : *IF* $x_2 =$ `Nacht` *THEN* $y =$ `schwach`

Wurde die Ausgabe–Partitions–Menge für `schwach` *nicht definiert und statt der Regel*

IF $x_2 =$ `Nacht` *THEN* $y =$ `schwach`

die Regel

IF $x_2 =$ *Nacht THEN* $y =$ *mittel*

verwendet, so wird durch das oben beschriebene Verfahren bei der Korrektur dieser Regel die Partitions–Menge $(0.2, 2, 3.8)$ *erzeugt, die nun für* `schwach` *steht. Gleichzeitig wird die Regel*

IF $x_2 =$ *Nacht THEN* $y =$ *mittel*

korrigiert zu

IF $x_2 =$ *Nacht THEN* $y =$ `schwach`

Mit der erzeugten Fuzzy–Menge liefert der Fuzzy–Controller genauso gute Ergebnisse wie mit der ursprünglichen Fuzzy–Menge $(0, 2, 4)$.

3. Beim Erzeugen von Regeln:

Bei der Erzeugung einer Regel werden für die Prämisse und die Konklusion die Partitions–Mengen bestimmt, bei denen die Eingabe–Werte bzw. Ausgabe–Werte des betrachteten Trainingsbeispiels den höchsten Zugehörigkeitsgrad ergeben. Falls für einen Wert der Zugehörigkeitsgrad bei keiner Partitions–Menge aus der zugehörigen Eingabe– bzw. Ausgabe–Dimension über einer Schranke liegt, ist keine passende Partitions–Menge für diesen Wert vorhanden. Daher wird während der Anwendung des Verfahrens zur Erzeugung neuer Regeln eine geeignete Fuzzy–Menge erzeugt und als Menge aus der Prämisse bzw. als Konklusion der Regel festgelegt:

Neue Partitions-Menge:

- Modalwert: betroffener Ein– bzw. Ausgabe–Wert
- Breite: 1.2 mal Abstand zum Modalwert der nächsten Nachbar–Menge

Beispiel 6.10 *Werden in der Situation von Beispiel 6.3 die Eingabe–Partitions–Menge für* `kalt` $\widehat{=}\widetilde{A}_2 = (16, 18, 20)$ *und die Regel*

IF $x =$ `kalt` *THEN* $y =$ *mittel*

vergessen, so wird durch das oben beschriebene Verfahren bei der Erzeugung der vergessenen Regel die Partitions–Menge $(16.2, 18, 19.8)$ *generiert, die nun für* `kalt` *steht. Gleichzeitig wird die vergessene Regel erstellt. Mit der erzeugten Fuzzy–Menge liefert der Fuzzy–Controller genauso gute Ergebnisse wie mit der ursprünglichen Fuzzy–Menge* $(16, 18, 20)$.

Bemerkung 6.2 *Beispiel 6.10 konnte weder von dem Verfahren von Lin und Lee (s. Kapitel 4) noch vom NEFCON–Modell (s. Kapitel 5) erfolgreich behandelt werden, da beide Verfahren keine neuen Eingabe–Partitions–Mengen erzeugen können.*

Löschen von Fuzzy–Mengen (überwacht):

Ebenso wie unnötige Regeln verringern auch unnötige Fuzzy–Mengen die Effektivität eines Fuzzy–Controllers. Die Eingabe–Partitions–Mengen unterteilen die Eingabe–Dimensionen in verschiedene zu unterscheidende Bereiche, in denen verschiedene Ausgaben korrekt sind. Daher ist die Eingabe–Partitionierung zu fein gewählt, falls es verschiedene Eingabe–Partitions–Mengen in einem Bereich gibt, in denen die gleiche Ausgabe richtig ist. Solche Fuzzy–Mengen lassen sich zu einer einzigen Fuzzy–Menge zusammenfassen. Das Löschen von Fuzzy–Mengen kann nicht ohne Einbeziehung der Regeln durchgeführt werden, da die Regeln sich nur auf vorhandene Fuzzy–Mengen beziehen dürfen. Nachdem das MFOS–M zwei gleichwertige Fuzzy–Mengen zusammengefaßt hat, werden daher alle Regeln, die sich nur in der Verwendung dieser Mengen unterscheiden, ebenfalls zusammengefaßt. Somit werden nicht nur unnötige Fuzzy–Mengen entfernt, sondern auch unnötig gewordene Regeln, die mit dem Verfahren zum Löschen von Regeln (s. Abschnitt 6.3.1) nicht identifiziert werden können.

Ob es möglich ist, zwei benachbarte Fuzzy–Mengen zusammen zu fassen, wird gemäß folgender Heuristik erkannt: falls im gesamten Bereich von zwei benachbarten Mengen bei jedem Trainingsbeispiel dieselbe Ausgabe richtig ist, können diese beiden Mengen zu einer größeren Menge zusammengefaßt werden. Da die richtige Ausgabe jedoch von *allen* Eingabe–Werten abhängt, müssen bei der Überprüfung von zwei Nachbar–Mengen *alle* Eingaben berücksichtigt werden. Der Wechsel eines Eingabe–Wertes von einer Partitions–Menge zur Nachbar–Menge könnte durch Änderungen der anderen Eingabe–Werte ausgeglichen werden, so daß die richtige Ausgabe gleich bleibt. Die separate Überprüfung von zwei Nachbar–Mengen ergibt kein Kriterium, um zu entscheiden, ob diese Mengen gleichwertig sind. Daher werden zwei Nachbar–Mengen wie folgt verglichen:

Es werden jeweils alle Trainingsbeispiele gemeinsam betrachtet, bei denen ein Eingabe–Wert in einer von zwei Nachbar–Mengen liegt, und alle anderen Eingabe–Werte jeweils in derselben Menge. Dabei wird die Partitions–Menge, in der ein Wert liegt, als diejenige bestimmt, bei der der Zugehörigkeitsgrad am höchsten ist. Falls bei allen gemeinsam betrachteten Beispielen jedesmal die gewünschte Ausgabe im selben Bereich liegt (bei verschiedenen Beispielgruppen dürfen es verschieden Bereiche sein), ist es gleichwertig, in welcher der beiden überprüften Nachbar–Mengen sich der Eingabe–Wert befindet. Somit können diese Mengen zusammengefaßt und die dann überflüssige Menge gelöscht werden. Als Modalwert der neuen Partitions–Menge wird das arithmetische Mittel der Modalwerte von den beiden Nachbar–Mengen festgelegt. Die Breite wird so gewählt, daß der gesamte Bereich beider Mengen abgedeckt wird. Anschließend werden noch alle Regeln zusammengefaßt, die sich nur bei den betrachteten Nachbar–Mengen unterscheiden. Dazu werden die folgenden Schritte durchgeführt:

- betrachte jeweils alle Trainingsbeispiele gemeinsam, bei denen ein Eingabe–Wert in einer von zwei Nachbar–Mengen liegt, und alle anderen Eingabe–Werte in derselben Menge
- bestimme für alle betrachteten Beispiele die Ausgabe–Partitions–Mengen, in denen die gewünschten Ausgaben liegen
- wenn die gewünschten Ausgaben jeweils bei allen gemeinsam betrachteten Beispielen in den gleichen Partitions–Mengen liegen, fasse die beiden Nachbar–Mengen zusammen:
  - Modalwert: arithmetisches Mittel der Modalwerte der Nachbar–Mengen
  - Breite: Breite des gemeinsam überdeckten Bereiches
- fasse danach alle Regeln zusammen, die sich nur in den beiden Nachbar–Mengen unterscheiden
  - Eingaben, die einer der Nachbar–Mengen zugeordnet sind, werden der zusammengefaßten Menge zugeordnet
  - die Konklusion bleibt unverändert

Beispiel 6.11 *Sei die gleiche Situation wie in Beispiel 6.4 gegeben. Ersetzt man die Partitions–Menge für* `sehr kalt` $\widehat{=}\widetilde{A}_{11} = (13, 15, 17)$ *durch zwei Mengen, ergeben sich z.B. folgende Partitionierungen:*

Eingabe 1 (Temperatur in °C): Dreiecks–Mengen auf [13, 23]:

$$\begin{aligned}
&\texttt{besonders kalt} \mathrel{\widehat{=}} \widetilde{A}_{11} = (13,\ 14.25, 15.5)\\
&\texttt{sehr kalt} \mathrel{\widehat{=}} \widetilde{A}_{12} = (14.5, 15.75, 17)\\
&\texttt{kalt} \mathrel{\widehat{=}} \widetilde{A}_{13} = (16, 18, 20)\\
&\texttt{warm} \mathrel{\widehat{=}} \widetilde{A}_{14} = (19, 21, 23)
\end{aligned}$$

Für Eingabe 2 und die Ausgabe werden die gleichen Partitions–Mengen wie in Beispiel 6.4 verwendet.

Die richtigen Regeln sind nun:

$$\begin{aligned}
R_1 &: \texttt{IF } x_1 = \texttt{besonders kalt } \texttt{UND } x_2 = \texttt{Tag } \texttt{THEN } y = \texttt{hoch}\\
R_2 &: \texttt{IF } x_1 = \texttt{sehr kalt } \texttt{UND } x_2 = \texttt{Tag } \texttt{THEN } y = \texttt{hoch}\\
R_3 &: \texttt{IF } x_1 = \texttt{kalt } \texttt{UND } x_2 = \texttt{Tag } \texttt{THEN } y = \texttt{mittel}\\
R_4 &: \texttt{IF } x_1 = \texttt{warm } \texttt{UND } x_2 = \texttt{Tag } \texttt{THEN } y = \texttt{schwach}\\
R_5 &: \texttt{IF } x_2 = \texttt{Nacht } \texttt{THEN } y = \texttt{schwach}
\end{aligned}$$

Das oben beschriebene Verfahren erkennt, daß die Aufteilung des unteren Temperaturbereiches in `besonders kalt` *und* `sehr kalt` *nicht nötig ist und faßt beide Partitions–Mengen zur Partitions–Menge* $(13, 15, 17)$ *zusammen, die nun für* `sehr kalt` *steht. Gleichzeitig werden die Regeln*

IF $x_1 =$ `besonders kalt` *UND* $x_2 =$ `Tag` *THEN* $y =$ `hoch`

und

IF $x_1 =$ `sehr kalt` *UND* $x_2 =$ `Tag` *THEN* $y =$ `hoch`

zusammengefaßt zu

IF $x_1 =$ `sehr kalt` *UND* $x_2 =$ `Tag` *THEN* $y =$ `hoch`

Die nun überflüssige Partitions–Menge für `besonders kalt` *wird gelöscht. Damit entsprechen die Partitions–Mengen und Regeln den ursprünglich in Beispiel 6.4 verwendeten.*

6.4 Die Fine–Tuning–Verfahren

Zur Feinabstimmung der Partitions–Fuzzy–Mengen wird ein Verschieben der Mengen und eine Änderung der Breite durchgeführt. Die Partitions–Fuzzy–Mengen sind die Gewichte der Verbindungen des verwendeten neuronalen Netzes. Das Ziel ist es, die Gewichte so zu verändern, daß der Fehler der einzelnen Neuronen und damit der Fehler der Netzausgabe minimiert wird. Um mit diesem Verfahren zur endgültigen Optimierung des Fuzzy–Controllers die Feinabstimmung durchzuführen, sollten vor seiner Anwendung die Regeln korrekt definiert sein, ebenso sollten die Partitions–Mengen "ungefähr" stimmen. Insbesondere muß die Anzahl der Partitions–Mengen passend sein. Diese Bedingungen lassen sich durch Anwendung der Pre–Tuning–Verfahren (s. Abschnitt 6.3) garantieren.

In manchen Fällen ist es von Vorteil, die Fine–Tuning–Verfahren vor den Pre–Tuning–Verfahren anzuwenden. Falls in einem Bereich der Eingabe– oder Ausgabe–Dimensionen keine geeignete Fuzzy–Menge vorhanden ist, wird mit den Pre–Tuning–Verfahren eine erzeugt. Es besteht jedoch die Möglichkeit, daß durch Anwendung der Fine–Tuning–Verfahren eine bereits vorhandene Fuzzy–Menge genau in diesen Bereich verschoben wird. Damit wäre das Erzeugen einer neuen Fuzzy–Menge unnötig. Dies ist insbesondere möglich, falls die Anzahl der erzeugten Partitions–Fuzzy–Mengen korrekt ist, und nur die Positionen nicht optimal sind. D.h. durch Anwendung der Fine–Tuning–Verfahren vor den Pre–Tuning–Verfahren läßt sich eventuell das unnötige Erzeugen zusätzlicher Partitions–Fuzzy–Mengen vermeiden. Eine Analyse der Wechselwirkungen zwischen den einzelnen Verfahren und Empfehlungen für die optimale Reihenfolge werden in Abschnitt 6.6 vorgestellt.

Beispiel 6.12 *Die Eingabe x wird von der gegebenen Fuzzy–Menge $\widetilde{A}_1$ nicht erfaßt. Entweder muß daher die Fuzzy–Menge $\widetilde{A}_2$ erzeugt werden, oder $\widetilde{A}_1$ wird an die Stelle x verschoben (s. Abb. 6.4). Welche Alternative die Bessere ist, hängt von der gegebenen Situation ab.*

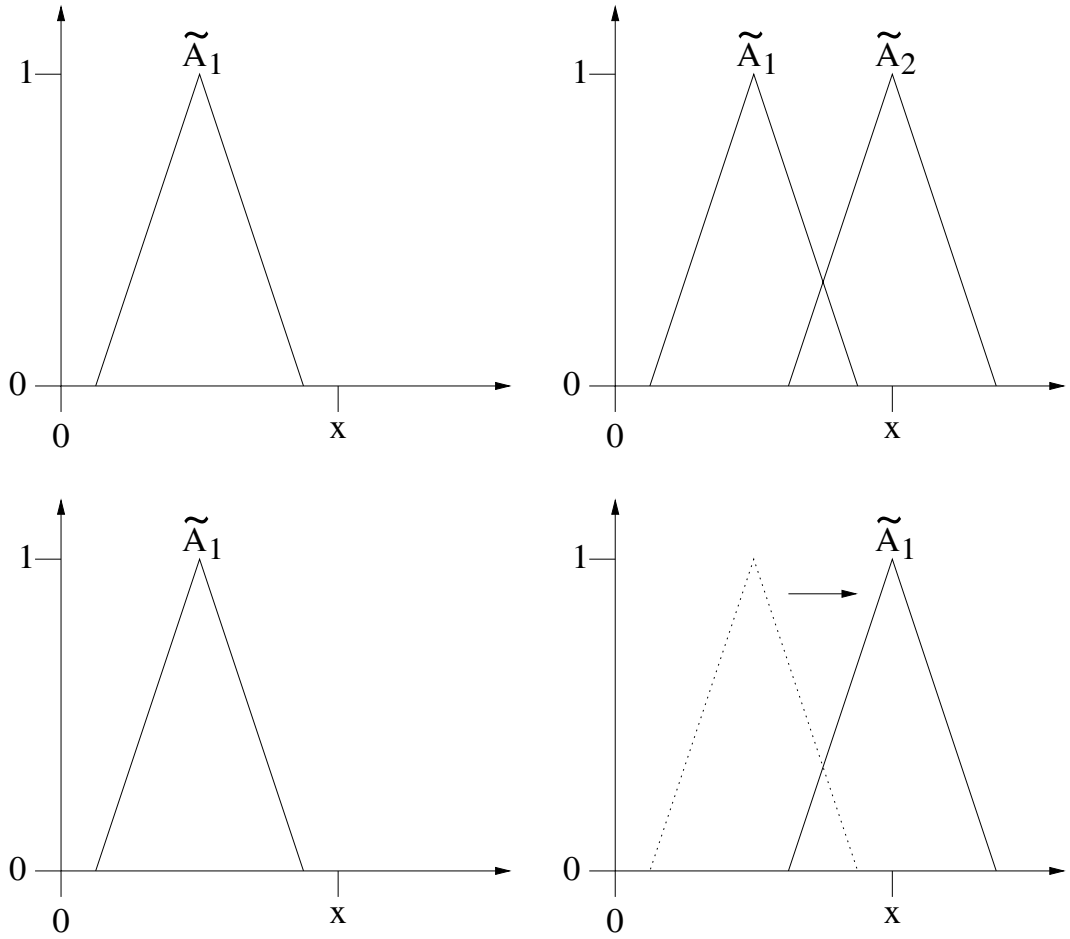

Abbildung 6.4: Erzeugen oder Verschieben einer Fuzzy–Menge

Während die bisher vorgestellten Verfahren auch die Struktur des Netzes verändern, werden beim Fine–Tuning ausschließlich die Gewichte des Netzes modifiziert. Zunächst soll die Frage geklärt werden, welche Konsequenzen eine Änderung der Breite bzw. ein Verschieben einer Partitions–Fuzzy–Menge für einen Fuzzy–Controller hat. Wird eine Fuzzy–Menge in Richtung eines Punktes verschoben, so vergrößert sich dessen Zugehörigkeitsgrad zu dieser Menge; wird die Fuzzy–Menge von dem Punkt weggeschoben, so verringert sich sein Zugehörigkeitsgrad zu dieser Menge. Wird eine Fuzzy–Menge verbreitert, so vergrößern sich die Zugehörigkeitsgrade der Punkte ihres Trägers, außerdem wird der Träger selber verbreitert. Eine Verschmälerung der Fuzzy–Menge bewirkt das Gegenteil.

Bei den Eingabe–Partitions–Mengen hat daher eine Änderung der Breite oder ein Verschieben eine Erhöhung oder Verringerung des Erfüllungsgrades der Prämissen von Regeln, die diese Mengen verwenden, zur Folge. Ein Verschieben einer Ausgabe–Partitions–Menge hat eine Verschiebung des Schwerpunktes der berechneten Ausgabe–Fuzzy–Menge zur Folge. Falls in der verschobenen Menge der maximale Zugehörigkeitsgrad liegt, wird dieser ebenfalls verschoben. Ein Ändern der Breite einer Ausgabe–Fuzzy–Menge erhöht oder reduziert den Anteil dieser Menge am Ergebnis. Außerdem wird damit die erste und die letzte Stelle mit maximalem Zugehörigkeitsgrad verschoben, falls dieser in der veränderten Menge liegt (s. Abb. 6.5). Daher läßt sich der Ausgabe–Wert auf diese Weise stark beeinflussen.

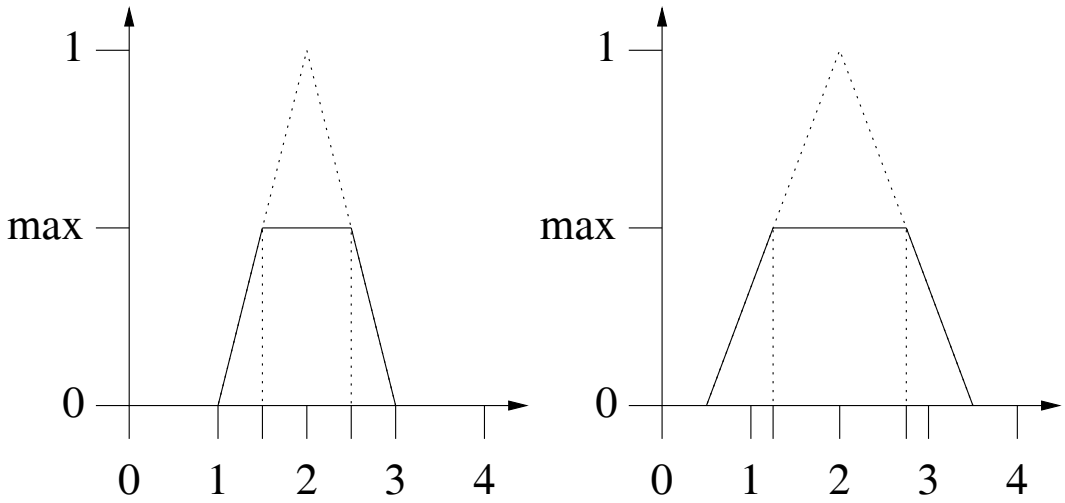

Abbildung 6.5: Verschiebung der ersten und letzten maximalen Stelle

Die angesprochenen Anpassungen der Fuzzy–Mengen sollen mit Hilfe des bei Standard–neuronalen–Netzen wie dem Multilayer–Perzeptron bewährten Gradientenabstiegsverfahrens (s. Kapitel 2) durchgeführt werden. Allerdings entstehen dabei im Gegensatz zum Multilayer–Perzeptron einige spezielle Probleme, die auf den Besonderheiten des MFOS–M–Netzes beruhen. Da es verschiedene Defuzzifizierungsmethoden und Fuzzy–Mengen gibt, werden im Forward–Pass unterschiedliche Funktionen angewendet. Aufgrund der Struktur des verwendeten Netzes gibt es für Schicht 2 teilweise gekoppelte Gewichte, wodurch Schwierigkeiten bei den Gewichts–Änderungen entstehen. Wie dennoch ein erfolgreicher Gradientenabstieg durchgeführt werden kann, wird im Folgenden beschrieben.

6.4.1 Das Gradientenabstiegsverfahren

Für den Einsatz des Gradientenabstiegsverfahrens müssen die partiellen Ableitungen der Fehlerfunktion F berechnet werden (s. auch Kapitel 2). Je nach gewählter Defuzzifizierungs–Methode und verwendeten Fuzzy–Mengen werden im Forward–Pass ver-

schiedene Aktivitäts– und Ausgabe–Funktionen angewendet. Daraus resultieren verschiedene partielle Ableitungen, die jeweils einzeln zu bestimmen sind. Wir wollen uns hier auf Gauß– und Dreiecks–Mengen sowie die Schwerpunkt– und Maximum–Methode in jeder Kombination beschränken. Damit sind die in der Praxis üblichen Methoden abgedeckt; zudem ist das MFOS–M–System mit den gewählten Spezifikationen ein universeller Approximator (s. Abschnitt 6.1 und Abschnitt 3.13).

Somit ergeben sich durch Kombination der Fuzzy–Mengen und Defuzzifizierungsmethoden insgesamt vier Fälle, die alle einzeln betrachtet werden müssen:

- Schwerpunkt–Defuzzifizierung mit Gauß–Mengen
- Schwerpunkt–Defuzzifizierung mit Dreiecks–Mengen
- Maximum–Defuzzifizierung mit Gauß–Mengen
- Maximum–Defuzzifizierung mit Dreiecks–Mengen

Grundsätzlich sind die Gewichte Fuzzy–Mengen, die durch die reellen Parameter m und w für Modalwert und Weite gegeben sind. Daher werden die partiellen Ableitungen der Fehlerfunktion F nach m und w separat berechnet und die Werte entsprechend folgender Formeln angepasst:

$$\Delta m = -\eta \cdot \frac{\partial F}{\partial m}$$

$$\Delta w = -\eta \cdot \frac{\partial F}{\partial w}$$

mit $\eta > 0$ der üblichen Lernrate.

Bemerkung 6.3 *Bei einem Multilayer–Perzeptron wird in jedem Neuron (außer in Schicht 1) mit Hilfe der Gewichte die gewichtete Summe der Eingaben berechnet (s. Kapitel 2). Die verwendeten Gewichte dürfen dabei grundsätzlich auch negativ sein. Dies ist bei einem MFOS–M–Netz nicht für jedes Gewicht der Fall:*

Die Weite w einer Fuzzy–Menge ist stets größer oder gleich 0. Die mit Hilfe des Gradientenabstiegsverfahrens durchgeführten Änderungen ergeben in manchen Fällen negative Werte für w. Daher muß gesichert werden, daß die Weiten nicht negativ werden. Dies ist z.B. durch folgende Modifikation der Formel für Δw gewährleistet:

$$\Delta w = -\eta \cdot \frac{\partial F}{\partial w}, \; falls \; \eta \cdot \frac{\partial F}{\partial w} > w$$

(Für m ist eine Modifikation nicht erforderlich, da Modalwerte negativ sein dürfen).

Bevor wir zu den speziellen Problemen des Gradientenabstiegs kommen, sehen wir uns zunächst einmal die Herleitung der Formeln an:

6.4.2 Herleitung der Formeln

Schicht 4

In Schicht 4 gibt es für jede der m Dimensionen des Ausgabe–Raumes ein Neuron. Bei Verwendung von Gauß–Mengen sowie der Schwerpunkt–Defuzzifizierung gilt:

Die Aktivitätsfunktion von Neuron Nr. i dieser Schicht ist:

$$f_{a\_4,i} = \sum_{v \in V_3(i)} m_{iv} \cdot w_{iv} \cdot z_{3v}$$

Die Ausgabefunktion von Neuron Nr. i dieser Schicht ist:

$$f_{o\_4,i} = \frac{f_{a\_4,i}}{\sum\limits_{v \in V_3(i)} w_{iv} \cdot z_{3v}}$$

Dabei läuft die Summe jeweils über alle Verbindungen von Neuron Nr. i mit den Neuronen aus Schicht 3 ($V_3(i)$). (Zur Erinnerung: in Schicht 3 gibt es für jede Ausgabe–Partitions–Menge ein Neuron, welches mit *dem* Neuron in Schicht 4 verbunden ist, das die zugehörige Ausgabe–Dimension repräsentiert). m_{iv} und w_{iv} sind die Modalwerte bzw. Weiten der Fuzzy–Mengen, die als Gewicht der Verbindung zwischen Neuron Nr. i aus Schicht 4 und Neuron Nr. v aus Schicht 3 verwendet werden. z_{3v} ist die Ausgabe von Neuron Nr. v aus Schicht 3.

Die Fehlerfunktion ist:

$$F = \frac{1}{2} \cdot \sum_{i=1}^{m} (y_i - z_{4i})^2$$

mit y_i der gewünschten Ausgabe und z_{4i} der tatsächlichen Ausgabe von Neuron Nr. i aus Schicht 4.

Für die partielle Ableitung der Fehlerfunktion nach den Parametern der Ausgabe–Fuzzy–Mengen gilt daher:

$$\frac{\partial F}{\partial m_{ik}} = \frac{\partial F}{\partial f_{a\_4,i}} \cdot \frac{\partial f_{a\_4,i}}{\partial m_{ik}} = \frac{\partial F}{\partial f_{o\_4,i}} \cdot \frac{\partial f_{o\_4,i}}{\partial f_{a\_4,i}} \cdot \frac{\partial f_{a\_4,i}}{\partial m_{ik}}$$

mit

$$\frac{\partial F}{\partial f_{o\_4,i}} = \left(\frac{1}{2} \cdot \left(y_i - z_{4i} \right)^2 \right)' = -(y_i - z_{4i})$$

(eindimensionale Kettenregel, z_{4i} als Variable)

$$\frac{\partial f_{o\_4,i}}{\partial f_{a\_4,i}} = \frac{1}{\sum\limits_{v \in V_3(i)} w_{iv} \cdot z_{3v}}$$

$$\frac{\partial f_{a\_4,i}}{\partial m_{ik}} = w_{ik} \cdot z_{3k}$$

und

$$\frac{\partial F}{\partial w_{ik}} = \frac{\partial F}{\partial f_{o\_4,i}} \cdot \frac{\partial f_{o\_4,i}}{\partial w_{ik}} \qquad \text{mit:}$$

$$\frac{\partial F}{\partial f_{o\_4,i}} = -(y_i - z_{4i})$$

$$\frac{\partial f_{o\_4,i}}{\partial w_{ik}} = \frac{m_{ik} \cdot z_{3k} \cdot \left(\sum_{v \in V_3(i)} w_{iv} \cdot z_{3v} \right) - z_{3k} \cdot \left(\sum_{v \in V_3(i)} m_{iv} \cdot w_{iv} \cdot z_{3v} \right)}{\left(\sum_{v \in V_3(i)} w_{iv} \cdot z_{3v} \right)^2}$$

(eindimensionale Quotientenregel)

Die Berechnung der Fehleranteile δ_{4i} der Neuronen dieser Schicht wird im Rahmen der Herleitung der Backpropagation–Formeln für Schicht 2 (s.u.) beschrieben.

Schicht 3:

In dieser Schicht gibt es für jede Ausgabe–Partitions–Menge ein Neuron. Dieses ist jeweils mit *dem* Neuron aus Schicht 4 verbunden, das die zugehörige Ausgabe–Dimension repräsentiert. Dabei wird als Gewicht die Fuzzy–Menge verwendet, die zu dem jeweiligen Neuron aus Schicht 3 gehört. Die Verbindungen mit Schicht 2 besitzen keine Gewichte, daher werden in Schicht 3 nur die Fehler–Anteile δ_{3i} der einzelnen Neuronen berechnet.

Die Aktivitätsfunktion von Neuron Nr. i dieser Schicht ist:

$$f_{a\_3,i} = \sum_{v \in V_2(i)} z_{2v}$$

Die Ausgabefunktion von Neuron Nr. i dieser Schicht ist:

$$f_{o\_3,i} = \min(1, A_{3i})$$

Dabei läuft die Summe über alle Verbindungen von Neuron Nr. i mit Schicht 2 ($V_2(i)$). (Zur Erinnerung: in Schicht 2 gibt es für jede Regel ein Neuron, welches mit *dem* Neuron in Schicht 3 verbunden ist, das die Konklusion dieser Regel repräsentiert.) z_{2v} ist die Ausgabe von Neuron Nr. v aus Schicht 2.

Die Berechnung der Fehleranteile δ_{3i} der Neuronen dieser Schicht wird im Rahmen der Herleitung der Backpropagation–Formeln für Schicht 2 (s.u.) beschrieben.

Schicht 2:

In dieser Schicht gibt es für jede Regel ein Neuron. Dieses ist mit *dem* Neuron aus Schicht 3 verbunden, welches die Konklusion der zugehörigen Regel repräsentiert. Sei j der Index des Neurons aus Schicht 3, das mit Neuron Nr. i aus Schicht 2 verbunden ist. Sei weiterhin l der Index des Neurons aus Schicht 4, das mit Neuron Nr. j aus Schicht 3 verbunden ist.

Die Prämisse der jeweiligen Regel ist durch die Verbindungen mit den Neuronen aus Schicht 1 realisiert. Dabei werden als Gewichte gerade die Eingabe–Partitions–Mengen verwendet, die bei der Prämisse berücksichtigt werden. Falls eine Regel nicht alle Eingabe–Variablen berücksichtigt, gibt es zu den entsprechenden Eingabe–Neuronen (Schicht 1) keine Verbindung. Seien $j_1, \ldots, j_{r(i)}$ die Indizes der Neuronen aus Schicht 1, die mit Neuron Nr. i aus Schicht 2 verbunden sind.

Die Ausgabe von Neuron Nr. i dieser Schicht wird berechnet durch:

$$f_{o\_2,i} = \min\left(\mu_{\widetilde{A}_{ij_1}}(z_{1j_1}), \ldots, \mu_{\widetilde{A}_{ij_{r(i)}}}(z_{1j_{r(i)}})\right)$$

wobei $\widetilde{A}_{ij_*}$ die Fuzzy–Mengen aus der Prämisse von Regel Nr. i sind, und z_{1j_*} die Ausgaben der entsprechenden Neuronen aus Schicht 1.

Für die partielle Ableitung der Fehlerfunktion nach den Parametern der Eingabe–Fuzzy–Mengen gilt daher:

$$\frac{\partial F}{\partial m_{ik}} = \frac{\partial F}{\partial f_{o\_2,i}} \cdot \frac{\partial f_{o\_2,i}}{\partial m_{ik}} = \frac{\partial F}{\partial f_{a\_3,j}} \cdot \frac{\partial f_{a\_3,j}}{\partial f_{o\_2,i}} \cdot \frac{\partial f_{o\_2,i}}{\partial m_{ik}}$$

$$= \frac{\partial F}{\partial f_{o\_3,j}} \cdot \frac{\partial f_{o\_3,j}}{\partial f_{a\_3,j}} \cdot \frac{\partial f_{a\_3,j}}{\partial f_{o\_2,i}} \cdot \frac{\partial f_{o\_2,i}}{\partial m_{ik}}$$

$$= \frac{\partial F}{\partial f_{o\_4,l}} \cdot \frac{\partial f_{o\_4,l}}{\partial f_{o\_3,j}} \cdot \frac{\partial f_{o\_3,j}}{\partial f_{a\_3,j}} \cdot \frac{\partial f_{a\_3,j}}{\partial f_{o\_2,i}} \cdot \frac{\partial f_{o\_2,i}}{\partial m_{ik}}$$

mit

$$\frac{\partial F}{\partial f_{o\_4,l}} = -(y_l - z_{4l}) =: \delta_{4l}$$

(Dies ist der Fehleranteil von Neuron Nr. l aus Schicht 4. δ_{4l} wird daher direkt in diesem Neuron berechnet.)

$$\frac{\partial f_{o\_4,l}}{\partial f_{o\_3,j}} = \frac{m_{lj} \cdot w_{lj} \left(\sum\limits_{v \in V_3(l)} w_{lv} \cdot z_{3v} \right) - w_{lj} \cdot \left(\sum\limits_{v \in V_3(l)} m_{lv} \cdot w_{lv} \cdot z_{3v} \right)}{\left(\sum\limits_{v \in V_3(l)} w_{lv} \cdot z_{3v} \right)^2}$$

(eindimensionale Quotientenregel, z_{3j} als Variable)

$$\frac{\partial f_{o\_3,j}}{\partial f_{a\_3,j}} = (\min(1, f_{a\_3,j}))' = \begin{cases} 1: & \text{falls } f_{a\_3,j} < 1 \\ 0: & \text{sonst} \end{cases}$$

$$\frac{\partial f_{a\_3,j}}{\partial f_{o\_2,i}} = (z_{2i})' = 1$$

Damit läßt sich nun der Fehleranteil δ_{3j} von Neuron Nr. j aus Schicht 3 bestimmen:

$$\delta_{3j} := \delta_{4l} \cdot \frac{\partial f_{o\_4,l}}{\partial f_{o\_3,j}} \cdot \frac{\partial f_{o\_3,j}}{\partial f_{a\_3,j}} \cdot \frac{\partial f_{a\_3,j}}{\partial f_{o\_2,i}}$$

Schließlich ergibt sich:

$$\frac{\partial f_{o\_2,i}}{\partial m_{ik}} = \begin{cases} 0: & \text{falls } \mu_{\widetilde{A}_{ik}}(z_{1k}) \text{ nicht minimal} \\ \left(e^{\frac{-(z_{1k} - m_{ik})^2}{w_{ik}^2}} \right)' & \text{sonst} \end{cases}$$

dabei ist:

$$\left(e^{\frac{-(z_{1k}-m_{ik})^2}{w_{ik}^2}}\right)' = e^{\frac{-(z_{1k}-m_{ik})^2}{w_{ik}^2}} \cdot \frac{2 \cdot (z_{1k} - m_{ik})}{w_{ik}^2}$$

(eindimensionale Kettenregel, m_{ik} als Variable)

Analog gilt:

$$\frac{\partial F}{\partial w_{ik}} = \frac{\partial F}{\partial f_{o\_4,l}} \cdot \frac{\partial f_{o\_4,l}}{\partial f_{o\_3,j}} \cdot \frac{\partial f_{o\_3,j}}{\partial f_{a\_3,j}} \cdot \frac{\partial f_{a\_3,j}}{\partial f_{o\_2,i}} \cdot \frac{\partial f_{o\_2,i}}{\partial w_{ik}}$$

mit

$$\frac{\partial f_{o\_2,i}}{\partial w_{ik}} = \begin{cases} 0: & \text{falls } \mu_{\tilde{A}_{ik}}(z_{1k}) \text{ nicht minimal} \\ \left(e^{\frac{-(z_{1k}-m_{ik})^2}{w_{ik}^2}}\right)' & \text{sonst} \end{cases}$$

dabei ist:

$$\left(e^{\frac{-(z_{1k}-m_{ik})^2}{w_{ik}^2}}\right)' = e^{\frac{-(z_{1k}-m_{ik})^2}{w_{ik}^2}} \cdot \frac{2 \cdot (z_{1k} - m_{ik})^2}{w_{ik}^3}$$

(eindimensionale Kettenregel, w_{ik} als Variable)

Werden statt Gauß–Mengen Dreiecks–Mengen verwendet, ändern sich lediglich die Berechnungen für Schicht 2 wie folgt:

$$\frac{\partial f_{o\_2,i}}{\partial m_{ik}} = \begin{cases} 0: & \text{falls } \mu_{\tilde{A}_{ik}}(z_{1k}) \text{ nicht minimal} \\ \left(1 - \frac{m_{ik}-z_{1k}}{w_{ik}}\right)' & \text{falls } m_{ik} - w_{ik} \leq z_{1k} \leq m_{ik} \\ \left(1 + \frac{m_{ik}-z_{1k}}{w_{ik}}\right)' & \text{falls } m_{ik} < z_{1k} \leq m_{ik} + w_{ik} \end{cases}$$

dabei ist:

$$\left(1 - \frac{m_{ik} - z_{1k}}{w_{ik}}\right)' = -\frac{1}{w_{ik}}$$

$$\left(1 + \frac{m_{ik} - z_{1k}}{w_{ik}}\right)' = \frac{1}{w_{ik}}$$

(m_{ik} als Variable)

und analog

$$\frac{\partial f_{o\_2,i}}{\partial w_{ik}} = \begin{cases} 0: & \text{falls } \mu_{\widetilde{A}_{ik}}(z_{1k}) \text{ nicht minimal} \\ \left(1 - \frac{m_{ik} - z_{1k}}{w_{ik}}\right)' & \text{falls } m_{ik} - w_{ik} \leq z_{1k} \leq m_{ik} \\ \left(1 + \frac{m_{ik} - z_{1k}}{w_{ik}}\right)' & \text{falls } m_{ik} < z_{1k} \leq m_{ik} + w_{ik} \end{cases}$$

dabei ist:

$$\left(1 - \frac{m_{ik} - z_{1k}}{w_{ik}}\right)' = \frac{m_{ik} - z_{1k}}{w_{ik}^2}$$

$$\left(1 + \frac{m_{ik} - z_{1k}}{w_{ik}}\right)' = -\frac{m_{ik} - z_{1k}}{w_{ik}^2}$$

(w_{ik} als Variable)

Wird zur Defuzzifizierung die Maximum–Methode verwendet, so ist in Schicht 4 die Aktivitätsfunktion gegeben durch $f_{a\_4,i} = v$, wobei z_{3v} maximal ist über alle $v \in V_3(i)$. Die Ausgabefunktion ist $f_{o\_4,i} = m_{iv}$, mit $v = f_{a\_4,i}$.

Daher gilt in diesem Fall:

Schicht 4:

$$\frac{\partial F}{\partial m_{ik}} = \frac{\partial F}{\partial f_{o\_4,i}} \cdot \frac{\partial f_{o\_4,i}}{\partial m_{ik}}$$

mit

$$\frac{\partial F}{\partial f_{o\_4,i}} = -(y_i - z_{4i})$$

sowie

$$\frac{\partial f_{o\_4,i}}{\partial m_{ik}} = \begin{cases} 1 : k = v \\ 0 : k \neq v \end{cases}$$

und

$$\frac{\partial F}{\partial w_{ik}} = \frac{\partial F}{\partial f_{o\_4,i}} \cdot \frac{\partial f_{o\_4,i}}{\partial w_{ik}}$$

mit

$$\frac{\partial F}{\partial f_{o\_4,i}} = -(y_i - z_{4i})$$

sowie

$$\frac{\partial f_{o\_4,i}}{\partial w_{ik}} = 0$$

Schicht 2:

$$\frac{\partial F}{\partial m_{ik}} = \frac{\partial F}{\partial f_{o\_4,l}} \cdot \frac{\partial f_{o\_4,l}}{\partial f_{o\_3,j}} \cdot \frac{\partial f_{o\_3,j}}{\partial f_{a\_3,j}} \cdot \frac{\partial f_{a\_3,j}}{\partial f_{o\_2,i}} \cdot \frac{\partial f_{o\_2,i}}{\partial m_{ik}}$$

und

$$\frac{\partial F}{\partial w_{ik}} = \frac{\partial F}{\partial f_{o\_4,l}} \cdot \frac{\partial f_{o\_4,l}}{\partial f_{o\_3,j}} \cdot \frac{\partial f_{o\_3,j}}{\partial f_{a\_3,j}} \cdot \frac{\partial f_{a\_3,j}}{\partial f_{o\_2,i}} \cdot \frac{\partial f_{o\_2,i}}{\partial w_{ik}}$$

Dabei unterscheidet sich lediglich die Berechnung von $\partial f_{o\_4,l}/\partial f_{o\_3,j}$ von der für die Schwerpunkt–Methode durchgeführten Berechnung: Seien $v_1, \ldots, v_{r(l)}$ die Indizes der Neuronen aus Schicht 3, die mit Neuron Nr. l aus Schicht 4 verbunden sind. Dann gilt:

$$f_{o\_4,l} = \begin{cases} m_{lv_1} : & \text{falls } z_{3v_1} \text{ maximal} \\ \vdots & \\ m_{lv_{r(l)}} : & \text{falls } z_{3v_{r(l)}} \text{ maximal} \end{cases}$$

D.h., der Ausgabewert ist eine Konstante, die durch Fallunterscheidung nach z_{3v_*} bestimmt wird. Jedoch gilt für jedes v_*: $(m_{lv_*})' = 0$, wenn mit z_{3v_*} als Variable die Ableitung bestimmt wird. $\partial f_{o\_4,l}/\partial f_{o\_3,j}$ ist also eigentlich konstant Null. Allerdings ist es mit diesem Wert nicht möglich, eine Änderung der Eingabe–Partitions–Mengen zu bewirken. Daher wird in diesem Fall folgendes festgelegt:

$$\frac{\partial f_{o\_4,l}}{\partial f_{o\_3,j}} \simeq \begin{cases} 1 : & \text{falls } z_{3j} \text{ maximal} \\ 0 : & \text{sonst} \end{cases}$$

Auf diese Weise wird die bei der Berechnung von $f_{o\_4,l}$ durchgeführte Fallunterscheidung angemessen berücksichtigt. Außerdem entspricht dieses dem Wesen der Maximum–Methode, einen Wert auszuwählen und alle anderen zu ignorieren. Daher ergibt sich so ein sinnvoller Wert für $\partial f_{o\_4,l}/\partial f_{o\_3,j}$, der gute Ergebnisse erwarten läßt.

Bei der Implementierung gestalten sich die erforderlichen Berechnungen wesentlich einfacher, da viele der benötigten Werte (insbesondere die Summen) bereits im Forward–Pass berechnet werden und daher im Backward–Pass verfügbar sind.

6.4.3 Behandlung der gekoppelten Gewichte

Ein MFOS–M–Netz verwendet für die Verbindungen von Schicht 1 zu Schicht 2 z.T. gekoppelte Gewichte: jedes Neuron in Schicht 2 entspricht einer linguistischen Regel. Als Gewichte der Verbindungen mit Schicht 1 werden genau die Partitions–Fuzzy–Mengen verwendet, die in der Prämisse der Regel vorkommen. Beziehen sich mehrere Regeln auf dieselbe Partitions–Fuzzy–Menge, so erhalten alle entsprechenden Verbindungen genau diese Fuzzy–Menge als Gewicht.

Beispiel 6.13
Regel 1: IF x_1 = $\widetilde{A}_{11}$ UND x_2 = $\widetilde{A}_{21}$ THEN y = $\widetilde{B}_1$
Regel 2: IF x_1 = $\widetilde{A}_{11}$ UND x_2 = $\widetilde{A}_{22}$ THEN y = $\widetilde{B}_2$

Beide Regeln verwenden die Eingabe–Partitions–Menge $\widetilde{A}_{11}$. Daher wird diese Menge bei den beiden Neuronen aus Schicht 2, die diese Regeln repräsentieren, als Gewicht der Verbindung mit Neuron Nr. 1 aus Schicht 1 verwendet. Tatsächlich ist es so, daß die Fuzzy–Mengen in einem eigenen Partitionierungs–Objekt gespeichert werden. Alle Berechnungen, die die Menge $\widetilde{A}_{11}$ betreffen, arbeiten mit denselben zugehörigen Daten

dieses Objekts. Jede Veränderung eines Gewichtes bewirkt eine Veränderung der Daten im Partitionierungs–Objekt selber.

Gewichte der Verbindungen zwischen Schicht 1 und Schicht 2, die der selben Fuzzy–Menge entsprechen, verwenden so dieselben Daten aus einem einzigen Objekt. Somit sind diese Gewichte gekoppelt. Jede Veränderung eines dieser Gewichte bewirkt exakt die gleiche Änderung der anderen, daran gekoppelten Gewichte. Aufgrund der Konstruktion eines MFOS–M–Netzes (s. Abschnitt 6.2) gibt es bei den Ausgabe–Partitions–Mengen keine solche Koppelung, da für jede dieser Mengen ein eigenes Neuron in Schicht 3 existiert.

Daß die Partitions–Fuzzy–Mengen ausschließlich in einem eigenen Partitionierungs–Objekt existieren, hat zwei Hauptvorteile:

1. Es liegt zu jedem Zeitpunkt des Trainings eine reguläre Partitionierung vor, die separat gespeichert und weiterverwendet werden kann. Dies ist Voraussetzung für die Rücktransformation des MFOS–M–Netzes in einen gewöhnlichen Fuzzy–Controller (s. Abschnitt 6.4.5).

2. Die Anzahl der Partitions–Mengen bleibt geringer, als wenn jedes Neuron aus Schicht 2 seine eigenen Gewichte (Partitions–Mengen) verwenden würde.

Für die Pre–Tuning–Verfahren (s. Abschnitt 6.3) ergibt sich aus der Koppelung kein Nachteil. Sämtliche dieser Verfahren wurden erfolgreich getestet (s. die Beispiele in Abschnitt 6.3 und zu ausführlicheren Tests [Niendieck, 1998]). Falls etwa eine Regel benötigt wird, die eine noch nicht berücksichtigte Situation repräsentiert (dargestellt durch die gewählten Fuzzy–Mengen der Prämisse), wird sie automatisch erzeugt. Somit sind auch Prämissen mit neuen Kombinationen von Eingabe–Partitions–Mengen erzeugbar, d.h. es ist möglich, die Koppelung aufzuheben bzw. zu verändern.

Erst bei Anwendung des Gradientenabstiegsverfahrens können durch die Koppelung Probleme entstehen: jedes Neuron aus Schicht 2 berechnet einen eigenen Wert für die Änderung der Gewichte seiner Verbindungen mit Schicht 1. Dabei ergeben sich i.A. für die verschiedenen Neuronen unterschiedliche Werte. Da die Änderungen bei gekoppelten Gewichten jeweils gemeinsam gelten, können sie sich z.B. gegenseitig aufheben. Eine korrekte Einstellung ist daher eventuell unmöglich.

Aus diesem Grund wurde eine zweite Version des MFOS–M–Netzes implementiert, bei der jedes Neuron aus Schicht 2 seine eigenen Gewichte hat. Damit gibt es beim Gradientenabstiegsverfahren keine gegenseitige Beeinflussung der Änderungen untereinander, so daß eine korrekte Einstellung der Partitions–Mengen zu erwarten ist.

Daß jedes Neuron in Schicht 2 seine eigenen, unabhängigen Gewichte besitzt, hat zwei Hauptnachteile:

1. Es liegt keine übliche Partitionierung mehr vor, die separat gespeichert und auf einfache Weise weiter verwendet werden kann, da sich nun jede Regel auf ihre eigenen Fuzzy–Mengen bezieht. Somit ist eine Rücktransformation des MFOS–M–Netzes in einen gewöhnlichen Fuzzy–Controller nicht möglich.

2. Aus dem gleichen Grund ist die Anzahl der Partitions–Fuzzy–Mengen sehr groß.

Die nachfolgend vorgestellten Untersuchungen zeigen, unter welchen Bedingungen ein Einsatz der Version mit gekoppelten Gewichten erfolgreich ist, und wann freie Gewichte erforderlich sind.

6.4.4 Untersuchung des Gradientenabstiegsverfahrens

Das Gradientenabstiegsverfahren ist grundsätzlich zur Feinabstimmung der Partitions–Fuzzy–Mengen vorgesehen, um nach Erstellung einer korrekten Regelbasis eine weitere Verringerung des Fehlers zu erreichen. Vor seiner Anwendung zur Feinabstimmung sollten daher die Pre–Tuning–Verfahren (s. Abschnitt 6.3) nach Bedarf eingesetzt werden. In bestimmten Fällen ist jedoch eine Anwendung des Gradientenabstiegsverfahrens vor dem Einsatz der Pre–Tuning–Verfahren besser, da so eventuell vermieden wird, nicht unbedingt erforderliche zusätzliche Partitions–Fuzzy–Mengen zu erzeugen. Eine Analyse der Wechselwirkungen der einzelnen Verfahren und Empfehlungen zur optimalen Reihenfolge werden in Abschnitt 6.6 vorgestellt.

Da bei Anwendung des Gradientenabstiegsverfahrens lediglich die vorhandenen Partitions–Fuzzy–Mengen (Modalwert und Weite) verändert werden, ist in jedem Fall das Vorhandensein einer korrekten Regelbasis Voraussetzung für einen erfolgreichen Einsatz zur Feinabstimmung. Ebenso muß die Anzahl der Partitions–Fuzzy–Mengen stimmen. Da sich beides durch die Pre–Tuning–Verfahren erreichen läßt, wurde dies bei den folgenden Untersuchungen vorausgesetzt.

Es wurden für alle theoretisch möglichen Situationen Tests durchgeführt. Aufgrund der Herleitung ergeben sich die vier bereits erwähnten Wahlmöglichkeiten zur Betreibung des Netzes, d.h. der angewendeten Funktionen:

- Schwerpunkt–Defuzzifizierung mit Gauß–Mengen
- Schwerpunkt–Defuzzifizierung mit Dreiecks–Mengen
- Maximum–Defuzzifizierung mit Gauß–Mengen
- Maximum–Defuzzifizierung mit Dreiecks–Mengen

Weiterhin ist zu unterscheiden, ob die Eingabe– bzw. die korrekten Ausgabe–Werte innerhalb der Träger der Partitions–Fuzzy–Mengen liegen oder nicht. Falls ein Wert nicht im Träger einer Menge liegt, ist der Zugehörigkeitsgrad zu dieser Menge und

somit der Wert der partiellen Ableitung 0. Ist dies bei *allen* Mengen der Fall, ist eine Änderung der Mengen mit dem Gradientenabstiegsverfahren nicht möglich. Schließlich ist zu berücksichtigen, ob die Version mit freien oder gekoppelten Gewichten verwendet wird. Entsprechend der Sichtweise des Anwenders, der die Defuzzifizierungsmethode und die Art der Fuzzy–Mengen festlegt, wurde zur Darstellung der Ergebnisse eine Unterteilung gemäß der erstgenannten vier Fälle vorgenommen:

Schwerpunkt–Defuzzifizierung mit Gauß–Mengen

Hier zeigen sich in allen Fällen sehr gute Ergebnisse. Unabhängig davon, wo die Partitions–Mengen liegen, ist in jedem Fall mit beiden Netz–Versionen ein erfolgreiches Training möglich. Es kommt vor, daß die Version mit freien Gewichten zu einem etwas geringeren Fehler führt bzw. etwas schneller ist. Aufgrund der genannten Nachteile sind diese sehr geringen Unterschiede jedoch kein Grund, die Version mit freien Gewichten vorzuziehen.

Es gibt zwei Gründe für dieses positive Verhalten:

1. Der Träger von Gauß–Mengen ist ganz $\mathbb{R}$ (die Gauß–Funktion nimmt den Wert 0 nie an). Daher liegt *jeder* Eingabe– und Ausgabe–Wert innerhalb des Trägers aller Partitions–Mengen der zugehörigen Ein– bzw. Ausgabe–Dimension.

2. Die Schwerpunkt–Defuzzifizierung ist (bis auf Nullstellen des Nenners) stetig und zeigt ein "gleichmäßiges" Ausgabeverhalten. Geringe Änderungen bei den Eingaben haben nur eine geringe Änderung des Schwerpunktes zur Folge.

Schwerpunkt–Defuzzifizierung mit Dreiecks–Mengen

Hier zeigen sich sehr gute Ergebnisse, wenn die Eingabe–Werte innerhalb der Träger der Eingabe–Partitions–Mengen liegen. Wo die gewünschten Ausgabe–Werte liegen, ist dabei unerheblich. Es gibt wie im vorherigen Fall nur unwesentliche Unterschiede zwischen beiden Netz–Versionen. Nur wenn es Eingabe–Partitions–Mengen gibt, für die bei *allen* Beispielen die Eingabe–Werte außerhalb der Träger liegen, ist ein erfolgreiches Training nicht möglich. Ursache hierfür ist, daß der Funktionswert und der Wert der Ableitung bei diesen Beispielen 0 ist. Daher kommt es hier zu gar keiner Änderung der betroffenen Mengen. Aufgrund der Formeln für den Backward–Pass gibt es bei den Ausgabe–Partitions–Mengen ein solches Problem nicht.

Dieses Verhalten stellt allerdings keine wesentliche Einschränkung dar: falls es Eingabe–Partitions–Mengen gibt, für die bei allen Trainingsbeispielen die Eingaben außerhalb der Träger liegen, ist der Erfüllungsgrad der Regeln, die diese Mengen verwenden, immer 0. Durch Anwendung der Verfahren zur Erzeugung von Regeln bzw. Partitions–Mengen sowie zum Löschen von Regeln ergibt sich folgendes:

1. Es gibt für jedes Beispiel Regeln mit einem Erfüllungsgrad über einer Schranke größer als Null.

2. Regeln, die immer einen Erfüllungsgrad von Null ergeben, werden gelöscht. Daher tritt nach Anwendung dieser Verfahren dieses Problem nicht mehr auf. Da das Gradientenabstiegsverfahren ohnehin zur Feinabstimmung vorgesehen ist, nachdem die anderen Verfahren durchgeführt wurden, liegt keine Beschränkung der Einsatzmöglichkeiten vor.

Maximum–Defuzzifizierung mit Gauß–Mengen

Die Netz–Version mit freien Gewichten ermöglicht in diesem Fall in jeder Situation ein erfolgreiches Training. Mit gekoppelten Gewichten ist dies oft nicht möglich. Bei bestimmten Eingabe–Werten, die zu verschiedenen Ausgaben führen sollen, ergibt sich immer dieselbe Ausgabe. Zwar ist bei den meisten Werten der Fehler sehr gering, aber diese Ausnahmen haben einen zu hohen Fehler, der sich nicht beseitigen läßt. Daher kann in diesem Fall nicht auf die Verwendung freier Gewichte verzichtet werden. Folgendes Beispiel verdeutlicht das Problem:

Beispiel 6.14 *Ausgabe soll die Summe von zwei Eingaben sein, Trainingsbeispiele sind:*

((Eingabe 1, Eingabe 2), Ausgabe):

$$\Big((2,5),7\Big), \Big((2,7),9\Big), \Big((4,5),9\Big), \Big((4,7),11\Big).$$

Zum Testen der Gradientenabstiegs–Methode wurden Gauß–Mengen verwendet, die ähnlich zu den Ein– und Ausgaben sind: (Modalwert, Weite):

$(1.9, 2)$, $(4.1, 2.1)$, $(5.1, 2)$, $(6.9, 2)$, $(9.1, 2.1)$, $(11.1, 1.9)$.

Die Regelbasis sorgt für eine Abbildung der Eingaben auf die korrekte Ausgabe.

Berechnete Ergebnisse unter Verwendung der Maximum–Methode sind vor Anwendung des Gradientenabstiegsverfahrens: (korrekte Ausgabe, berechnete Ausgabe):

$(7, 6.9), (9, 9.1), (9, 9.1), (11, 11.1)$.

Das Gradientenabstiegsverfahren sollte diese Werte weiter verbessern. Doch unter Verwendung gekoppelter Gewichte ändern sich die Ausgaben zu:

$(7, 6.9998), (9, 8.9375), (9, 8.9375), (11.8.9375)$.

Drei Ausgabe–Werte wurden verbessert, jedoch ist die vierte Ausgabe absolut falsch. Das System kann nicht mehr zwischen den Ausgaben 9 und 11 unterscheiden.

Beim Training mit freien Gewichten ergeben sich dagegen sehr gute Ergebnisse:

$(7, 6.999), (9, 9), (9, 9), (11, 11.0001)$.

Der Grund für dieses Verhalten liegt in der Maximum–Methode selber. Ihr Ausgabeverhalten zeigt generell sprunghafte Änderungen (s. Abschnitt 6.4.4, Auswertung und Erklärung der Ergebnisse), so daß eine gleichmäßige Anpassung der Gewichte nicht möglich ist. Bei Verwendung von freien Gewichten ergeben sich nach dem Training (geringe) Unterschiede zwischen ursprünglich gleichen Eingabe–Partitions–Mengen. Diese geringen Unterschiede führen zu ebenfalls geringen Unterschieden bei den Zugehörigkeitsgraden und damit bei den Erfüllungsgraden der Prämissen der betroffenen Regeln. Dadurch kann allerdings das Maximum der berechneten Ausgabe–Fuzzy–Menge an einer ganz anderen Stelle liegen, so daß sich für den Ausgabe–Wert große Unterschiede ergeben, die zu korrekten Ergebnissen führen. Bei Verwendung von gekoppelten Gewichten ist eine solche Entwicklung dagegen ausgeschlossen.

Maximum–Defuzzifizierung mit Dreiecks–Mengen

Auch hier ist die Verwendung von freien Gewichten unverzichtbar. Wie schon im 2. Fall ist ein erfolgreiches Training nicht möglich, wenn es Eingabe–Partitions–Mengen gibt, für die bei allen Beispielen die Eingabe–Werte außerhalb der Träger liegen. Die Gründe dafür sind die gleichen, ebenso stellt dies auch hier keine wirkliche Einschränkung dar. Es zeigt sich, daß die Ergebnisse nicht so gut sind, wie in den anderen Fällen. Zwar tritt eine deutliche Verringerung des Fehlers ein, aber es werden nicht die sehr guten Resultate der anderen Fälle erreicht.

Ein Grund hierfür ist die generell problematische Maximum–Methode. Zusätzlich könnte es sich (gerade bei Verwendung der Maximum–Methode) negativ auswirken, daß innerhalb der Eingabe–Partitionierungen die Übergänge zwischen den einzelnen Partitions–Mengen mit Gauß–Mengen "weicher" sind als mit Dreiecks–Mengen. Dies liegt an der geschwungen Form der Gauß–Mengen, während Dreiecks–Mengen stückweise linear sind (vgl. Abb. 3.1).

Auswertung und Erklärung der Ergebnisse

Es zeigt sich, daß bei Verwendung von freien Gewichten ehemals identische Partitions–Fuzzy–Mengen für eine optimale Anpassung unterschiedlich eingestellt werden. Allerdings sind die Unterschiede nur gering. Alle Fuzzy–Mengen, die aus der gleichen Ursprungs–Menge hervorgehen, haben nach Beendung des Trainings eine ähnliche Form und Position. Daß eine Menge so weit verschoben wird, daß sie die Position einer anderen Menge einnimmt, kommt nicht vor. Abb. 6.6 verdeutlicht dies.

Die Menge Nr. 1 wird unabhängig für zwei Regeln (zwei Neuronen aus Schicht 2) passend eingestellt. Oben ist ein mögliches Ergebnis zu sehen, die Ergebnis–Mengen

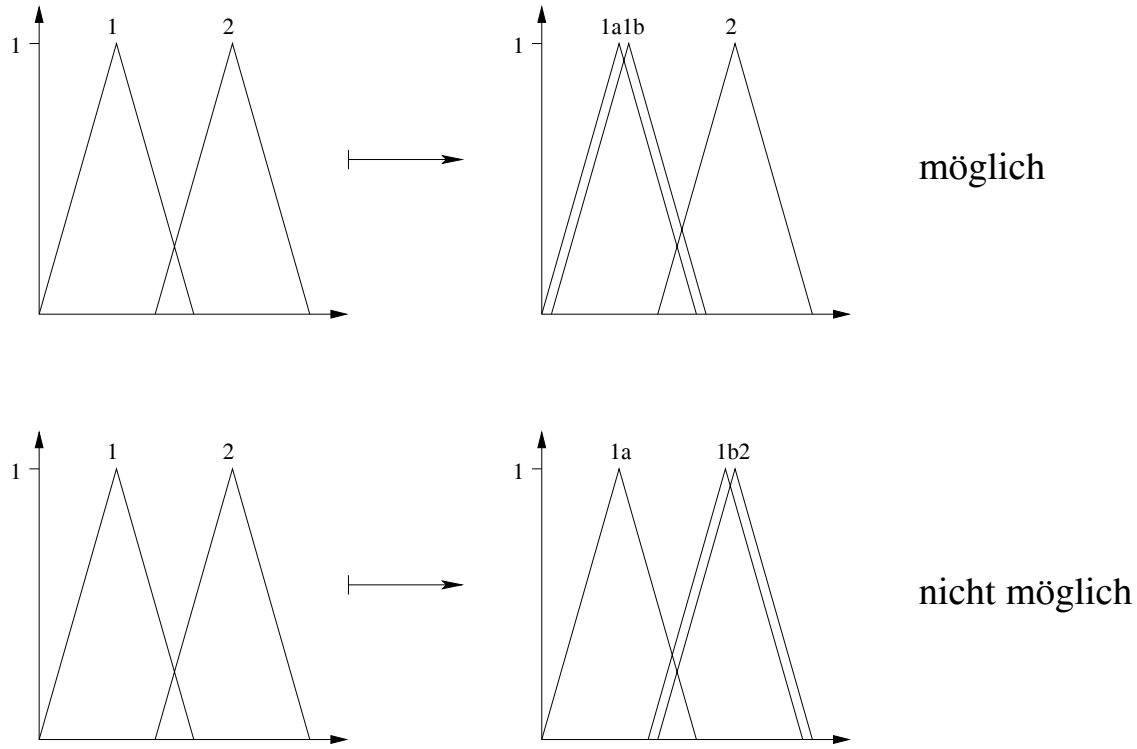

Abbildung 6.6: Mögliche und nicht mögliche Modifikationen einer Fuzzy–Menge

Nr. $1a$ und $1b$ sind ähnlich. Unten ist ein Ergebnis zu sehen, das bei der Feinabstimmung der Partitions–Fuzzy–Mengen ausgeschlossen ist. Ergebnis–Menge Nr. $1b$ ist so weit verschoben, daß sie der Menge Nr. 2 ähnelt. Aufgrund der Voraussetzung für die Anwendung des Gradientenabstiegsverfahrens zur Feinabstimmung ist dieses Ergebnis nicht möglich. Wenn die Regeln korrekt eingestellt sind, sind die verwendeten Mengen "ungefähr" richtig. Sollte Menge Nr. 1 von einer Regel an der Position von Menge Nr. 2 benötigt werden, wäre vorher die Menge Nr. 2 für diese Regel ausgewählt worden.

Die einzige Situation, in der eine derart weite Verschiebung vorkommen kann, ist die Anwendung des Gradientenabstiegsverfahrens vor den Pre–Tuning–Verfahren, welche in Abschnitt 6.6 beschrieben wird. Unter der hier betrachteten Voraussetzung, das Fine–Tuning nach dem Pre–Tuning durchzuführen, ist eine derart starke Verschiebung in jedem Fall ausgeschlossen.

Obwohl sich also bei der Verwendung von freien Gewichten nur geringe Unterschiede zwischen ursprünglich gleichen Partitions–Fuzzy–Mengen ergeben, ist bei Anwendung der Maximum–Methode eine korrekte Einstellung bei Verwendung von gekoppelten Gewichten nicht möglich. Wie bereits angesprochen wurde, liegt dies im sprunghaften Verhalten der Maximum–Methode begründet. Diese bestimmt eine Stelle der Ausgabe–Fuzzy–Menge mit maximalem Zugehörigkeitsgrad. Die Zugehörigkeitsgrade hängen aber direkt von den Erfüllungsgraden der Prämissen der Regeln ab (vgl. Ka-

pitel 3). Diese wiederum hängen von den Zugehörigkeitsgraden der Eingabe–Daten zu den Eingabe–Partitions–Mengen ab. Daher kann bereits eine geringe Änderung dieser Mengen dazu führen, daß eine andere Regel den maximalen Zugehörigkeitsgrad bekommt und somit das Maximum einen großen Sprung zu einer anderen Stelle macht. Abb. 6.7 verdeutlicht dies.

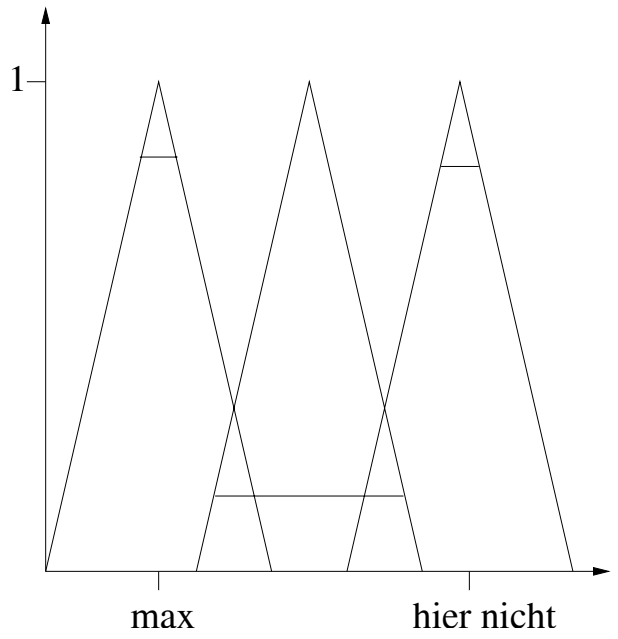

Abbildung 6.7: Ergebnis der Maximum–Methode

Hier liegt das Maximum in der linken Menge, da die Schnitthöhe der rechten Menge geringfügig kleiner ist. Schon eine leichte Verschiebung der Eingabe–Partitions–Mengen kann bewirken, daß die größte Schnitthöhe in der rechten Menge liegt. Das Maximum wäre fast an das andere Ende der Partitionierung "gesprungen". Bei der Schwerpunkt–Methode kommen solche Sprünge hingegen nicht vor. Der Schwerpunkt liegt hier in der mittleren Menge. Eine Änderung der Eingabe–Partitions–Mengen wie zuvor wird hier nur eine geringe Verschiebung des Schwerpunktes bewirken (s. Abb. 6.8).

Dieses prinzipiell unterschiedliche Verhalten der Maximum–Methode und der Schwerpunkt–Methode erklärt, warum bei der Erstgenannten schon geringste Unterschiede der Eingabe–Partitions–Mengen zu großen Unterschieden bei der Ausgabe führen können. Es ist daher unvermeidlich, bei Anwendung der Maximum–Methode für den Einsatz des Gradientenabstiegsverfahrens auf die Verwendung von gekoppelten Gewichten zu verzichten. Dies ist unerwünscht, da somit eine Rücktransformation des MFOS–M–Netzes in einen Fuzzy–Controller nicht mehr möglich ist. In Abschnitt 6.4.5 wird eine Lösung dieses Problems vorgestellt.

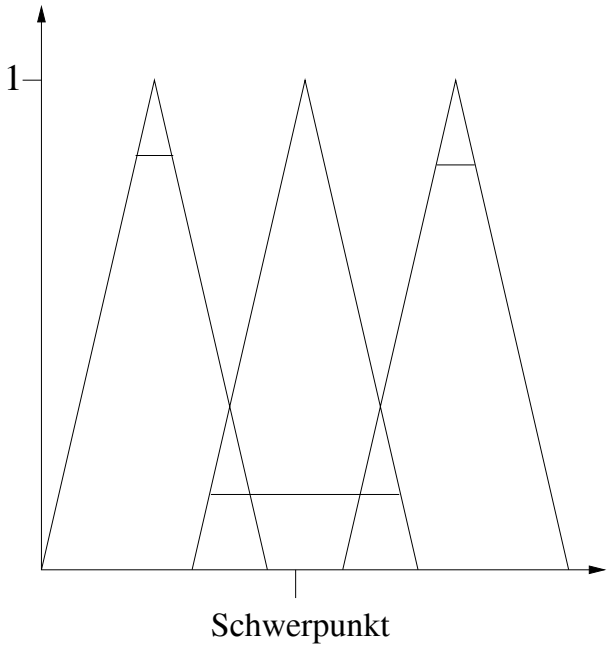

Abbildung 6.8: Ergebnis der Schwerpunkt–Methode

Fazit der Untersuchungen

Das Gradientenabstiegsverfahren unter Anwendung der Schwerpunkt–Methode ist beim Training generell unproblematisch. Damit sind in jedem Fall gute Ergebnisse zu erwarten. Aufgrund ihres sprunghaften Verhaltens ist das Gradientenabstiegsverfahren unter Anwendung der Maximum–Methode schwieriger. Das gilt insbesondere, wenn Dreiecks–Mengen verwendet werden. Außerdem ist im Fall der Maximum–Methode die eigentlich unerwünschte Verwendung von freien Gewichten erforderlich. Damit sind allerdings auch mit der Maximum–Methode gute Ergebnisse zu erreichen.

6.4.5 Alternative Struktur eines MFOS–M–Netzes

In Abschnitt 6.2 wurde die Abbildung–1 definiert, die einen gegebenen Fuzzy–Controller in ein funktional äquivalentes MFOS–M–Netz transformiert. Wie beschrieben, werden die Regeln und Partitions–Mengen durch die Struktur des Netzes bzw. die Gewichte repräsentiert. Die vorgestellten Trainingsverfahren modifizieren die Struktur und die Gewichte, um das Ein– Ausgabe–Verhalten zu optimieren. Das modifizierte MFOS–M–Netz repräsentiert immer noch einen Fuzzy–Controller, die optimierte Version des ursprünglichen Fuzzy–Controllers. Bei Verwendung von gekoppelten Gewichten für die Verbindungen zwischen Schicht 1 und Schicht 2 lassen sich anhand der Gewichte und der Netzstruktur in direkter Umkehrung von Abbildung–1 Partitionierungen und Regeln extrahieren. Somit ist eine Rücktransformation des MFOS–M–Netzes in einen (optimierten) Fuzzy–Controller möglich.

Definition 6.2 *Abbildung–2: MFOS–M $\longrightarrow$ FC ist die im Folgenden beschriebene Portierung eines MFOS–M–Netzes auf einen Fuzzy–Controller.*

Der Mamdani–Controller wird wie folgt aus einem MFOS–M–Netz gebildet:

Jedes Neuron in Schicht 2 repräsentiert eine Regel. Die Verbindungen zu Schicht 1 ergeben die Prämisse. Die Gewichte dieser Verbindungen entsprechen den Eingabe–Partitions–Mengen. Die Verbindung zu Schicht 3 entspricht der Konklusion der Regel. Das Gewicht der von dem betroffenen Neuron aus Schicht 3 weitergehenden Verbindung zu Schicht 4 entspricht der Ausgabe–Partitions–Menge.

Ist Neuron R_k aus Schicht 2 mit den Neuronen $1, \ldots, n$ aus Schicht 1 über die Gewichte $\widetilde{A}_{1*}, \ldots, \widetilde{A}_{n*}$ verbunden und mit dem Neuron aus Schicht 3, welches die Ausgabe–Partitions–Menge $\widetilde{B}_{1*}$ aus Ausgabe–Dimension 1 repräsentiert, so ergibt sich Regel R_k wie folgt:

R_k : `IF` $x_1 = \widetilde{A}_{1*}$ `UND` ... `UND` $x_n = \widetilde{A}_{n*}$ `THEN` $y_1 = \widetilde{B}_{1*}$

Falls Neuron R_k mit einzelnen Eingabe–Neuronen nicht verbunden ist, entfallen die entsprechenden Eingaben bei der Regel. D.h. die Regel berücksichtigt nicht alle Eingabe–Werte.

Beispiel 6.15 *Sei das Neuron R_4 aus Schicht 2 mit folgenden Verbindungen gegeben:*

- *es gibt je eine Verbindung mit Neuron 1 und Neuron 2 aus Schicht 1 mit den Eingabe–Partitions–Mengen $\widetilde{A}_{13}$ bzw. $\widetilde{A}_{21}$ als Gewicht*
- *es gibt eine Verbindung mit dem Neuron aus Schicht 3, welches die Ausgabe–Partitions–Menge $\widetilde{B}_{1*}$ repräsentiert*

Daraus läßt sich Regel R_4 extrahieren:

R_4 : ***IF*** $x_1 = \widetilde{A}_{13}$ ***UND*** $x_2 = \widetilde{A}_{21}$ ***THEN*** $y_1 = \widetilde{B}_{1*}$

Abbildung–2 ist die Umkehrabbildung von Abbildung–1. Der daraus gewonnene Fuzzy–Controller verhält sich nun so, wie das MFOS–M–Netz, aus dem er hervorgegangen ist. D.h. es werden exakt dieselben Berechnungen durchgeführt wie im MFOS–M–Netz. Somit stellt das MFOS–M–System eine Methode zur Verfügung, einen potentiell beliebigen Fuzzy–Controller auf ein funktional äquivalentes neuronales Netz zu übertragen, dieses zu optimieren, und anschließend wieder auf einen funktional äquivalenten (optimierten) Fuzzy–Controller zu übertragen. Die Korrektheit von Abbildung–2 wurde in [Tenhagen, 2000] bewiesen. D.h. zu gleichen Eingaben berechnen ein MFOS–Netz

und der zugehörige Fuzzy–Controller dieselben Ausgaben. Insgesamt ergibt sich durch die Abbildungen und die Pre–Tuning– und Fine–Tuning–Verfahren das in Abb. 6.9 gezeigte Diagramm:

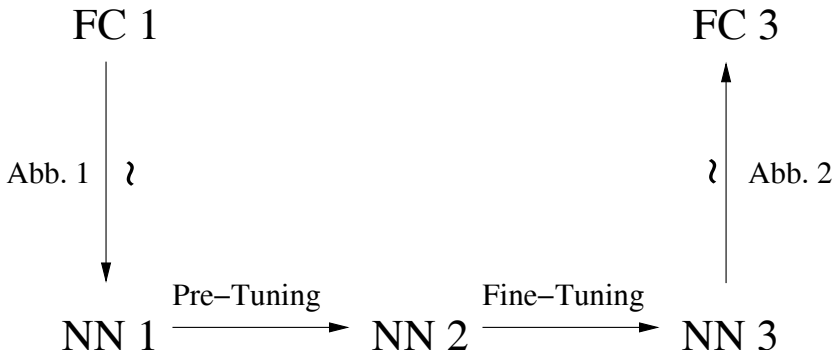

Abbildung 6.9: Diagramm der MFOS–M–Abbildungen

In einem typischen Anwendungsfall wird ein gegebener Fuzzy–Controller FC 1 auf ein MFOS–M–Netz NN 1 übertragen, um in zwei Phasen optimiert zu werden. Das Pre–Tuning ergibt das bereits deutlich verbesserte MFOS–M–Netz NN 2. Dieses wird durch die Anwendung der Fine–Tuning–Verfahren weiter optimiert, wodurch das endgültige MFOS–M–Netz NN 3 entsteht. NN 3 wird anschließend zurücktransformiert in einen optimierten Fuzzy–Controller FC 3 (s. Abb. 6.9). Letztlich erhält man somit einen verbesserten Fuzzy–Controller, der ohne das verwendete MFOS–M–Netz eingesetzt werden kann. Allerdings ist diese Vorgehensweise nicht mehr möglich, wenn freie Gewichte verwendet werden. Freie Gewichte repräsentieren keine gewöhnlichen Partitionierungen mehr, ebenso sind in diesem Fall die Regeln anders strukturiert. Mathematisch ausgedrückt: ein MFOS–M–Netz mit freien Gewichten ist nicht im Bild von Abbildung–1 enthalten. Somit liegt es nicht im Definitions–Bereich von Abbildung–2. Daher läßt sich Abbildung–2 nicht mehr anwenden.

Wie wir in Abschnitt 6.4.4 gesehen haben, ist es jedoch bei Anwendung der Maximum–Methode für die erfolgreiche Optimierung mittels des Gradientenabstiegsverfahrens unerläßlich, freie Gewichte zu verwenden. Hier entsteht also ein Konflikt:

- mit freien Gewichten ist erfolgreiches Training möglich, aber keine Extraktion der Regeln und Partitions–Mengen
- mit gekoppelten Gewichten ist eine Extraktion der Regeln und Partitions–Mengen möglich, aber kein erfolgreiches Training

Da der Mißerfolg des Trainings mit gekoppelten Gewichten auf grundsätzlichen Eigenschaften der Maximum–Methode beruht, ist eine Lösung des Konflikts nur in der Regelextraktion zu finden. Hier bieten sich prinzipiell zwei Möglichkeiten an:

1. Modifikation von Abbildung–2, so daß ein MFOS–M–Netz mit freien Gewichten eingesetzt werden kann

2. Modifikation des MFOS–M–Netzes, so daß es im Definitions–Bereich von Abbildung–2 liegt

Wir entscheiden uns hier für die zweite Möglichkeit. Es wird zunächst ein MFOS–M–Netz mit freien Gewichten beim Training eingesetzt. Nach dem Training werden die ursprünglich gleichen Gewichte (d.h. die gekoppelt sein sollten) wieder zusammengefaßt. Die Regeln verwenden danach wieder gemeinsame Gewichte:

- markiere jeweils alle gekoppelten Gewichte
- trainiere mit freien Gewichten
- fasse jeweils alle vorher markierten Gewichte zu einem gemeinsamen Wert zusammen
  - Modalwert ist der Durchschnitt der zusammengehörenden Modalwerte
  - Weite ist der Durchschnitt der zusammengehörenden Weiten
- Regeln, die sich auf markierte Gewichte beziehen, verwenden nun das zugehörige gemeinsame Gewicht

Nach dieser Modifikation liegt wieder ein MFOS–M–Netz vor, das im Definitionsbereich von Abbildung–2 liegt. Die Extraktion von Regeln und Partitionierungen ist also wieder möglich. Tests zeigen, daß nach dem Zusammenfassen der markierten Gewichte praktisch keine Verschlechterung der Ergebnisse eintritt. Die freien Gewichte sind also nur für das eigentliche Training notwendig. Der Grund hierfür liegt darin, daß sich die Änderungen bei gekoppelten Gewichten gegenseitig beeinflußen und einander Endgegenwirken. Wird eine Partitions–Fuzzy–Menge in einem Schritt nach links verschoben, ist es möglich, daß sie sofort danach nach rechts verschoben wird, etc. Nach der separaten Einstellung der Gewichte sind auch die zusammengefaßten Fuzzy–Mengen korrekt und führen zu optimierten Ausgaben.

Die beschriebene Vorgehensweise ist also geeignet, den Konflikt zu lösen. Somit ist nun in jedem Fall ein erfolgreiches Training und eine anschließende Extraktion der Regeln und Partitionierungen möglich. Das folgende Beispiel macht die weitere Verbesserung eines mit dem Pre–Tuning–Verfahren trainierten MFOS–M–Netzes durch das anschließende Fine–Tuning deutlich:

Beispiel 6.16 *Der hier vorgestellte Fuzzy–Controller dient der Regelung eines Heizgerätes. Die Eingaben (Temperatur) und korrekten Ausgaben (Heizleistung) sind:*

$(15, 8)$, $(18, 4)$, $(21, 2)$.

Die Werte für die korrekte Ausgabe sind nicht gleichmäßig verteilt. Falls die korrekten Partitions–Fuzzy–Mengen unbekannt sind, werden vom Anwender häufig gleichmäßig verteilte Mengen gewählt. Deshalb werden hier die folgenden Gauß–Mengen eingesetzt: (Modalwert, Weite):

Eingabe–Partitions–Mengen:

$$\begin{aligned} \texttt{sehr kalt} &\mathrel{\widehat{=}} \widetilde{A}_1 = (15, 2) \\ \texttt{kalt} &\mathrel{\widehat{=}} \widetilde{A}_2 = (18, 2) \\ \texttt{warm} &\mathrel{\widehat{=}} \widetilde{A}_3 = (21, 2) \end{aligned}$$

Ausgabe–Partitions–Mengen:

$$\begin{aligned} \texttt{schwach} &\mathrel{\widehat{=}} \widetilde{B}_1 = (2, 2) \\ \texttt{mittel} &\mathrel{\widehat{=}} \widetilde{B}_2 = (5, 2) \\ \texttt{hoch} &\mathrel{\widehat{=}} \widetilde{B}_3 = (8, 2) \end{aligned}$$

Die korrekte Regelbasis ist:

$$\begin{aligned} &\texttt{IF } x = \texttt{sehr kalt THEN } y = \texttt{hoch} \\ &\texttt{IF } x = \texttt{kalt} \quad\;\; \texttt{THEN } y = \texttt{mittel} \\ &\texttt{IF } x = \texttt{warm} \quad\;\; \texttt{THEN } y = \texttt{schwach} \end{aligned}$$

Für den Pre–Tuning–Test wird eine fehlerhafte Regelbasis ausgewählt:

$$\begin{aligned} &\texttt{IF } x = \texttt{sehr kalt THEN } y = \texttt{schwach} \\ &\texttt{IF } x = \texttt{kalt} \quad\;\; \texttt{THEN } y = \texttt{mittel} \\ &\texttt{IF } x = \texttt{warm} \quad\;\; \texttt{THEN } y = \texttt{hoch} \end{aligned}$$

Natürlich ergeben sich dadurch sehr schlechte berechnete Ausgaben: (korrekte Ausgabe, berechnete Ausgabe):

$(8, 2.28669), (4, 5.0), (2, 7.71331)$.

Das Verfahren zum Korrigieren von Regeln verbessert die fehlerhaften Regeln und erzeugt genau die richtige Regelbasis. Damit ergeben sich folgende berechnete Ausgabe–Werte:

$(8, 7.71331)$, $(4, 5)$, $(2, 2.28669)$.

Offensichtlich sind diese Ergebnisse wesentlich besser als die vorherigen und je nach Anforderungen an die Genauigkeit des Systems brauchbar. Durch Anwendung der Fine–Tuning–Verfahren lassen sich jedoch weitere Verbesserungen erreichen. Insbesondere ist die ungünstig gewählte Fuzzy–Menge `mittel` $\widehat{=}\widetilde{B}_2 = (5, 2)$ *verantwortlich für den Fehler 1 bei* $(4, 5)$*. Nach Anwendung des Gradientenabstiegsverfahrens sind die optimierten Fuzzy–Mengen:*

$$\begin{aligned}
\texttt{sehr kalt} &\widehat{=} \widetilde{A}_1 = (14.7752, 1.60203)\\
\texttt{kalt} &\widehat{=} \widetilde{A}_2 = (18.0703, 1.78574)\\
\texttt{warm} &\widehat{=} \widetilde{A}_3 = (20.75, 2.31935)
\end{aligned}$$

$$\begin{aligned}
\texttt{schwach} &\widehat{=} \widetilde{B}_1 = (1.84041, 2.09948)\\
\texttt{mittel} &\widehat{=} \widetilde{B}_2 = (4.5026, 1.94062)\\
\texttt{hoch} &\widehat{=} \widetilde{B}_3 = (8.19568, 1.95888)
\end{aligned}$$

Mit diesem optimierten System bekommen wir sehr gute Ergebnisse:

$(8, 7.99759), (4, 4.00208), (2, 1.99909)$.

Durch das Fine–Tuning lassen sich also die schon guten Ergebnisse der Pre–Tuning–Verfahren noch einmal deutlich verbessern. Dennoch können weiterhin die bei jedem Gradientenabstiegsverfahren möglichen Probleme wie z.B. Stillstand auf flachen Plateaus und Oszillieren in Schluchten auftreten. Um dies so weit wie möglich zu vermeiden, sollen als nächstes die vom Backpropagation bekannten Verfahren Momentum–Version und $\bar{\delta} - \delta$–Regel an MFOS–M–Netze angepaßt werden (vgl. auch Kapitel 2).

6.5 Modifikationen der Fine–Tuning–Verfahren

Um die bekannten Probleme beim Gradientenabstieg zu vermeiden, haben sich in der Praxis insbesondere die Momentum–Version und die $\bar{\delta} - \delta$–Regel bewährt (s. Kapitel 2). Es ist zu erwarten, daß eine Übertragung dieser Verfahren auf MFOS–M–Netze ebenfalls zu einer Verbesserung des Trainingserfolges führen wird. Daher sollen nun die Momentum–Version, die $\bar{\delta} - \delta$–Regel und eine Kombination von beiden an MFOS–M–Netze angepaßt werden. Aufgrund der in Abschnitt 6.4.5 beschriebenen Untersuchungen wird von der Version des Netzes mit freien Gewichten und anschließender Zusammenfassung ausgegangen.

6.5.1 Vorstellung der Modifikationen

Momentum–Version

Die Integration der Momentum–Version in MFOS–M–Netze ist problemlos möglich. Beim gewöhnlichen Gradientenabstieg mit MFOS–M–Netzen werden die Modalwerte m_{ik} und die Weiten w_{ik} der Partitions–Fuzzy–Mengen jeweils ein Stück in die Gegenrichtung des Gradienten verschoben. Nun wird für die Verschiebung zum Zeitpunkt $t+1$ noch die Verschiebung zum Zeitpunkt t mit berücksichtigt. Somit geschieht die Adaption der Gewichte nach folgenden Formeln:

$$\Delta m_{ik}(t+1) = -(1-\alpha)\cdot\eta\cdot\frac{\partial F}{\partial m_{ik}}(t+1) + \alpha\cdot\Delta m_{ik}(t)$$

und

$$\Delta w_{ik}(t+1) = -(1-\alpha)\cdot\eta\cdot\frac{\partial F}{\partial w_{ik}}(t+1) + \alpha\cdot\Delta w_{ik}(t)$$

mit η der üblichen Lernrate und $\alpha \in [0,1[$ dem *Momentum–Term*.

$-\eta\cdot\frac{\partial F}{\partial m_{ik}}$ bzw. $-\eta\cdot\frac{\partial F}{\partial w_{ik}}$ sind exakt die Änderungen von m_{ik} bzw. w_{ik} beim gewöhnlichen Gradientenabstieg. Deshalb werden bei dieser Version zunächst alle Berechnungen analog wie beim Gradientenabstieg durchgeführt, um diese Werte zu bestimmen. Anschließend erfolgt die Kombination mit dem Momentum–Term.

Der Momentum–Term α bestimmt den Anteil des aktuellen Gradienten und der Änderung im vorherigen Schritt für die Anpassung der Gewichte im aktuellen Schritt. Haben der Gradient und die vorherige Änderung das gleiche Vorzeichen, so ist der Betrag der Summe entsprechend groß. Üblicherweise ist dies die Situation auf flachen Plateaus der Fehleroberfläche. Daher wird dort die Änderung der Gewichte beschleunigt. Haben der Gradient und die vorherige Änderung verschiedene Vorzeichen, ist der Betrag der Summe meistens kleiner. Üblicherweise ist dies die Situation in zerklüfteten Gebieten, in denen die Gewichte häufig oszillieren. Daher werden die Gewichte hier nur gering verändert. Somit ist zu erwarten, daß die Momentum Version eine bessere und schnellere Anpassung der Gewichte ermöglicht.

Die Wahl von α hat entscheidenden Einfluß auf den Effekt der Momentum–Version. Ein großer Wert (z.B. $\alpha = 0.9$) bewirkt ein schnelles Überwinden von flachen Plateaus. Allerdings erhöht sich dabei das Risiko, daß die Summe in zerklüfteten Gebieten ein anderes Vorzeichen hat als der Gradient selber. Dies tritt ein, wenn der Betrag von $\alpha\cdot\Delta m_{ik}(t)$ bzw. $\alpha\cdot\Delta w_{ik}(t)$ zu groß wird. Ein kleinerer Wert von α kann dies weitgehend verhindern. Der optimale Wert von α hängt vom gewählten Netz und dem trainierten

Problem ab und muß experimentell bestimmt werden. Noch besser wäre daher eine automatische Anpassung von α während des Trainings. Eine Möglichkeit dazu wird im Abschnitt 6.5.1 "Kombination von Momentum–Version und $\bar{\delta} - \delta$–Regel" vorgestellt.

$\bar{\delta} - \delta$–Regel

Bei der $\bar{\delta} - \delta$–Regel bekommt *jedes* Gewicht m_{ik} und w_{ik} eine eigene, individuelle Lernrate $\eta_{m_{ik}}$ bzw. $\eta_{w_{ik}}$. Auf diese Weise wird die in jeder Dimension unterschiedliche Krümmung der Fehleroberfläche berücksichtigt. Zudem wird jede Lernrate während des Trainings verändert, um sie der aktuellen Situation anzupassen. Bleibt das Vorzeichen der partiellen Ableitung $\frac{\partial F}{\partial m_{ik}}$ bzw. $\frac{\partial F}{\partial w_{ik}}$ in aufeinanderfolgenden Schritten gleich, wird $\eta_{m_{ik}}$ bzw. $\eta_{w_{ik}}$ linear erhöht. Sobald das Vorzeichen der partiellen Ableitung wechselt, wird die jeweilige Lernrate um einen bestimmten Anteil ihres Wertes verringert. Die Bestimmung von $\eta_{m_{ik}}$ und $\eta_{w_{ik}}$ erfolgt gemäß:

$$\eta_{m_{ik}}(t+1) = \eta_{m_{ik}}(t) + \Delta\eta_{m_{ik}}(t)$$

$$\eta_{w_{ik}}(t+1) = \eta_{w_{ik}}(t) + \Delta\eta_{w_{ik}}(t)$$

mit

$$\Delta\eta_{m_{ik}}(t) = \begin{cases} \kappa, & \text{falls } \bar{\delta}_{m_{ik}}(t-1) \cdot \delta_{m_{ik}}(t) > 0 \\ -\phi \cdot \eta_{m_{ik}}(t), & \text{falls } \bar{\delta}_{m_{ik}}(t-1) \cdot \delta_{m_{ik}}(t) < 0 \\ 0, & \text{sonst} \end{cases}$$

und

$$\Delta\eta_{w_{ik}}(t) = \begin{cases} \kappa, & \text{falls } \bar{\delta}_{w_{ik}}(t-1) \cdot \delta_{w_{ik}}(t) > 0 \\ -\phi \cdot \eta_{w_{ik}}(t), & \text{falls } \bar{\delta}_{w_{ik}}(t-1) \cdot \delta_{w_{ik}}(t) < 0 \\ 0, & \text{sonst} \end{cases}$$

sowie

$$\delta_{m_{ik}}(t) = \frac{\partial F}{\partial m_{ik}}(t), \delta_{w_{ik}}(t) = \frac{\partial F}{\partial w_{ik}}(t)$$

und

$$\bar{\delta}_{m_{ik}}(t) = (1-\theta) \cdot \delta_{m_{ik}}(t) + \theta \cdot \bar{\delta}_{m_{ik}} k(t-1),$$

$$\bar{\delta}_{w_{ik}}(t) = (1-\theta) \cdot \delta_{w_{ik}}(t) + \theta \cdot \bar{\delta}_{w_{ik}} k(t-1)$$

κ, θ, ϕ sind Konstanten: $\kappa > 0$; $\theta, \phi \in [0,1]$.

Die $\bar{\delta} - \delta$–Regel basiert auf ähnlichen Heuristiken wie die Momentum–Version, realisiert diese jedoch völlig anders. Anstatt die Änderungen im Schritt vorher direkt zu berücksichtigen, werden die Lernraten während des Trainings fortlaufend gezielt angepaßt. Bleibt das Vorzeichen der partiellen Ableitung mehrere Schritte hintereinander gleich, ist anzunehmen, daß die Bewegung des Gewichtsvektors auf einem flachen Plateau der Fehleroberfläche erfolgt. Daher ist eine Beschleunigung der Bewegung erwünscht. Wechselt das Vorzeichen der partiellen Ableitung, ist anzunehmen, daß die Bewegung des Gewichtsvektors in einem zerklüfteten Gebiet erfolgt. Daher ist eine Verlangsamung der Bewegung erwünscht. Beides bewerkstelligt die $\bar{\delta} - \delta$–Regel.

Aufgrund der Verwendung von $\bar{\delta}$ zur Bestimmung der Änderung werden die früheren partiellen Ableitungen zu einem bestimmten Anteil mit berücksichtigt. Dadurch gelangen Informationen über die Entwicklung der Fehlerfunktion in die Bestimmung der Lernraten, so daß die Eigenschaften der Fehleroberfläche zuverlässiger bestimmt werden können. Die Erhöhung der Lernraten ist linear, daher erfolgt auf flachen Plateaus eine stetige Beschleunigung der Gewichtsänderungen, ohne plötzliches zu starkes Ansteigen. Die Verringerung der Lernraten ist hingegen exponentiell. Dadurch werden die Gewichtsänderungen in zerklüfteten Gebieten sofort wieder klein, unabhängig davon, wie stark sie zuvor erhöht wurden.

Die Wahl von κ hat entscheidenden Einfluß auf den Effekt der $\bar{\delta} - \delta$-Regel. Ein zu kleiner Wert läßt die Lernrate nur langsam ansteigen, so daß die Bewegung auf flachen Plateaus nicht effektiv genug beschleunigt wird. Ein zu großer Wert von κ hingegen läßt die Lernraten leicht so groß werden, daß eine genaue Anpassung der Gewichte verhindert wird. Der optimale Wert von κ hängt vom gewählten Netz sowie dem trainierten Problem ab und muß für jede Anwendung erneut experimentell bestimmt werden. Noch besser wäre daher eine automatische Anpassung von κ während des Trainings. Eine Möglichkeit dazu wird im folgenden Abschnitt "Kombination von Momentum–Version und $\bar{\delta} - \delta$–Regel" vorgestellt.

Kombination von Momentum–Version und $\bar{\delta} - \delta$–Regel

Sowohl die Momentum–Version als auch die $\bar{\delta} - \delta$–Regel benötigen auf flachen Plateaus einen möglichst großen Wert von α bzw. κ. Nur so ist eine effektive Beschleunigung der Gewichtsänderungen zu erreichen. Jedoch wird bei beiden Methoden durch hohe Werte von α bzw. κ gleichermaßen die Gefahr erhöht, in zerklüfteten Gebieten zu versagen. Bei zu kleinen Werten dieser Parameter ist hingegen eine effektive Beschleunigung des Trainings nicht mehr zu erreichen. Daher scheint eine Anpassung dieser Parameter–Werte während des Trainings sinnvoll. Ebenso bietet sich eine Kombination von der Momentum–Version und der $\bar{\delta} - \delta$–Regel an. Hohe Lernraten und der Momentum–Term können die Bewegung auf flachen Plateaus ausreichend beschleunigen, ohne die einzelnen Parameter zu groß wählen zu müssen. In zerklüfteten Gebieten ergibt das Verringern der Lernrate bei gleichzeitiger Anwendung des Momentum–Terms eine doppelte Absicherung gegen zu starke Änderung des Gewichtsvektors.

Für ein erfolgreiches Zusammenwirken beider Verfahren ist allerdings eine sorgfältige Einstellung der Parameter α und κ erforderlich. Je größer α ist, desto stärker ist der Effekt der Momentum–Version. Gleichzeitig läßt jedoch der Einfluß der Lernraten nach, so daß deren aufwendige Steuerung redundant wird. Um eine optimale Einstellung der Parameter und eine effektive Kombination zu gewährleisten, sollen nun α und κ während des Trainings laufend neu eingestellt werden. Dies wird im Folgenden durch einen in das MFOS–M–System integrierten Sugeno–Controller (s. Abschnitt 3.12) durchgeführt.

Aufgrund der beschriebenen Einflüsse von α und κ soll die Anpassung dieser Parameter folgende Heuristiken berücksichtigen:

- je länger die Adaption der Gewichte auf einem flachen Plateau erfolgt, desto größer müssen α und κ werden
- je stärker die lokale Fehleroberfläche gekrümmt ist, desto kleiner müssen α und κ werden

Bevor eine Anpassung der Parameter möglich ist, muß demnach zunächst einmal die Dauer der Adaption auf einem flachen Plateau bzw. die Stärke der lokalen Krümmung bewertet werden. Die $\bar{\delta} - \delta$–Regel stellt dazu eine geeignete Methode zur Verfügung:

- je öfter der Wert von η nacheinander erhöht wird, desto länger erfolgt die Adaption der Gewichte auf einem flachen Plateau
- je öfter der Wert von η nacheinander verringert wird, desto stärker ist die lokale Fehleroberfläche gekrümmt

Zur Bewertung dieser Eigenschaften wird nun für jedes Gewicht m_{ik} bzw. w_{ik} eine zusätzliche Variable $c_{m_{ik}}$ bzw. $c_{w_{ik}}$ definiert. Während des Trainings werden diese Variablen bei der Anpassung der Lernraten mit der $\bar{\delta} - \delta$–Regel gemäß folgender Formeln verändert:

$$\Delta c_{m_{ik}} = \begin{cases} c_1, & \text{falls } \eta_{m_{ik}} \text{ erhöht wird} \\ -c_2, & \text{falls } \eta_{m_{ik}} \text{ verringert wird} \end{cases}$$

$$\Delta c_{w_{ik}} = \begin{cases} c_1, & \text{falls } \eta_{w_{ik}} \text{ erhöht wird} \\ -c_2, & \text{falls } \eta_{w_{ik}} \text{ verringert wird} \end{cases}$$

mit $c_1, c_2 \in]0, 5]$ Konstanten, wobei $c_1 < c_2$ gilt.

Damit ergeben $c_{m_{ik}}$ und $c_{w_{ik}}$ die gewünschte Bewertung der durchgeführten Gewichtsänderungen. Je höher diese Werte sind, desto länger erfolgt die Adaption auf einem flachen Plateau. Je kleiner diese Werte sind, desto zerklüfteter ist die lokale Fehleroberfläche. Die Erhöhung um c_1 bewirkt eine stetige Vergrößerung auf flachen Plateaus.

Da $c_2 > c_1$ gilt, wird der Wert von $c_{m_{ik}}$ bzw. $c_{w_{ik}}$ in zerklüfteten Gebieten sofort stark verringert. Die Werte von $c_{m_{ik}}$ und $c_{w_{ik}}$ sollen die Eingaben des Sugeno–Controllers werden. Um alle möglichen Werte mit einer geeigneten Partitionierung verarbeiten zu können, wird zusätzlich sichergestellt, daß $c_{m_{ik}}$ und $c_{w_{ik}}$ beschränkt sind. Aus diesem Grund genügt auch eine lineare Verringerung um c_2, statt der bei der $\bar{\delta} - \delta$–Regel verwendeten exponentiellen Verringerung.

Zur Verarbeitung der Eingaben werden vier Fuzzy–Mengen für die linguistische Variable *Krümmung* verwendet: VeryLow, Low, NotSure und High. Bei Wahl von $c_1 = 1.0$ und $c_2 = 2.0$ sind folgende Zugehörigkeitsfunktionen geeignet (Trapezmengen):

$\mu_H = (-1, -1, 1, 2)$
$\mu_N = (-1, 2, 2, 4)$
$\mu_L = (2, 5, 5, 8)$
$\mu_V = (6, 8, 10, 10)$

Da nun α und κ während des Trainings individuell angepaßt werden sollen, bekommt hier jedes Gewicht m_{ik} und w_{ik} eigene Parameter $\alpha_{m_{ik}}$ und $\alpha_{w_{ik}}$ bzw. $\kappa_{m_{ik}}$ und $\kappa_{w_{ik}}$. Die Konklusionen der Regeln sind reellwertig, da ein Sugeno–Controller eingesetzt wird. Folgende Regelbasis wird verwendet:

R_{01} : IF $c_{m_{ik}} =$ VeryLow THEN $\alpha_{m_{ik}} = 0.9$
R_{02} : IF $c_{m_{ik}} =$ Low THEN $\alpha_{m_{ik}} = 0.7$
R_{03} : IF $c_{m_{ik}} =$ Notsure THEN $\alpha_{m_{ik}} = 0.3$
R_{04} : IF $c_{m_{ik}} =$ High THEN $\alpha_{m_{ik}} = 0.01$
R_{05} : IF $c_{m_{ik}} =$ VeryLow THEN $\kappa_{m_{ik}} = 100 \cdot \kappa_0$
R_{06} : IF $c_{m_{ik}} =$ Low THEN $\kappa_{m_{ik}} = 10 \cdot \kappa_0$
R_{07} : IF $c_{m_{ik}} =$ NotSure THEN $\kappa_{m_{ik}} = 1 \cdot \kappa_0$
R_{08} : IF $c_{m_{ik}} =$ High THEN $\kappa_{m_{ik}} = 0.1 \cdot \kappa_0$
R_{09} : IF $c_{w_{ik}} =$ VeryLow THEN $\alpha_{w_{ik}} = 0.9$
R_{10} : IF $c_{w_{ik}} =$ Low THEN $\alpha_{w_{ik}} = 0.7$
R_{11} : IF $c_{w_{ik}} =$ Notsure THEN $\alpha_{w_{ik}} = 0.3$
R_{12} : IF $c_{w_{ik}} =$ High THEN $\alpha_{w_{ik}} = 0.01$
R_{13} : IF $c_{w_{ik}} =$ VeryLow THEN $\kappa_{w_{ik}} = 100 \cdot \kappa_0$
R_{14} : IF $c_{w_{ik}} =$ Low THEN $\kappa_{w_{ik}} = 10 \cdot \kappa_0$
R_{15} : IF $c_{w_{ik}} =$ NotSure THEN $\kappa_{w_{ik}} = 1 \cdot \kappa_0$
R_{16} : IF $c_{w_{ik}} =$ High THEN $\kappa_{w_{ik}} = 0.1 \cdot \kappa_0$

Mit einem Startwert κ_0.

Die Berechnung der Ausgabe des Sugeno–Controllers geschieht gemäß:

$$\alpha_{m_{ik}} = \frac{\mu_V(c_{m_{ik}}) \cdot 0.9 + \mu_L(c_{m_{ik}}) \cdot 0.7 + \mu_N(c_{m_{ik}}) \cdot 0.3 + \mu_H(c_{m_{ik}}) \cdot 0.01}{\mu_V(c_{m_{ik}}) + \mu_L(c_{m_{ik}}) + \mu_N(c_{w_{ik}}) + \mu_H(c_{m_{ik}})}$$

$$\kappa_{m_{ik}} = \frac{\Big(\mu_V(c_{m_{ik}}) \cdot 100 + \mu_L(c_{m_{ik}}) \cdot 10 + \mu_N(c_{m_{ik}}) \cdot 1 + \mu_H(c_{m_{ik}}) \cdot 0.1\Big) \cdot \kappa_0}{\mu_V(c_{m_{ik}}) + \mu_L(c_{m_{ik}}) + \mu_N(c_{m_{ik}}) + \mu_H(c_{m_{ik}})}$$

$$\alpha_{w_{ik}} = \frac{\mu_V(c_{w_{ik}}) \cdot 0.9 + \mu_L(c_{w_{ik}}) \cdot 0.7 + \mu_N(c_{w_{ik}}) \cdot 0.3 + \mu_H(c_{w_{ik}}) \cdot 0.01}{\mu_V(c_{w_{ik}}) + \mu_L(c_{w_{ik}}) + \mu_N(c_{w_{ik}}) + \mu_H(c_{w_{ik}})}$$

$$\kappa_{w_{ik}} = \frac{\Big(\mu_V(c_{w_{ik}}) \cdot 100 + \mu_L(c_{w_{ik}}) \cdot 10 + \mu_N(c_{w_{ik}}) \cdot 1 + \mu_H(c_{w_{ik}}) \cdot 0.1\Big) \cdot \kappa_0}{\mu_V(c_{w_{ik}}) + \mu_L(c_{w_{ik}}) + \mu_N(c_{w_{ik}}) + \mu_H(c_{w_{ik}})}$$

Mit Hilfe dieses Sugeno–Controllers werden die Werte von $\alpha_{m_{ik}}$ und $\alpha_{w_{ik}}$ bzw. $\kappa_{m_{ik}}$ und $\kappa_{w_{ik}}$ während des Trainings ständig angepaßt. Die Adaption der Gewichte erfolgt gemäß der Momentum–Version. Dabei werden nun individuelle Lernraten verwendet, die gemäß $\bar{\delta} - \delta$–Regel variieren.

6.5.2 Bewertung der Modifikationen

Aufgrund der durchgeführten Untersuchungen läßt sich folgern, daß alle Methoden gute Ergebnisse liefern. Der Standard–Gradientenabstieg funktioniert solide, ohne besonders schnell die besten Werte zu erreichen. Die $\bar{\delta} - \delta$–Regel und die Momentum–Version ergeben häufig die besten Einzelergebnisse, sowohl nach wenigen als auch nach mehreren Trainingsdurchläufen. Die Kombination liefert nur selten die besten Einzelergebnisse, hat aber häufiger als alle anderen Verfahren die schlechtesten Einzelwerte.

Es läßt sich also die Tendenz erkennen, daß bei einer gewissen Anzahl von Einzelwerten die $\bar{\delta} - \delta$–Regel und die Momentum–Version etwas bessere Ergebnisse liefern bzw. etwas schneller zu guten Resultaten führen. Dieser Vorteil wird durch den Nachteil erkauft, daß bestimmte Einzelwerte etwas schlechter sind, jedoch immer noch vollkommen akzeptabel. Daher bieten sich diese beiden Verfahren als lohnende Alternative zu dem Standard–Gradientenabstieg an.

Die Kombination der beiden Verfahren ergibt nicht den erhofften Vorteil. Es ergeben sich nur selten die besten Einzelwerte, dagegen gibt es überdurchschnittlich häufig etwas schlechtere Resultate. Zwar sind grundsätzlich alle Einzelwerte akzeptabel, jedoch

lohnt sich der extrem hohe Rechenaufwand bei der Fuzzy–Steuerung auf keinen Fall. Ein Grund hierfür ist die Konfiguration eines MFOS–M–Netzes. Die Struktur und die Anzahl der verborgenen Neuronen werden nicht wie beim Multilayer–Perzeptron aufgrund von Erfahrungswerten bestimmt, sondern mit Hilfe der Pre–Tuning–Verfahren optimal eingestellt.

Außerdem werden die Fuzzy–Gewichte nicht zufällig initialisiert, sondern liegen zu Beginn der Anwendung der Fine–Tuning–Methoden "ungefähr" im korrekten Bereich. D.h. das Gradientenabstiegsverfahren wird für ein vortrainiertes Netz mit optimaler Struktur und bereits voreingestellten Gewichten, die im richtigen Bereich liegen, eingesetzt. In dieser Situation ist die Fehleroberfläche nicht so stark zerklüftet, wie es bei einer großen Anzahl von Gewichten mit zufälliger Initialisierung für ein Multilayer–Perzeptron zu erwarten ist. In so einem Fall kann die Kombination der Verfahren mit Hilfe der Fuzzy–Steuerung echte Vorteile bringen. Beim MFOS–M–Netz kommen aufgrund der Voreinstellung des Netzes und der Gewichte diese Vorteile nicht zum tragen. Daher ist diese Kombination hier nicht sinnvoll. Weitere Tests mit verschiedenen Einstellungen der Partitions–Mengen des Sugeno–Controllers bestätigen dies.

6.6 Einsatz des MFOS–M–Sytems

Das MFOS–M–System stellt unterschiedliche Methoden zur Verfügung, die Regelbasis und die Partitionierungen eines Fuzzy–Controllers zu optimieren. Grundsätzlich hat der Anwender die Möglichkeit, einzelne dieser Verfahren auszuwählen sowie deren Reihenfolge zu bestimmen. In diesem Abschnitt soll nun untersucht werden, welche Verfahren in einer gegebenen Situation sinnvoll sind und welche Reihenfolge die besten Ergebnisse ermöglicht.

Da es bestimmte Wechselwirkungen und Abhängigkeiten zwischen den einzelnen Methoden gibt, ist die Reihenfolge ihrer Anwendung nicht ohne Bedeutung. Es ist offensichtlich, daß bei einer leeren Regelbasis (keine Regeln definiert) eine Korrektur von Regeln sinnlos ist. In diesem Fall *muß* mit der Erzeugung von Regeln begonnen werden. Falls einige Regeln vom Anwender vorgegeben wurden, ist die beste Wahl zur Optimierung nicht so offensichtlich. Sollen erst die vorhandenen Regeln optimiert werden, oder ist es besser, zunächst eventuell fehlende Regeln zu generieren? Bevor diese Fragen beantwortet werden können, müssen zunächst einmal sämtliche Wechselwirkungen zwischen den einzelnen Verfahren genau analysiert werden.

6.6.1 Wechselwirkungen zwischen den einzelnen Verfahren

Es wurden sämtliche vorgestellten Verfahren (s. Abschnitt 6.3 und Abschnitt 6.4) paarweise miteinander verglichen und alle Möglichkeiten einer gegenseitigen Beeinflussung untersucht. Insbesondere war zu klären, ob sich Methoden in ihrer Wirkung gegenseitig unterstützen oder sogar beeinträchtigen können und welche Schritte u.U. überflüssig sind. Falls sich z.B. eine Partitions–Menge durch das Fine–Tuning an die richtige Stelle

verschieben läßt, ist es eventuell nicht nötig, eine zusätzliche Menge an dieser Stelle zu erzeugen. Sollten zwei unterschiedliche Mengen in dem Bereich für verschiedene Ausgaben notwendig sein, ist hingegen eine neue Menge unverzichtbar. Dies muß jeweils im Einzelfall überprüft werden.

- Die Verfahren zur Korrektur von Regeln mit bzw. ohne Berücksichtigung aller Eingabe–Variablen haben keine Wechselwirkung. In beiden Fällen bekommt eine vorhandene Regel eine neue Konklusion.
- Die Verfahren zur Korrektur von Regeln haben keine Wechselwirkung mit dem Verfahren zur Erzeugung neuer Regeln. Eine erzeugte Regel ist korrekt und muß nicht korrigiert werden. Bei der Korrektur einer Regel ändert sich nicht ihre Prämisse und somit nicht der Erfüllungsgrad ihrer Prämisse bei den einzelnen Trainingsbeispielen. Es wird nur eine Regel korrigiert, die bei mindestens einem Beispiel einen hohen Erfüllungsgrad der Prämisse hat. Eine neue Regel wird nur erzeugt, wenn es bei einem Beispiel gar keine Regel mit einem genügend hohen Erfüllungsgrad der Prämisse gibt. Es ist somit nicht möglich, durch die Korrektur einer vorhandenen Regel eine Regel zu erhalten, die sonst erzeugt werden müßte.
- Bei der Korrektur bekommt eine Regel mit einem hohen Erfüllungsgrad eine neue Konklusion. Das bedeutet, daß diese Regel nun die Schnitthöhe für eine *andere* Ausgabe–Partitions–Menge bestimmt. Beim Löschen von Regeln wird überprüft, ob eine Regel immer die maximale bzw. minimale Schnitthöhe bei einer Ausgabe–Partitions–Menge bestimmt. Somit ist es möglich, daß nach der Korrektur die korrigierte Regel die maximale Schnitthöhe der neuen Konklusions–Menge bestimmt. Bei der alten Konklusions–Menge sind nun *andere* Regeln für die maximalen Schnitthöhen zuständig. Daher ist es notwendig, vor dem Löschen von Regeln die Regeln zu korrigieren. Die Strategie, erst unnötige Regeln zu löschen und anschließend die restlichen Regeln zu korrigieren, um unnötige Korrekturen zu vermeiden, funktioniert nicht: es werden nur die Regeln mit maximalem Erfüllungsgrad korrigiert, und das sind genau die Regeln, die *nicht* gelöscht werden.

Beispiel 6.17 *Sei die gleiche Situation wie in Beispiel 6.7 gegeben. Ist die Regel*

IF $x_2 =$ `niedrig` *THEN* $y =$ `klein`

fälschlicherweise als

IF $x_2 =$ `niedrig` *THEN* $y =$ `mittel`

erzeugt worden, wird durch das Verfahren zum Löschen von Regeln keine einzige Regel entfernt, da nun keine Regel immer die maximale bzw. minimale Schnitthöhe einer Ausgabe–Menge bestimmt. Zudem sind die Ergebnisse bei Eingaben im Bereich der

Menge niedrig *fehlerhaft. Nach der Korrektur der Regel sind die berechneten Ausgaben korrekt. Erst jetzt ist es möglich, durch das Verfahren zum Löschen von Regeln die nun überflüssigen Regeln zu identifizieren und wie im Beispiel 6.7 zu entfernen.*

- Das Erzeugen einer neuen Partitions–Fuzzy–Menge erfolgt, falls beim Korrigieren oder Erzeugen von Regeln festgestellt wird, daß keine geeignete Menge vorhanden ist. Es wird also in jedem Fall zuerst ein Verfahren zur Erzeugung oder Korrektur von Regeln angewendet. Eine einmal erzeugte Menge steht selbstverständlich sofort für alle weiteren Regeln zur Verfügung. D.h. falls beim Korrigieren von Regeln eine neue Fuzzy–Menge erzeugt wurde, ist es möglich, sie als Partitions–Menge bei der Erzeugung von Regeln einzusetzen und umgekehrt. Da die Fuzzy–Mengen unter Berücksichtigung der Trainingsdaten generiert werden, ist es gleichwertig, ob eine neue Menge während der Korrektur oder während der Erzeugung von Regeln kreiert wird. Die Reihenfolge ist hier also unerheblich.

- Beim Löschen von Fuzzy–Mengen werden gleichwertige Fuzzy–Mengen zu einer gemeinsamen Menge zusammengefaßt. Anschließend werden alle Regeln zusammengefaßt, die sich nur bezüglich dieser Mengen unterscheiden. Dafür ist es notwendig, daß alle Regeln korrekt sind. Hat eine Regel die falsche Konklusion, so wird sie nicht mit einer korrekten Regel für die selbe Situation zusammengefaßt. Deshalb muß vor dem Einsatz des Verfahrens zum Löschen von Fuzzy–Mengen die Korrektur der Regeln durchgeführt werden.

- Durch die Fine–Tuning–Verfahren werden die Eingabe– und Ausgabe–Partitions–Mengen in ihrer Position (Modalwert) und Weite geändert. Dies hat eine Änderung der Erfüllungsgrade von den Prämissen der Regeln und eine Änderung der berechneten Ausgabe–Werte zur Folge. Beim Korrigieren von Regeln werden Regeln mit einem Erfüllungsgrad der Prämissen über einer Schranke betrachtet. Nach dem Fine–Tuning sind möglicherweise die Erfüllungsgrade anderer Regeln als vorher über bzw. unter dieser Schranke. Korrigiert werden nur Regeln, die einen Fehler über einer Schranke verursachen. Falls durch das Fine–Tuning der Fehler unter diese Schranke sinkt, ist eine Korrektur nicht mehr nötig. Daher sollte vor der Korrektur von Regeln das Fine–Tuning durchgeführt werden.

- Neu erzeugte Regeln bestimmen zusätzliche Schnitthöhen für die Ausgabe–Partitions–Mengen ihrer Konklusion. Eine Regel wird nur erzeugt, falls für ein Trainingsbeispiel keine Regel mit einem genügend hohen Erfüllungsgrad der Prämisse vorhanden ist. Deshalb ist es möglich, daß eine neue Regel nun die maximale Schnitthöhe der Ausgabe–Partitions–Menge ihrer Konklusion bestimmt. Daher müssen vor dem Einsatz des Verfahrens zum Löschen von Regeln zunächst alle notwendigen Regeln erzeugt werden.

- Falls die Partitionierungen zu fein gewählt wurden, werden eventuell für gleichwertige Partitions–Mengen jeweils eigene Regeln erzeugt. Beim anschließenden

Zusammenfassen dieser Mengen würden auch diese Regeln zusammengefaßt werden. D.h. es sind aufgrund der eigentlich unnötigen Partitions–Mengen eigentlich unnötige Regeln erzeugt worden, die später wieder entfernt werden. Dies läst sich vermeiden, indem zuerst alle unnötigen Fuzzy–Mengen gelöscht werden. Unter Berücksichtigung der tatsächlich benötigten Mengen werden auch nur tatsächlich benötigte Regeln erzeugt. Da neue Regeln keinen Einfluß auf die Bewertung von Partitions–Mengen haben, sollten vor dem Erzeugen neuer Regeln gleichwertige Partitions–Mengen zusammengefaßt werden.

- Eine neue Regel wird erzeugt, falls zu einem Trainingsbeispiel keine Prämisse einer Regel einen Erfüllungsgrad über einer Schranke hat. Da durch das Fine–Tuning die Erfüllungsgrade beeinflußt werden, besteht die Möglichkeit, daß durch das Fine–Tuning diese Schranke bei mindestens einer Regel überschritten wird. In diesem Fall wäre es nicht mehr nötig, für das aktuelle Trainingsbeispiel eine neue Regel zu erzeugen. Daher sollte vor der Erzeugung neuer Regeln das Fine–Tuning durchgeführt werden.

- Beim Löschen von Regeln werden jeweils die Schnitthöhen der einzelnen Ausgabe–Partitions–Mengen miteinander verglichen. Daher müssen vor dem Einsatz des Verfahrens zum Löschen von Regeln alle notwendigen Ausgabe–Partitions–Mengen vorhanden sein. Gleiches gilt für die Eingabe–Partitions–Mengen, da sie die Erfüllungsgrade der Prämissen und somit die Schnitthöhe bestimmen. Deshalb sollte das Verfahren zum Löschen von Regeln erst angewendet werden, nachdem alle benötigten Partitions–Mengen erzeugt wurden.

- Falls zwei gleichwertige Fuzzy–Mengen zusammengefaßt werden, werden auch die Regeln zusammengefaßt, die sich nur bei diesen Mengen unterscheiden. Deshalb gibt es nach dem Zusammenfassen weniger Schnitthöhen bei einzelnen Ausgabe–Partitions–Mengen. Daher sollte das Verfahren zum Löschen von Regeln erst angewendet werden, nachdem alle gleichwertigen Partitions–Mengen zusammengefaßt wurden.

- Das Fine–Tuning ändert die Erfüllungsgrade der Prämissen der Regeln und somit die einzelnen Schnitthöhen der Ausgabe–Partitions–Mengen. Dabei ist es möglich, daß in einzelnen Ausgabe–Mengen nach dem Fine–Tuning eine andere Regel die maximale Schnitthöhe bewirkt als vor dem Fine–Tuning. Deshalb sollte vor dem Einsatz des Verfahrens zum Löschen von Regeln das Fine–Tuning durchgeführt werden.

- Das Zusammenfassen gleichwertiger Fuzzy–Mengen und das Erzeugen neuer Fuzzy–Mengen hat keine Wechselwirkung. Neue Fuzzy–Mengen werden nur erzeugt, falls in einem Bereich, in dem gewünschte Ein– bzw. Ausgabe–Werte liegen, gar keine Fuzzy–Menge vorhanden ist. Das Zusammenfassen gleichwertiger Partitions–Mengen ändert an keiner Stelle den überdeckten Bereich. Daher ist die Reihenfolge bei der Anwendung dieser Verfahren gleichwertig.

- Beim Zusammenfassen von zwei benachbarten Eingabe–Partitions–Mengen wird jeweils überprüft, ob für alle Eingaben, die sich nur in diesen Mengen unterscheiden, die Ausgabe in der selben Ausgabe–Partitions–Menge liegt. Durch das Fine–Tuning ist eventuell ein Wechsel der maximalen Zugehörigkeitsgrade bei den Ein– und Ausgabe–Partitions–Mengen möglich. In diesem Fall würden u.U. andere Fuzzy–Mengen als gleichwertig erkannt werden. Daher sollte vor dem Einsatz des Verfahrens zum Löschen von Fuzzy–Mengen das Fine–Tuning durchgeführt werden.

- Neue Partitions–Mengen werden erzeugt, falls in einem Bereich keine geeignete Partitions–Menge liegt. In den Eingabe–Dimensionen ist hier ein genügend hoher Erfüllungsgrad der Prämissen der Regeln und somit der Zugehörigkeitsgrad der Eingabe–Werte zu den Eingabe–Partitions–Mengen das verwendete Kriterium. In den Ausgabe–Dimensionen ist der bei Verwendung einer der vorhandenen Ausgabe–Partitions–Mengen als Konklusion resultierende minimale Fehler der entscheidende Maßstab. Durch das Fine–Tuning ist es in manchen Fällen möglich, den Erfüllungsgrad unter Verwendung vorhandener Mengen über eine Schranke anzuheben. Ebenso ist es in manchen Fällen möglich, durch das Fine–Tuning den Fehler der berechneten Ausgabe–Werte unter eine Schranke abzusenken. In diesen Fällen verschiebt das Fine–Tuning–Verfahren vorhandene Fuzzy–Mengen an die Stellen, wo sie benötigt werden. Falls das möglich ist, ist das Erzeugen neuer Mengen nicht notwendig. Daher sollte vor dem Erzeugen neuer Mengen überprüft werden, ob mit dem Fine–Tuning eine vorhandene Menge an die benötigte Stelle geschoben werden kann. Falls jedoch in einem Bereich mehrere unterschiedliche Fuzzy–Mengen benötigt werden, um verschiedene Situationen zu unterscheiden oder um verschiedene Ausgaben zu ermöglichen, reichen die vorhandenen Mengen unter Umständen nicht aus. In diesem Fall ist ein Erzeugen neuer Fuzzy–Mengen unverzichtbar.

Beispiel 6.18 *Sei die gleiche Situation wie in Beispiel 6.3 gegeben. Sei die Ausgabe–Partitions–Menge* `hoch` $\widehat{=}\widetilde{B}_3 = (6, 8, 10)$ *nun unvorteilhaft als* `hoch` $\widehat{=}\widetilde{B}_3 = (5, 7, 9)$ *definiert worden. Dies führt zu hohen Fehlern bei Eingaben im Bereich der Menge* `sehr kalt`*. Falls nun sofort neue Fuzzy–Mengen erzeugt werden, wird die Ausgabe–Partitions–Menge* `sehr hoch` $\widehat{=}\widetilde{B}_4 = (7.4, 8, 8.6)$ *erzeugt und als neue Konklusion von Regel 1 eingesetzt:*

IF $x =$ `sehr kalt` *THEN* y `sehr hoch`

Damit sind nun alle berechneten Ausgaben optimal. Allerdings ist die zusätzliche Partitions–Menge nicht unbedingt erforderlich. Falls als erstes das Fine–Tuning durchgeführt wird, ändert sich die unvorteilhaft definierte Menge `hoch` $\widehat{=}\widetilde{B}_3 = (5, 7, 9)$ *zu* `hoch` $\widehat{=}\widetilde{B}_3 = (5.99485, 7.99485, 9.99485)$*, mit der ebenfalls alle berechneten Ausgaben optimal sind. Der Vorteil dieser Reihenfolge ist somit, daß bei gleich guten Ergebnissen keine zusätzliche Partitions–Menge erzeugt werden mußte.*

Beispiel 6.19 *Sei auch hier die gleiche Situation wie in Beispiel 6.3 gegeben. Sei nun jedoch die Ausgabe–Partitions–Menge* **hoch** *nicht definiert worden. Die Ausgabe–Partitionierungen sind:*

$\texttt{schwach} \mathrel{\widehat{=}} \widetilde{B}_1 = (0, 2, 4)$
$\texttt{mittel} \mathrel{\widehat{=}} \widetilde{B}_2 = (4.5, 6.5, 8.5)$

Die Regeln sind:

IF $x =$ **sehr kalt** *THEN* $y =$ **mittel**
IF $x =$ **kalt** *THEN* $y =$ **mittel**
IF $x =$ **warm** *THEN* $y =$ **schwach**

In dieser Konstellation führen Eingaben im Bereich der Mengen **sehr kalt** *und* **kalt** *zu hohen Ausgabe–Fehlern. Mit dem Fine–Tuning ist eine optimale Einstellung in dieser Situation nicht möglich. Sowohl Ausgaben, die im Bereich von 5 liegen sollen, als auch Ausgaben, die im Bereich von 8 liegen sollen, schwanken um den Wert 6.5. Der Grund hierfür ist, daß zur Unterscheidung dieser Bereiche eine zusätzliche Partititions–Menge unverzichtbar ist. Durch das Verfahren zur Erzeugung neuer Partitions–Mengen werden geeignete Mengen*

$\texttt{hoch} \mathrel{\widehat{=}} \widetilde{B}_3 = (4.1, 5, 5.9)$
$\texttt{sehr hoch} \mathrel{\widehat{=}} \widetilde{B}_4 = (7.1, 8, 8.9)$

erzeugt und als Konklusionen der Regeln 1 und 2 eingesetzt:

IF $x =$ **sehr kalt** *THEN* $y =$ **sehr hoch**
IF $x =$ **kalt** *THEN* $y =$ **hoch**
IF $x =$ **warm** *THEN* $y =$ **schwach**

Damit werden nun optimale Ergebnisse erzielt (die Menge $\widetilde{B}_2$ *kann gelöscht werden).*

Die Beispiele 6.18 und 6.19 zeigen, daß in manchen Fällen das Fine–Tuning die Erzeugung neuer Partitions–Mengen überflüssig werden läst, in anderen Fällen jedoch das Erzeugen neuer Mengen unverzichtbar ist. Die Anzahl der unbedingt erforderlichen Partitions–Mengen hängt grundsätzlich von der Anzahl der zu unterscheidenden Situationen ab. Hier muß jeweils im Einzelfall überprüft werden, ob allein mit dem Fine–Tuning eine Optimierung möglich ist, oder ob zusätzliche Fuzzy–Mengen erzeugt werden müssen.

6.6.2 Die richtige Reihenfolge der einzelnen Verfahren

Aufgrund der erkannten Wechselwirkungen zwischen den einzelnen Verfahren ergeben sich zunächst – als direkte Folgerung – folgende Empfehlungen für die Reihenfolge:

- zunächst die Regeln korrigieren, anschließend unnötige Regeln löschen
- zunächt die Regeln korrigieren, anschließend unnötige Fuzzy–Mengen löschen
- zunächst das Fine–Tuning durchführen, anschließend die Regeln korrigieren
- zunächst neue Regeln erzeugen, anschließend unnötige Regeln löschen
- zunächst unnötige Fuzzy–Mengen löschen, anschließend neue Regeln erzeugen
- zunächst das Fine–Tuning durchführen, anschließend neue Regeln erzeugen
- zunächst Partitions–Mengen erzeugen, anschließend unnötige Regeln löschen
- zunächst unnötige Fuzzy–Mengen löschen, anschließend unnötige Regeln löschen
- zunächst das Fine–Tuning durchführen, anschließend unnötige Regeln löschen
- zunächst das Fine–Tuning durchführen, anschließend unnötige Fuzzy–Mengen löschen
- zunächst das Fine–Tuning durchführen, anschließend neue Partitions–Mengen erzeugen

Sämtliche Abhängigkeiten der einzelnen Verfahren sind in Abb. 6.10 in der Form

[zunächst] $\longrightarrow$ [anschließend]

dargestellt.

In Abb. 6.10 ist zu erkennen, daß von [Fine–Tuning] ein Pfeil zu jedem anderen Verfahren hinführt. Andererseits gibt es keinen Pfeil, der bei [Fine–Tuning] ankommt. Dies bedeutet, daß das Fine–Tuning in jedem Fall als erstes Verfahren vor allen anderen Verfahren durchgeführt werden sollte.

Umgekehrt verhält es sich bei [Regeln löschen]. Kein Pfeil führt von [Regeln löschen] weg, andererseits führt von jedem Verfahren ein Pfeil zu [Regeln löschen]. Dies bedeutet, daß das Verfahren zum Löschen von Regeln als letztes nach allen anderen Verfahren durchgeführt werden sollte. Damit sind alle Abhängigkeiten für die Verfahren "Fine–Tuning" und "Regeln löschen" geklärt. Somit läst sich das Diagramm aus Abb. 6.10 zum Diagramm in Abb. 6.11 vereinfachen, um die weiteren Abhängigkeiten zu erkennen.

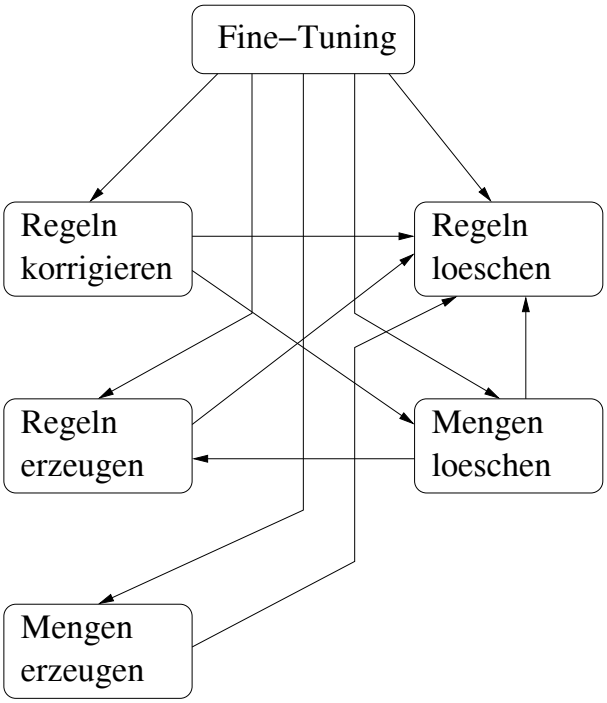

Abbildung 6.10: Abhängigkeiten der einzelnen Verfahren (1)

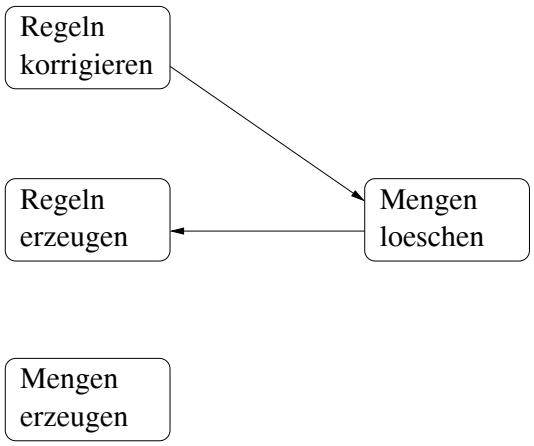

Abbildung 6.11: Abhängigkeiten der einzelnen Verfahren (2)

In Abb. 6.11 ist zu erkennen, daß erst das Verfahren zum Korrigieren von Regeln angewendet werden sollte, anschließend das Verfahren zum Löschen von Fuzzy–Mengen und erst danach das Verfahren zum Erzeugen von neuen Regeln.

Als Fazit dieser Untersuchungen ergibt sich folgende Reihenfolge für den Einsatz der MFOS–M–Verfahren:

1. Fine–Tuning
2. Regeln korrigieren und bei Bedarf neue Partitions–Mengen erzeugen
3. Partitions–Mengen löschen
4. Regeln erzeugen und bei Bedarf neue Partitions–Mengen erzeugen
5. Regeln löschen

Dies ist die Reihenfolge, die sich aus den Abhängigkeiten und Wechselwirkungen zwischen den einzelnen Verfahren ergibt. Beispiel 6.18 hat gezeigt, daß die Anwendung des Fine–Tuning vor dem Pre–Tuning das Erzeugen nicht unbedingt benötigter zusätzlicher Fuzzy–Mengen verhindern kann. Da das Fine–Tuning keine negativen Auswirkungen hat, ist gegen seine Anwendung als erstes Verfahren nichts einzuwenden. In der Praxis muß im Einzelfall entschieden werden, welches die optimale Reihenfolge ist. Falls gar keine Regeln und Partitions–Mengen vordefiniert wurden, muß mit dem Erzeugen von Regeln und Partitions–Mengen begonnen werden. Anschließend erfolgt eine weitere Optimierung mit dem Fine–Tuning. Falls einige Regeln und Partitions–Mengen vordefiniert wurden, sollte zunächst eine Optimierung der vorhandenen Regeln und Partititions–Mengen versucht werden. Nur wenn dies nicht zu hinreichend guten Ergebnissen führt, fehlen offensichtlich benötigte Regeln bzw. Fuzzy–Mengen und müssen erzeugt werden. Anschließend kann in jedem Fall mit dem Fine–Tuning eine weitere Optimierung erfolgen (vgl. Beispiel 6.16).

Das Fine–Tuning erfüllt also zwei Aufgaben. Wird es als erstes Verfahren eingesetzt, schiebt es ungünstig positionierte Partitions–Mengen an die richtige Stelle, so daß keine eigentlich überflüssigen Fuzzy–Mengen generiert werden. Nachdem mit Hilfe der übrigen Verfahren alle Bestandteile des Fuzzy–Controllers korrekt eingestellt wurden, sind die berechneten Ausgaben des Fuzzy–Controllers hinreichend gut. Anschließend ist es möglich, mit den Fine–Tuning Verfahren eine weitere Verbesserung der Ergebnisse zu erreichen. In diesem Fall wird mit dem Fine–Tuning tatsächlich die eigentliche Feinabstimmung der Fuzzy–Mengen durchgeführt.

Konkrete Empfehlungen für die Vorgehensweise

Aufgrund der Untersuchungen bezüglich der optimalen Reihenfolge für den Einsatz der einzelnen Verfahren können wir folgende Empfehlungen für den Einsatz des MFOS–M–Systems geben. Wir unterscheiden dabei zwei Anwendungsfälle: 1. Es wurden keine Regeln bzw. Partitions–Mengen vordefiniert. In diesem Fall wird das MFOS–M–System zur automatischen Erzeugung eines geeigneten Fuzzy–Controllers verwendet. 2. Es wurden einige Regeln und Partitions–Mengen vordefiniert. In diesem Fall wird das MFOS–M–System zur Optimierung eines vorhanden Fuzzy–Controllers eingesetzt. Entsprechend dem beabsichtigten Einsatz unterscheidet sich die Vorgehensweise:

1. Einsatz des MFOS–M zum Erzeugen eines Fuzzy–Controllers

Hier empfiehlt sich folgende Vorgehensweise:

1. Erzeugen neuer Regeln und bei Bedarf Erzeugen neuer Fuzzy–Mengen
2. Fine–Tuning

Da erzeugte Regeln korrekt sind, ist eine Korrektur nicht erforderlich. Ebensowenig ist das Löschen von Regeln oder Partitions–Mengen nötig, da in diesem Fall nur unbedingt notwendige Regeln und Fuzzy–Mengen erzeugt werden. Das Fine–Tuning bewirkt die Feinabstimmung der erzeugten Mengen und somit deren Optimierung. In diesem Fall ergibt sich folgendes Diagramm als Modifikation von Abb. 6.9:

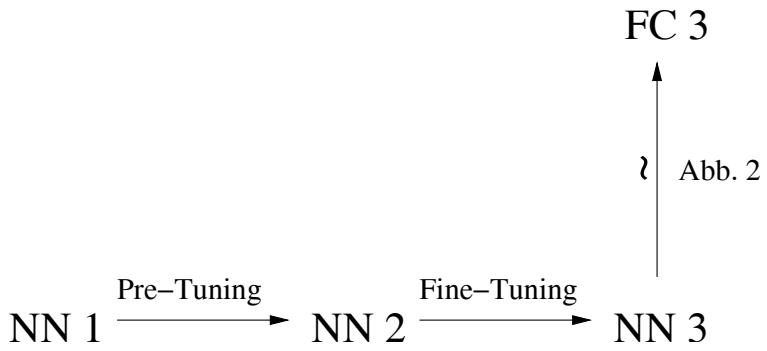

Abbildung 6.12: Reihenfolge Fall 1

Es wird direkt mit einem nicht initialisierten MFOS–M–Netz NN 1 begonnen. Dieses wird durch Pre–Tuning zu NN 2 optimiert, und durch Fine–Tuning weiter zu NN 3. Das endgültige MFOS–M–Netz NN 3 wird anschließend in den optimierten Fuzzy–Controller FC 3 transformiert.

2. Einsatz des MFOS–M zur Optimierung eines vorhandenen Fuzzy–Controllers

Hier empfiehlt sich folgende Vorgehensweise:

1. Fine–Tuning
2. Regeln korrigieren und bei Bedarf neue Partitions–Mengen erzeugen
3. Partitions–Mengen löschen
4. Regeln erzeugen und bei Bedarf neue Partitions–Mengen erzeugen
5. Regeln löschen
6. Fine–Tuning

Das Fine–Tuning zu Beginn verschiebt falsch positionierte Partitions–Mengen an die richtige Stelle. Damit wird schon eine Verbesserung der berechneten Ausgaben erreicht. Insbesondere ist nun eventuell die Korrektur bzw. Erzeugung von bestimmten Regeln nicht mehr notwendig. D.h. es werden weniger neuer Regeln erzeugt als ohne vorheriges Fine–Tuning. Es ist sogar möglich, daß der Fuzzy–Controller nach dem Fine–Tuning schon so weit optimiert wurde, daß weitere Verfahren nicht mehr eingesetzt werden müssen. Andernfalls werden im weiteren Verlauf die Regeln und Partitions–Mengen korrekt eingestellt, wobei nur tatsächlich notwendige Regeln und Fuzzy–Mengen erzeugt werden. Das abschließende Fine–Tuning bewirkt die Feinabstimmung der Mengen und damit die weitere Optimierung. In diesem Fall ergibt sich folgendes Diagramm als Modifikation von Abb. 6.9:

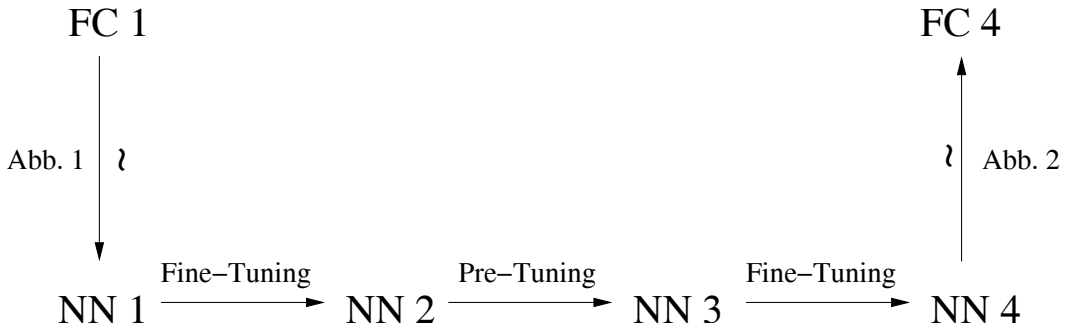

Abbildung 6.13: Reihenfolge Fall 2

Der Fuzzy–Controller FC 1 wird in das äquivalente MFOS–M–Netz NN 1 transformiert. Durch Fine–Tuning wird dieses optimiert zu NN 2, dieses wird durch Pre–Tuning weiter optimiert zu NN 3. Erneutes Fine–Tuning ergibt das endgültige MFOS–M–Netz NN 4, welches in den optimierten Fuzzy–Controller FC 4 transformiert wird.

Pre–Tuning und Fine–Tuning im Vergleich

Die Untersuchungen haben gezeigt, daß die Pre–Tuning– und die Fine–Tuning–Verfahren in einer direkten Wechselwirkung stehen. Pre–Tuning und Fine–Tuning unterscheiden sich prinzipiell in der Art der durchgeführten Modifikationen eines MFOS–M–Netzes. Die Pre–Tuning–Verfahren verändern das Netz in seiner Struktur. D.h. es werden Neuronen und Verbindungen gelöscht bzw. eingefügt. Ebenso werden neue, zusätzliche Gewichte ergänzt bzw. vorhandene Gewichte gelöscht. Das Löschen geschieht nicht analog zu klassischen Feedforward–Netzen durch setzen der Gewichte auf Null, sondern durch tatsächliches entfernen der zugehörigen Werte.

Die Fine–Tuning–Verfahren entsprechen eher den klassischen Lernverfahren und bewirken ausschließlich eine Änderung vorhandener Gewichts–Werte. Beispiel 6.20 verdeutlicht die unterschiedlichen Auswirkungen der Anwendung von Pre– und Fine–Tuning–Verfahren in verschiedenen Reihenfolgen.

Beispiel 6.20 *Sei die gleiche Situation wie in Beispiel 6.3 gegeben. Seien die Ausgabe–Partitions–Mengen definiert als*

$$\begin{aligned} \texttt{schwach} &= (-1.5, 0.5, 2.5) \\ \texttt{mittel} &= (2.8, 4.8, 6.8) \\ \texttt{hoch} &= (6.2, 8.2, 10.2) \end{aligned}$$

Dies führt zu folgenden Ergebnissen: (korrekte Ausgabe, berechnete Ausgabe):

(2.0, 0.5), (5.0, 4.8), (8.0, 8.2)

Insbesondere beim ersten Paar liegt ein hoher Fehler vor, beim zweiten und dritten Paar ist der Fehler nur gering.

Durch die Anwendung der Pre–Tuning–Verfahren wird eine neue Ausgabe–Menge mittelschwach $= (1.1, 2.0, 2.9)$ *erzeugt und die Regel*

IF $x =$ `warm` *THEN* $y =$ `schwach`

korrigiert zu

IF $x =$ `warm` *THEN* $y =$ `mittelschwach`

Damit ergeben sich folgende Ergebnisse: (korrekte Ausgabe, berechnete Ausgabe):

(2.0, 2.0), (5.0, 4.8), (8.0, 8.2)

Der hohe Fehler beim ersten Paar ist korrigiert worden, die geringen Fehler der anderen Paare bleiben unverändert. Wird nun anschließend das Fine–Tuning durchgeführt, ergibt sich eine weitere Optimierung der berechneten Ausgaben:

(2.0, 2.0), (5.0, 4.99704), (8.0, 8.00296)

Erst nach Anwendung der Fine–Tuning–Verfahren sind sämtliche Ergebnisse optimal. Das Pre–Tuning hat nur hohe Fehler korrigiert, geringe Fehler wurden nicht verbessert.

Falls in der Ausgangssituation sofort das Fine–Tuning durchgeführt wird, ergeben sich folgende Resultate: (korrekte Ausgabe, berechnete Ausgabe):

(2, 1.99227), (5, 4.99897), (8, 8.00103)

Mit dem Fine–Tuning wird in dieser Situation also sofort eine Optimierung aller Werte erreicht. Die Ergebnisse sind praktisch genau so gut wie im ersten Fall. Der entscheidende Vorteil dieser Vorgehensweise ist, daß keine zusätzliche Partitions–Menge erzeugt wurde. Falls nach dem Fine–Tuning noch das Pre–Tuning durchgeführt wird, bringt dies hier keine weitere Veränderung mehr.

In Beispiel 6.20 haben wir einen Fall, bei dem die separate Anwendung der Fine–Tuning–Verfahren zu besseren Ergebnissen führt als die separate Anwendung der Pre–Tuning–Verfahren. Beispiel 6.19 hat gezeigt, daß dies nicht immer der Fall ist. Falls nicht genügend Partitions–Mengen vordefiniert wurden, ist eine Optimierung ausschließlich mit Fine–Tuning–Methoden unmöglich. Falls dagegen genügend Partitions–Mengen vordefiniert wurden, verhindert das Fine–Tuning zu Beginn die Erzeugung eigentlich unbenötigter zusätzlicher Fuzzy–Mengen. Ein abschließendes Fine–Tuning ist grundsätzlich empfehlenswert, um eine weitere Optimierung der Ausgaben zu erreichen.

Ob ausschließliches Fine–Tuning zu einer hinreichenden Optimierung führt oder nicht, muß jeweils im Einzelfall überprüft werden. Da ein zu Beginn durchgeführtes Fine–Tuning keine negativen Auswirkungen hat, bleiben die in Abschnitt 6.6.2 gemachten Empfehlungen für die Reihenfolge der Anwendung der einzelnen Verfahren in jedem Fall gültig. Dabei ist es eventuell möglich, daß nach dem ersten Fine–Tuning keine weitere Optimierung mehr erforderlich ist. Die Empfehlungen lassen sich somit auch wie folgt formulieren:

1. Fine–Tuning durchführen
2. Falls der Ausgabe–Fehler zu groß ist:
   (a) Pre–Tuning durchführen
   (b) Fine–Tuning durchführen

Kapitel 7

Vergleich der Optimierungs–Systeme

In Teil II dieser Arbeit wurden unterschiedliche Optimierungs–Systeme für Fuzzy–Controller nach Mamdani vorgestellt: das Verfahren von Lin und Lee (Kapitel 4), das NEFCON–Modell (Kapitel 5) und das MFOS–System (Kapitel 6). In diesem Kapitel wird eine vergleichende Bewertung der unterschiedlichen Ansätze vorgestellt. Insbesondere soll untersucht werden, welche konkreten Fuzzy–Controller optimiert werden können, welche Optimierungsmöglichkeiten unter welchen Voraussetzungen möglich sind und ob eine Übertragung bestimmter Methoden auf andere Systeme durchführbar ist.

7.1 Einsetzbare Fuzzy–Controller

Die vorgestellten Optimierungs–Systeme werden initialisiert, indem ein gegebener Fuzzy–Controller auf das entsprechende neuronale Netz abgebildet wird. Somit repräsentiert das neuronale Netz durch seine Struktur und Gewichte diesen Fuzzy–Controller und berechnet die selben Ausgaben. Allerdings ist nicht jeder beliebige Fuzzy–Controller für die Übertragung auf die bei den einzelnen Systemen verwendeten neuronalen Netze geeignet. Welche Fuzzy–Controller lassen sich mit Hilfe der vorgestellten Systeme optimieren?

Das Verfahren von Lin und Lee:

Das Verfahren von Lin und Lee ermöglicht die Übertragung eines gegebenen Fuzzy–Controllers nach Mamdani auf ein funktional äquivalentes neuronales Netz. Dabei gibt es keine besonderen Einschränkungen. Es sind beliebig viele Eingabe– und Ausgabe–Dimensionen möglich, als Fuzzy–Mengen werden prinzipiell Gauß– bzw. Dreiecks–Mengen verwendet. Die T–Norm zur Berechnung des Erfüllungsgrades der Prämissen der Regeln ist das Minimum. Als Defuzzifizierungsmethode steht ausschließlich die Schwerpunkt–Methode zur Verfügung.

Das NEFCON–Modell:

Das NEFCON–Modell ermöglicht die Übertragung eines gegebenen Fuzzy–Controllers nach Mamdani auf ein funktional äquivalentes neuronales Netz, falls einige Voraussetzungen erfüllt werden. Die Anzahl der Eingabe–Dimensionen ist beliebig, es wird jedoch nur eine Ausgabe–Dimension verwendet. Für die Eingabe–Dimensionen werden Dreiecks–Mengen genommen, für die Ausgabe–Dimension Zacken–Mengen. Die T–Norm zur Berechnung des Erfüllungsgrades der Prämissen der Regeln ist beliebig.

Die Netzausgabe wird ähnlich wie beim Sugeno–Controller als gemittelte, gewichtete Summe bestimmt. Jedoch werden die reellen Ergebnisse einer Regel nicht in Form von reellen Zahlen oder Linearkombinationen angegeben, sondern durch die Umkehrabbildung der Zugehörigkeitsfunktionen der Ausgabe–Partitions–Mengen berechnet. Die Stellwerte müssen für die Anwendung der Lernverfahren so normiert sein, daß im optimalen Zustand die Ausgabe 0 ist, dementsprechend in anderen Situationen die Ausgabe positive oder negative Werte annimmt.

Das MFOS–M–System:

Das MFOS–M–System ermöglicht die Übertragung eines gegebenen Fuzzy–Controllers nach Mamdani auf ein funktional äquivalentes neuronales Netz. Dabei gibt es keine besonderen Einschränkungen. Es sind beliebig viele Eingabe– und Ausgabe–Dimensionen möglich. Als Fuzzy–Mengen werden prinzipiell Gauß– bzw. Dreiecks–Mengen verwendet. Die T–Norm und die T–Conorm sind frei wählbar. Als Defuzzifizierungsmethode stehen die Schwerpunkt–Methode und die Maximum–Methode zur Verfügung.

Vergleich und Bewertung:

Bis auf das NEFCON–Modell werden relativ geringe Voraussetzungen an den zu optimierenden Fuzzy–Controller gemacht. Prinzipiell optimieren die Verfahren Fuzzy–Controller nach Mamdani (s. Abschnitt 3.10). Da Fuzzy–Controller nach Mamdani universelle Approximatoren sind (s. Abschnitt 3.13), stellt dies keine wesentliche Einschränkung dar. (Zur Optimierung von Sugeno–Controllern s. Teil III, Kapitel 8 – 10.) Auch die Beschränkung auf bestimmte T–Normen und T–Conormen ist grundsätzlich unproblematisch.

Die Möglichkeit Gauß– und Dreiecks–Mengen zu verwenden ist für praktische Anwendungen in jedem Fall genügend und sichert ebenfalls Universalität. Die Beschränkung auf die Schwerpunkt–Methode beim Verfahren von Lin und Lee führt in speziellen Fällen eventuell zu Problemen, wie Beispiel 3.31 zeigt.

Durch geeignete Wahl der Regeln läßt sich allerdings auch das Fahrzeug aus Beispiel 3.31 bei Verwendung der Schwerpunkt–Methode korrekt steuern. In diesem Fall muß für jede Situation eindeutig festgelegt werden, ob einem Hindernis nach links oder

rechts ausgewichen werden soll. Dies erfordert jedoch eine genauere Bewertung der Situation, so daß eine größere Anzahl von Regeln benötigt wird als bei Verwendung der Maximum–Methode.

Das NEFCON–Modell unterscheidet sich in einigen wesentlichen Punkten von den anderen Systemen. Die verwendeten Zacken–Mengen gehören nicht zu den sonst üblichen Fuzzy–Mengen. Die häufig eingesetzten Dreiecks–Mengen und Gauß–Mengen sind als Ausgabe–Partitions–Mengen nicht wählbar. Die spezielle Berechnung der Ergebnisse mit Hilfe der Unkehrfunktion der Zugehörigkeitsfunktionen der Ausgabe–Partitions–Mengen ist ebenfalls unüblich und erschwert das Verständnis der Regeln und Zusammenhänge. Gerade diese Verständlichkeit ist ein Vorteil bei der Anwendung von Fuzzy–Controllern, der hier verloren geht. Die notwendige Normierung des Stellwertes und die damit verbundene Skalierung der Meßwerte bringt zusätzliche Schwierigkeiten mit sich, da hierfür zumindest eine weitere Funktion definiert werden muß, die in Wechselwirkung mit dem Fuzzy–Controller steht.

Insgesamt läßt sich folgendes Aussagen:

Ist es das Ziel, einen gegebenen (d.h. vom Anwender erstellten) Fuzzy–Controller nach Mamdani mit Hilfe eines der vorgestellten Systeme zu optimieren, so ist das Verfahren von Lin und Lee oder das MFOS–M–System eine geeignete Wahl. Falls in jedem Fall die Maximum–Methode zur Defuzzifizierung genommen werden soll, ist dies nur mit dem MFOS–M–System möglich. Lediglich das NEFCON–Modell ist zur Optimierung eines vorhandenen Fuzzy–Controllers nur mit Einschränkungen zu verwenden, da die notwendigen Voraussetzungen zu viele Beschränkungen ergeben.

7.2 Optimierungsmöglichkeiten

Sämtliche vorgestellten Optimierungs–Systeme übertragen einen Fuzzy–Controller nach Mamdani auf ein spezielles neuronales Netz (jedes System mit einer individuellen Methode), um dieses zu trainieren. Während des Trainings werden strukturelle Änderungen des Netzes bzw. Änderungen der Gewichte durchgeführt. Diese Adaptionen des neuronalen Netzes korrelieren mit Anpassungen der zu optimierenden Bestandteile des verwendeten Fuzzy–Controllers. D.h. es werden durch das Training die Regeln und die Partitionierungen des Fuzzy–Controllers eingestellt.

Jedoch werden die gleichen Adaptionen mit dem selben Ziel von den einzelnen Optimierungs–Systemen z.T. mit unterschiedlichen Methoden realisiert. Auch ist nicht mit jedem System jede theoretisch mögliche Modifikation durchführbar.

Das Verfahren von Lin und Lee:

Folgende Modifikationen sind prinzipiell möglich:

- Neufestlegung von Regel–Konklusionen
- Erzeugen neuer Ausgabe–Partitions–Mengen
- Modifikation von Eingabe–Partitions–Mengen (Modalwert und Weite)
- Modifikation von Ausgabe–Partitions–Mengen (Modalwert und Weite)

Diese Modifikationen werden von einem hybriden Lernalgorithmus situationsabhängig durchgeführt. Kriterium für die Wahl der Modifikationen sind ausschließlich die mit dem Backpropagation–Algorithmus berechneten Änderungen der Ausgabe–Partitions–Mengen (zur genaueren Beschreibung s. Kapitel 4):

- falls die zu $\widetilde{B}_{j*}$ mit dem Backpropagation–Algorithmus neu berechnete Menge $\widetilde{B}_{j*}^{neu}$ zu keiner vorhandenen Ausgabe–Partitions–Menge ähnlich genug ist, wird $\widetilde{B}_{j*}^{neu}$ als neue, zusätzliche Ausgabe–Partitions–Menge eingefügt und als neue Konklusion von Regeln mit hohem Erfüllungsgrad der Prämissen eingesetzt
- falls die berechnete Menge $\widetilde{B}_{j*}^{neu}$ zur Originalmenge $\widetilde{B}_{j*}$ ähnlich genug ist, erfolgt die Änderung von $\widetilde{B}_{j*}$ zu $\widetilde{B}_{j*}^{neu}$
- falls die berechnete Menge $\widetilde{B}_{j*}^{neu}$ zu einer anderen Ausgabe–Partitions–Menge als $\widetilde{B}_{j*}$ ähnlich genug, wird diese andere Menge als neue Konklusion von Regeln mit hohem Erfüllungsgrad eingesetzt
- die Eingabe–Partitions–Menge werden in jedem Fall nach der Backpropagation–Regel geändert

Das NEFCON–Modell:

Folgende Modifikationen sind prinzipiell möglich:

- Erzeugen neuer Regeln
- Modifikation von Eingabe–Partitions–Mengen (nur Weite)
- Modifikation von Ausgabe–Partitions–Mengen (nur Weite)

Das NEFCON–Modell stellt zwei Lernalgorithmen zur Verfügung (s. Kapitel 5). Lernalgorithmus 1 erzeugt bei vordefinierten Partitionierungen eine vollständige Regelbasis. Dazu wird zunächst jede Regel erzeugt, die sich mit den gegebenen Partitions–Mengen darstellen läßt. D.h. jede Kombination von Eingabe–Partitions–Mengen wird als Regel–Prämisse eingesetzt, jede Ausgabe–Partitions–Menge wird als Konklusion verwendet.

Anschließend werden "falsche" und überflüssige Regeln entfernt, bis eine geeignete Regelbasis übrig bleibt. Kriterium für die Einstufung einer Regel als "falsch" oder "richtig" ist ausschließlich das korrekte Vorzeichen des berechneten Ergebnisses der Regel.

Lernalgorithmus 2 führt eine Feinabstimmung der Partitions–Fuzzy–Mengen durch, jedoch ausschließlich eine Änderung der Weiten. Eine Verschiebung von Partitions–Fuzzy–Mengen (ändern von Modalwerten) ist nicht vorgesehen. Das Ziel dabei ist, den Einfluß "guter" Regeln zu stärken und den Einfluß "schlechter"Regeln zu verringern (wiederum mit Hilfe des Vorzeichens bewertet). Die Änderungen berücksichtigen den Fehleranteil der Regeln und den Erfüllungsgrad ihrer Prämissen. Durch Verbreitern von Eingabe–Partitions–Mengen wird der Erfüllungsgrad der Prämissen "guter" Regeln vergrößert. Durch Verbreitern von Konklusions–Mengen "guter" Regeln wird der Anteil dieser Regeln am Ergebnis vergrößert. "Schlechte" Regeln erfahren die gegenteiligen Modifikationen.

Das MFOS–M–System:

Folgende Modifikationen sind prinzipiell möglich:

- Neufestlegung von Regel–Konklusionen
- Erzeugen neuer Regeln
- Löschen unnötiger Regeln
- Erzeugen neuer Eingabe–Partitions–Mengen
- Erzeugen neuer Ausgabe–Partitions–Mengen
- Löschen unnötiger Partitons–Fuzzy–Mengen
- Modifikation von Eingabe–Partitions–Mengen (Modalwert und Weite)
- Modifikation von Ausgabe–Partitions–Mengen (Modalwert und Weite)

Die Auswahl und Reihenfolge der Modifikationen durch das MFOS–M–System ist dem User überlassen (s. Abschnitt 6.6). Zur genauen Darstellung der einzelnen Methoden s. die Abschnitte 6.3, 6.4 und 6.5. Zur Neufestlegung von Regel–Konklusionen (d.h. Korrigieren von Regeln) wird diejenige Ausgabe–Partitions–Menge als neue Konklusion ausgewählt, mit der die überprüfte Regel den minimalen Fehler verursacht. Korrigiert wird jeweils die Regel mit dem maximalen Erfüllungsgrad ihrer Prämisse, da sie den höchsten Anteil am Ergebnis hat.

Neue Regeln werden erzeugt, falls zu einem Trainingsbeispiel keine geeignete Regel vorhanden ist. Kriterium hierfür ist der Erfüllungsgrad der Prämissen der Regeln.

Zur Definition der Prämisse einer neuen Regel werden diejenigen Eingabe–Partitions–Mengen ausgewählt, die das aktuelle Trainingsbeispiel am besten repräsentieren. Kriterium hierfür ist der Zugehörigkeitsgrad der Eingabe–Werte zu den Eingabe–Partitions–Mengen. Die Konklusion wird analog mit den Ausgabe–Partitions–Mengen bestimmt. Falls Regeln nachweislich keinen Einfluß auf das Ergebnis haben, werden sie gelöscht. Dies wird mit Hilfe der berechneten Schnitthöhen überprüft.

Falls beim Korrigieren bzw. Erzeugen von Regeln festgestellt wird, daß mit keiner vorhandenen Partitions–Fuzzy–Menge der Erfüllungsgrad groß genug ist bzw. der von einer Regel verursachte Fehler klein genug ist, werden neue Eingabe– bzw. Ausgabe–Partitions–Mengen erzeugt. Als Modalwert werden die Eingabe– bzw. Ausgabe–Werte des aktuellen Trainingsbeispiels verwendet. Die Weite wird so gewählt, das eine Überlappung zu eventuellen Nachbar–Mengen gegeben ist.

Auf diese Weise wird erstens sichergestellt, daß die erzeugten Partitions–Fuzzy–Mengen perfekt zum aktuellen Trainingsbeispiel passen. Zweitens wird erreicht, daß sich die neu erzeugten Partitions–Fuzzy–Mengen in die vorhandenen Partitionierungen einfügen. Somit sind neu erzeugte Partitions–Mengen auch für andere Trainingsbeispiele mit ähnlichen Werten geeignet.

Neben dem Erzeugen neuer Partitions–Fuzzy–Mengen ist auch das Löschen unnötiger Partitions–Fuzzy–Mengen möglich. Gleichwertige Eingabe–Partitions–Mengen lassen sich zu einer Menge zusammenfassen. Kriterium hierfür ist, ob zwei Nachbar–Mengen zur Unterscheidung von Ausgabe–Werten notwendig sind oder nicht.

Zusätzlich zu diesen als Pre–Tuning bezeichneten Modifikationen ist auch ein Fine–Tuning möglich. Als Fine–Tuning wird beim MFOS–M–System die Anpassung der Parameter der Eingabe– und Ausgabe–Partitions–Mengen (Modalwert und Weite) durchgeführt. Dies geschieht mit dem Backpropagation–Algorithmus oder wahlweise mit den Modifikationen $\bar{\delta} - \delta$–Regel, Momentum–Version und einer speziellen Kombination von beiden.

Vergleich und Bewertung:

Die einzelnen Optimierungs–Systeme ermöglichen die Anpassung ausgesuchter Komponenten eines Fuzzy–Controllers. Jedoch ist es nur mit dem MFOS–M–System möglich, jede Komponente eines Fuzzy–Controllers nach Bedarf optimieren zu lassen. Die anderen Systeme beschränken sich jeweils auf einige vorgegebene Bestandteile.

Beim Verfahren von Lin und Lee bewirkt der hybride Lernalgorithmus, daß jede vorhandene Regel die optimale Konklusion erhält. Zudem werden die Eingabe– und Ausgabe–Partitions–Mengen optimiert. Jedoch ist weder ein Erzeugen zusätzlicher Eingabe–Partitions–Mengen noch ein Erzeugen zusätzlicher Regeln möglich. Wird beim Konfigurieren des zugehörigen Netzes eine zur korrekten Steuerung unverzichtbare Eingabe–

Partitions–Menge oder Regel nicht berücksichtigt, ist eine optimale Einstellung des Fuzzy–Controllers mit dem gegebenen Lernalgorithmus unmöglich.

Beispiel 7.1 *Die folgenden Fuzzy–Mengen und Regeln ergeben die Partitionierungen und die Regelbasis für einen einfachen Fuzzy–Controller zur Steuerung eines Heizgeräts:*

Eingabe (Temperatur in °C): Dreiecks–Mengen auf dem Grundraum $[13, 23]$*:*

$$\begin{aligned} \texttt{sehr kalt} &\mathrel{\widehat{=}} \widetilde{A}_1 = (13, 15, 17) \\ \texttt{kalt} &\mathrel{\widehat{=}} \widetilde{A}_2 = (16, 18, 20) \\ \texttt{warm} &\mathrel{\widehat{=}} \widetilde{A}_3 = (19, 21, 23) \end{aligned}$$

Ausgabe (Heizleistung): Dreiecks–Mengen auf dem Grundraum $[0, 10]$*:*

$$\begin{aligned} \texttt{schwach} &\mathrel{\widehat{=}} \widetilde{B}_1 = (0, 2, 4) \\ \texttt{mittel} &\mathrel{\widehat{=}} \widetilde{B}_2 = (3, 5, 7) \\ \texttt{hoch} &\mathrel{\widehat{=}} \widetilde{B}_3 = (6, 8, 10) \end{aligned}$$

Die linguistischen Variablen sind x für die Temperatur und y für die Heizleistung.

Die richtigen Regeln sind:

$$\begin{aligned} &\texttt{IF } x = \texttt{sehr kalt } \texttt{THEN } y = \texttt{hoch} \\ &\texttt{IF } x = \texttt{kalt } \texttt{THEN } y = \texttt{mittel} \\ &\texttt{IF } x = \texttt{warm } \texttt{THEN } y = \texttt{schwach} \end{aligned}$$

Wird nun bei der Erstellung des Netzes die Eingabe–Partitions–Menge für `kalt` *und die Regel*

$$\texttt{IF } x = \texttt{kalt } \texttt{THEN } y = \texttt{mittel}$$

vergessen, so ist es mit dem Verfahren von Lin und Lee nicht möglich, einen korrekt funktionierenden Fuzzy–Controller zu erstellen. Hierfür wären eine zusätzliche Eingabe–Partitions–Menge und eine zusätzliche Regel erforderlich, die jedoch nicht erzeugt werden können.

Das Löschen von Regeln bzw. Partitions–Fuzzy–Mengen dient im wesentlichen zur Verbesserung der Performanz, es hat keinen Einfluß auf den Fehler. Daher ist es zur Optimierung des Ein– Ausgabe–Verhaltens nicht erforderlich. Jedoch werden mit dem hybriden Lernalgorithmus fast in jedem Schritt zusätzliche Ausgabe–Partitions–Mengen erzeugt, die eigentlich nicht erforderlich sind und eventuell einen Schritt später nicht mehr benötigt werden. Daher wäre eine Möglichkeit zum Löschen von Partitions–Fuzzy–Mengen bzw. eine Kontrolle über die Erzeugung neuer Partitions–Fuzzy–Mengen wünschenswert.

Im Gegensatz zum Verfahren von Lin und Lee ist es mit dem NEFCON–Modell möglich, neue Regeln zu erzeugen. Jedoch beschränkt sich der Algorithmus auf das Kombinieren sämtlicher vordefinierter Partitions–Fuzzy–Mengen. Ein gezieltes Erzeugen einzelner Regeln nach Bedarf ist nicht vorgesehen.

Voraussetzung für die Erzeugung einer vollständigen Regelbasis ist, daß bis auf eine Feinabstimmung geeignete Partitionierungen vordefiniert wurden. D.h. falls eine zur korrekten Steuerung unverzichtbare Partitions–Fuzzy–Menge beim Konfigurieren des zugehörigen Netzes vergessen wurde, ist eine optimale Einstellung des Fuzzy–Controllers mit dem gegebenen Lernalgorithmus unmöglich.

Beispiel 7.2 *Sei die gleiche Situation wie in Beispiel 7.1 gegeben. Wird nun bei der Erstellung des NEFCON–Modells die Eingabe–Partitions–Menge* `kalt` *auf dem Eingaberaum und die Regel*

IF $x =$ `kalt` *THEN* $y =$ `mittel`

vergessen, so ist es mit Lernalgorithmus 1 nicht möglich, die gegebenen Fuzzy–Mengen so einzustellen, daß die Steuerung in jeder Situation funktioniert. Hierfür wäre eine zusätzliche Eingabe–Partitions–Menge erforderlich, die jedoch nicht erzeugt werden kann.

Aus dem gleichen Grund ist es auch Lernalgorithmus 2 bei weglassen der Eingabe–Partitions–Menge `kalt` *nicht möglich, eine für jede Situation geeignete Regelbasis zu erstellen. Durch Kombination der vorhandenen Partitions–Fuzzy–Mengen lassen sich in diesem Fall nicht alle tatsächlich benötigten Regeln erzeugen.*

Zur optimalen Einstellung der Partititions–Fuzzy–Mengen erfolgt lediglich eine Änderung der Weiten, nicht der Modalwerte. Das ist ungünstig, falls einzelne Mengen nicht optimal positioniert sind. Da keine neuen Partitions–Fuzzy–Mengen erzeugt werden können, müssen in jedem Fall die Partitionierungen vom User vordefiniert werden. Dabei wird häufig eine gleichmäßige Verteilung der Partitions–Fuzzy–Mengen gewählt. Beispiel 6.16 zeigt, daß dies nicht immer die beste Wahl ist. Eine Möglichkeit, Partitions–Fuzzy–Mengen zu verschieben, würde daher eine bessere Anpassung der Partitions–Fuzzy–Mengen ermöglichen.

Die Bewertung der Regel–Ergebnisse mit dem Ziel, den Einfluß "guter" Regeln zu vergrößern und den Einfluß "schlechter" Regeln zu verringern, ist im Prinzip eine gute Idee. Jedoch scheint das gewählte Kriterium zur Bewertung nicht optimal zu sein. Eine Regel wird ausschließlich danach bewertet, ob ihr Anteil am Ergebnis das richtige Vorzeichen hat.

Beispiel 7.3 *Sei zu einem Trainingsbeispiel das richtige Ergebnis* $+0.1$.

Sei das berechnete Ergebnis von Regel 1: -0.1.
Sei das berechnete Ergebnis von Regel 2: $+10.0$.

In diesem Fall verursacht Regel 1 einen Fehler von $\mid 0.1 - -0.1 \mid = 0.2$,
Regel 2 verursacht einen Fehler von $\mid 0.1 - 10.0 \mid = 9.9$.

Dennoch wird Regel 1 als "schlecht" bewertet und Regel 2 als "gut", da nur das Ergebnis von Regel 2 das richtige Vorzeichen hat.

Dieses Beispiel zeigt, daß mit dem gegebenen Kriterium nicht in jedem Fall die tatsächlich beste Regel am besten bewertet wird. Daher scheint ein anderes bzw. zusätzliches Kriterium wie der verursachte Fehler für die Bewertung von Regeln besser geeignet zu sein als ausschließlich das Vorzeichen des mit der Regel berechneten Ergebnisses. Mit dem entsprechenden Algorithmus des NEFCON–Modells wird der Fehleranteil einer Regel bereits berechnet, jedoch wird er nur verwendet, um die Stärke der Änderungen zu steuern, und nicht, um die Regel zu bewerten.

Das MFOS–M–System bietet alle Modifikationsarten, die prinzipiell möglich sind. Vorhandene Regeln werden korrigiert, nach Bedarf werden zusätzliche Regeln erzeugt bzw. überflüssige Regeln gelöscht. Ebenso ist das Erzeugen und Löschen von Partitions–Fuzzy–Mengen möglich. Bei der Konfiguration eines MFOS–M–Netzes wird ein vom Anwender vorgegebener Fuzzy–Controllers übernommen und nach Bedarf optimiert. Dabei ist es unbedeutend, ob einzelne Regeln oder Partitions–Fuzzy–Mengen noch nicht definiert sind, oder nur vorhandene Partitions–Fuzzy–Mengen verändert werden müssen.

Beispiel 7.4 *Sei die gleiche Situation wie in Beispiel 7.1 gegeben. Wird nun bei der Erstellung des MFOS–M–Systems die Eingabe–Partitions–Menge* `kalt` *auf dem Eingaberaum und die Regel*

IF $x =$ `kalt` *THEN* $y =$ `mittel`

vergessen, so erzeugt das MFOS–M–Verfahren die fehlende Regel und eine neue Eingabe–Partitions–Menge

`kalt` $\widehat{=}\ \widetilde{A}_2 = (16.2, 18.0, 19.8)$

mit der die berechneten Ergebnisse genau so gut sind wie mit der ursprünglichen Menge $(16, 18, 20)$ *(vgl. Beispiel 6.10).*

Die einzelnen Modifikationen werden nach Wahl des Benutzers in Abhängigkeit der Trainingsdaten durchgeführt. Zur Bewertung einer Regel wird der Fehler berücksichtigt, den die Regel verursacht. Neue Partitions–Mengen werden nach Bedarf erzeugt und von den Regeln verwendet. Dabei wird vermieden, wie beim Verfahren von Lin und Lee, mehrere ähnliche Partitions–Fuzzy–Mengen zu erzeugen, von denen eine genügen würde.

Zur Feinabstimmung der Partitions–Fuzzy–Mengen werden die Modalwerte und Weiten mit Hilfe des Backpropagation–Algorithmus und einiger Modifikationen angepaßt. Aufgrund der Berücksichtigung sämtlicher prinzipiell möglicher Modifikationen gibt es somit keine besonderen Einschränkungen oder Voraussetzungen für die Optimierung eines Fuzzy–Controllers mit dem MFOS–M–System.

Insgesamt läßt sich folgendes Aussagen:

Lediglich mit dem MFOS–M–System lassen sich sämtliche Komponenten eines Fuzzy–Controllers nach Bedarf optimieren. Eine konkrete Einschränkung oder notwendige Voraussetzung wie bei den anderen Systemen läßt sich nicht feststellen. Daher ist zu erwarten, daß mit dem MFOS–M–System in nahezu allen Fällen die Optimierung eines vorgegebenen Fuzzy–Controllers zu erreichen ist, unabhängig von der Konfiguration, die der Anwender vorgibt.

Die anderen vorgestellten Optimierungs–Systeme sind von folgenden Voraussetzungen abhängig:

- für den erfolgreichen Einsatz des Verfahrens von Lin und Lee müssen in jedem Fall die Anzahl der Eingabe–Partitions–Mengen und die Anzahl der Regeln vorher korrekt bestimmt werden
- für den erfolgreichen Einsatz des NEFCON–Modells sind bis auf Feinabstimmung korrekte Partitionierungen unbedingt erforderlich

Diese Bedingungen müssen vom Anwender sichergestellt werden. Dabei sind eventuell andere Systeme zur automatischen Generierung von Fuzzy–Mengen oder Fuzzy–Regeln hilfreich. Wenn die genannten Bedingungen erfüllt werden, ist mit jedem dieser Systeme die Optimierung eines Fuzzy–Controllers zu erreichen.

7.3 Vergleich gemeinsamer Verfahren

Sämtliche vorgestellten Optimierungs–Systeme stellen Methoden zur Verfügung, Fuzzy–Controller zu optimieren. Manche Methoden ähneln sich im Ziel und Durchführung,

andere sind nur in einem bestimmten System vorhanden. Zum Teil werden die gleichen Ziele mit unterschiedlichen Methoden realisiert. Welche Methoden sind besser geeignet? Lassen sich bestimmte Methoden eines Systems auf ein anderes System übertragen?

Neufestlegung von Regel–Konklusionen, erzeugen neuer Ausgabe–Partitions–Mengen:

Das Verfahren von Lin und Lee und das MFOS–M–System ermöglichen das Ändern von Regel–Konklusionen und das Erzeugen neuer Ausgabe–Partitions–Mengen. Beide Systeme setzen eine tatsächlich andere Partitions–Fuzzy–Menge als neue Konklusion einer Regel ein. Daher soll hier verglichen werden, wie dieser Vorgang durchgeführt wird.

Sämtliche Modifikationen des Verfahrens von Lin und Lee basieren auf dem Backpropagation–Algorithmus. Es wird grundsätzlich angenommen, daß eine mit dem Backpropagation–Algorithmus aus der Ausgabe–Partitions–Menge $\widetilde{B}_{j*}$ neu berechnete Menge $\widetilde{B}_{j*}^{neu}$ die bessere Konklusion für eine Regel mit hohem Erfüllungsgrad der Prämisse ist. Das heißt, die Konklusions–Menge $\widetilde{B}_{j*}$ soll durch $\widetilde{B}_{j*}^{neu}$ ersetzt werden, falls der Erfüllungsgrad der Prämisse einer Regel mit Konklusion $\widetilde{B}_{j*}$ groß genug ist. Dies wird durch eine der drei folgenden Möglichkeiten durchgeführt:

1. ist $\widetilde{B}_{j*}^{neu}$ ähnlich zu $\widetilde{B}_{j*}$, wird die $\widetilde{B}_{j*}$ geändert zu $\widetilde{B}_{j*}^{neu}$
2. ist $\widetilde{B}_{j*}^{neu}$ ähnlich zu einer anderen Fuzzy–Menge $\widetilde{B}_{j*}^{x}$, wird diese als neue Konklusion der betroffenen Regeln eingesetzt
3. ist $\widetilde{B}_{j*}^{neu}$ zu keiner vorhandenen Ausgabe–Partitions–Menge ähnlich genug, wird $\widetilde{B}_{j*}^{neu}$ als neue Fuzzy–Menge eingefügt und als Konklusion betroffener Regeln eingesetzt

Das MFOS–M–System ändert die Konklusion einer Regel, falls der Fehler, den diese Regel verursacht, zu groß ist. Als neue Konklusion wird die Ausgabe–Partitons–Menge gewählt, die bei dieser Regel den geringsten Fehler verursacht. Falls auch dieser Fehler zu groß ist, wird eine neue Ausgabe–Partitions–Menge erzeugt. Somit bekommt jede Regel in einem Schritt sofort die richtige Konklusion. Insbesondere werden nur tatsächlich benötigte Fuzzy–Mengen erzeugt. Beim Verfahren vom Lin und Lee werden die Konklusionen eventuell mehrmals geändert, da in jedem Durchgang neue Fuzzy–Mengen mit dem Backpropagation–Algorithmus berechnet werden. Auch ist es möglich, immer wieder zusätzliche Fuzzy–Mengen zu erzeugen, die noch etwas besser sind als die zuvor definierten. Somit werden viele Fuzzy–Mengen erzeugt, die nicht unbedingt benötigt werden.

Letztlich bewirken beide Systeme, daß eine Regel die optimale Konklusion bekommt. Da auch mit dem MFOS–M–System das Backpropagation–Verfahren möglich ist, sind die Ergebnisse des MFOS–M–Systems in keinem Fall schlechter. Da der Aufwand beim

Verfahren von Lin und Lee größer ist, ist eine Übertragung dieser Methode auf das MFOS–M–System nicht sinnvoll. Ungekehrt ist zu erwarten, daß eine direkte Bestimmung der Konklusion und eine direkte Erzeugung neuer Partitions–Fuzzy–Mengen aufgrund von Trainingsdaten das Verfahren von Lin und Lee beschleunigen würde. Daher scheint es sinnvoll zu sein, eine Methode zur Korrektur von Regeln und zur Erzeugung von Fuzzy–Mengen als Pre–Tuning in das Verfahren von Lin und Lee zu integrieren.

Erzeugen neuer Regeln

Das NEFCON–Modell und das MFOS–M–System bieten die Möglichkeit, neue Regeln zu erzeugen. Dieses Ziel wird von beiden Systemen jedoch auf grundlegend unterschiedliche Weise realisiert:

- Das NEFCON–Modell kombiniert jede vorhandene Eingabe–Partitions–Menge zu Regel–Prämissen und wählt jede vorhandene Ausgabe–Partitions–Menge als Konklusion. Anschließend werden alle ungeeigneten und unnötigen Regeln entfernt.
- Das MFOS–M–System erzeugt gezielt eine neue Regel, falls für eine bestimmte Situation keine geeignete Regel vorhanden ist. Dabei werden direkt die optimal geeigneten Partitions–Fuzzy–Mengen als Prämisse und Konklusion ausgewählt.

Das NEFCON–Modell ist prinzipiell abhängig von bis auf Feinabstimmung korrekt definierten Partitionierungen. Unter dieser Voraussetzung ist das Verfahren geeignet, eine korrekte Regelbasis zu erzeugen. Jedoch scheint es umständlicher zu sein als das Verfahren des MFOS–M–Systems, das direkt die benötigten Regeln erzeugt. Eine Übertragung des NEFCON–Verfahrens auf das MFOS–M–System wäre prinzipiell möglich, falls die Bewertung der Regeln modifiziert wird. Jedoch scheint die Übertragung nicht sinnvoll, da sich bereits mit dem MFOS–M–Verfahren eine komplette Regelbasis erzeugen läßt.

Eine Übertragung der MFOS–M–Methode zum Erzeugen von Regeln auf das NEFCON–System ist prinzipiell möglich. Aufgrund der speziellen Art der Regelauswertung beim NEFCON–System ist jedoch zu erwarten, daß die analog zur MFOS–M–Methode ausgewählten Ausgabe–Partitions–Mengen zu hohen Fehlern führen. Eine anschließende Anpassung der Partitions–Fuzzy–Menge mit dem NEFCON–Lernalgorithmus 1 könnte diesen Fehler minimieren.

Modifikation von Partitions–Fuzzy–Mengen

Jedes vorgestellte Optimierungs–System ermöglicht die Modifikation von Partitions–Fuzzy–Mengen. Das Verfahren von Lin und Lee und das MFOS–M–System verwenden dazu den Backpropagation–Algorithmus. Beim MFOS–M–System werden zusätzlich einige Modifikationen des Standard–Backpropagation–Verfahrens eingesetzt. Das

NEFCON–System modifiziert die Weiten der Partitions–Fuzzy–Mengen mit einer anderen Methode, basierend auf einem Fehler–Maß und der Bewertung der Regeln.

Das Backpropagation–Verfahren ist grundsätzlich geeignet für eine Anwendung mit dem NEFCON–System. Falls zusätzlich zu den Weiten auch die Modalwerte der Partitions–Fuzzy–Mengen angepaßt werden, ist eine Verbesserung zu erwarten. Umgekehrt ist auch eine Übertragung der NEFCON–Methode auf die anderen Systeme prinzipiell möglich. Jedoch muß dazu die Bewertung der Regeln anders vorgenommen werden, z.B. aufgrund des Fehlers, den eine Regel verursacht.

Weitere von mehreren der vorgestellten Optimierungs–Systemen durchgeführten Verfahren gibt es nicht. Die Möglichkeiten

- Löschen unnötiger Regeln
- Erzeugen neuer Eingabe–Partitions–Mengen
- Löschen unnötiger Partitions–Fuzzy–Mengen

stellt nur das MFOS–M–System zur Verfügung. Daher gibt es keine weiteren Verfahren, die verglichen werden können.

7.4 Übertragung von Verfahren

Zum Abschluß dieser Untersuchungen soll nun überprüft werden, in wie Weit sich die einzelnen Optimierungs–Verfahren auf eines der anderen Systeme übertragen lassen. Insbesondere geht es dabei um die Frage, ob sich die erwähnten Defizite der einzelnen Optimierungs–Systeme beheben lassen bzw. ob sich die Optimierungs–Ziele effektiver erreichen lassen.

Das Verfahren von Lin und Lee:

Falls bei der Erstellung des zugehörigen Netzes eine zur korrekten Steuerung unverzichtbare Eingabe–Partitions–Menge oder Regel nicht berücksichtigt wird, ist eine optimale Einstellung des Fuzzy–Controllers mit dem Verfahren von Lin und Lee nicht möglich (s. Beispiel 7.1). Daher würde eine Erweiterung um ein Verfahren zum Erzeugen von Regeln bzw. Eingabe–Partitions–Mengen die Einsatzmöglichkeiten dieses Optimierungs–Systems erweitern.

Das NEFCON–Modell und das MFOS–M–System stellen unterschiedliche Methoden zum Erzeugen von Regeln zur Verfügung. Ist es möglich, eine dieser Methoden auf das Verfahren von Lin und Lee zu übertragen?

- Die NEFCON–Methode erzeugt zunächst alle Regeln, die aus den vordefinierten Partitions–Fuzzy–Mengen durch Kombination konstruiert werden können. Anschließend werden abhängig von der Bewertung der Regeln "falsche" und überflüssige Regeln entfernt. Mit einigen Modifikationen ist eine Übertragung dieser Methode auf das Verfahren von Lin und Lee prinzipiell möglich.

  Das verwendete Netz ist so konstruiert, daß eine Initialisierung mit sämtlichen aus den Partitions–Fuzzy–Mengen kombinierbaren Regeln durchführbar ist. Die Bewertung der Regeln muß allerdings modifiziert werden, da die Ausgaben beim Verfahren von Lin und Lee nicht normiert sind. Werden die einzelnen Regeln z.B. mit Hilfe des von ihnen verursachten Fehlers bewertet, ist eine Auswahl geeigneter Regeln analog der NEFCON–Methode durchführbar.

- Die MFOS–M–Methode erzeugt eine neue Regel, falls zu einem Trainingsbeispiel der Erfüllungsgrad der Prämisse von jeder vorhandenen Regel zu gering ist. Die verwendeten Eingabe– und Ausgabe–Partitions–Mengen werden mit Hilfe des maximalen Zugehörigkeitsgrades der Trainingsdaten bestimmt. Diese Methode ist analog beim Verfahren von Lin und Lee durchführbar.

  Neue Eingabe–Partitions–Mengen werden ausschließlich mit dem MFOS–M–System erzeugt. Falls beim Erzeugen einer Regel der maximale Zugehörigkeitsgrad der Trainingsdaten zu den Eingabe–Partitions–Mengen zu gering ist, wird eine zusätzliche Eingabe–Partitititions–Menge erzeugt. Modalwert ist der Eingabe–Wert, Weite ist das 1.2–fache des Abstandes zum Modalwert der nächsten Nachbar–Menge. Diese Methode ist analog beim Verfahren von Lin und Lee durchführbar.

Somit lassen sich durch Übertragung der MFOS–M–Methoden zum Erzeugen von Regeln und Eingabe–Partitions–Mengen die Einschränkungen für den Einsatz des Verfahrens von Lin und Lee beheben. Alternativ ist eine Übertragung der NEFCON–Methode zum Erzeugen von Regeln auf das Verfahren von Lin und Lee möglich, falls die Bewertung der Regeln mit einer geeigneten Methode erfolgt.

Das Verfahren von Lin und Lee bestimmt mit der hybriden Methode (s. Kapitel 4) in mehreren Schritten die optimale Konklusion für jede vorhandene Regel. Dabei werden fast in jedem Schritt zusätzliche Ausgabe–Partitions–Mengen erzeugt, die einige Schritte später nicht mehr benötigt werden. Daher wäre, wie in Abschnitt 7.3 vorgeschlagen, eine Kombination mit der MFOS–M–Methode zum Korrigieren von Regeln sinnvoll:

- In einer Pre–Tuning–Phase wird analog zur MFOS–M–Methode zum Korrigieren von Regeln für jede Regel die beste Ausgabe–Partitions–Menge als Konklusion bestimmt. Da die MFOS–M–Methode die Ausgabe–Partitions–Mengen aufgrund des Zugehörigkeitsgrades der Trainingsdaten zu diesen Mengen bewertet, ist eine

Übertragung dieser Methode auf das Verfahren von Lin und Lee möglich. Falls der maximale Zugehörigkeitsgrad der Trainingsdaten zu den Ausgabe–Partitions–Mengen zu gering ist, läßt sich analog zur MFOS–M–Methode zum Erzeugen von Partitions–Mengen eine geeignete Ausgabe–Partitions–Menge kreieren und als Konklusion einsetzen.

- Die hybride Methode von Lin und Lee wird anschließend als Fine–Tuning durchgeführt.

Es ist zu erwarten, daß durch diese Vorgehensweise der Aufwand für die hybride Methode von Lin und Lee stark reduziert wird, da sämtliche Regeln vorher eine bis auf Feinabstimmung korrekte Konklusion erhalten. Insbesondere werden dadurch weniger zusätzliche Ausgabe–Partitions–Mengen erzeugt, die nicht unbedingt benötigt werden.

Das NEFCON–Modell:

Falls bei der Erstellung des zugehörigen Netzes eine zur korrekten Steuerung unverzichtbare Eingabe–Partitions–Menge nicht berücksichtigt wird, ist eine optimale Einstellung des Fuzzy–Controllers mit dem NEFCON–Modell nicht möglich (s. Beispiel 7.2). Gleiches gilt, falls eine unbedingt benötigte Ausgabe–Partitions–Menge nicht berücksichtigt wurde. Daher würde eine Erweiterung um ein Verfahren zum Erzeugen von Partitions–Fuzzy–Mengen die Einsatzmöglichkeiten dieses Optimierungs–Systems erweitern.

Eine Übertragung der MFOS–M–Methode zum Erzeugen neuer Partitions–Fuzzy–Mengen auf das NEFCON–Modell ist grundsätzlich möglich. Jedoch ist zu erwarten, daß die mit Hilfe der Trainingsdaten mit der MFOS–M–Methode erzeugten Ausgabe–Partitions–Mengen nicht optimal für ein NEFCON–Netz sind. Der Grund hierfür liegt in der speziellen Berechnung der Regel–Ergebnisse (s. Kapitel 5). Nach Erzeugung aller benötigten Partitions–Fuzzy–Mengen lassen sich mit der NEFCON–Methode zur Erzeugung einer Regelbasis alle benötigten Regeln erzeugen. Anschließend müssen mit dem NEFCON–Lernalgorithmus 1 die Partitions–Fuzzy–Mengen korrekt eingestellt werden.

Alternativ ist es möglich, die MFOS–M–Methode zum Erzeugen von Regeln auf das NEFCON–Modell zu übertragen und direkt zusätzliche Regeln zu erzeugen. Auch bei dieser Vorgehensweise ist zu erwarten, daß die analog zur MFOS–M–Methode ausgewählten Konklusionen zunächst nicht optimal sind. Eine anschließende Anpassung der Partitions–Fuzzy–Mengen mit NEFCON–Lernalgorithmus 1 minimiert den Fehler. Beide Alternativen beheben die Einschränkungen für den Einsatz des NEFCON–Modells.

Das MFOS–M–System:

Die Untersuchungen in Abschnitt 7.2 haben gezeigt, daß für die Optimierung eines Fuzzy–Controllers mit dem MFOS–M–System keine besonderen Einschränkungen oder

Voraussetzungen bestehen. Somit gibt es keinen grundsätzlichen Bedarf, die Einsatzmöglichkeiten dieses Optimierungs–Systems zu erweitern.

Dennoch stellt sich die Frage, ob es sinnvoll ist, bestimmte Methoden eines anderes Systems auf das MFOS–M–System zu übertragen. Möglicherweise ergeben sich dadurch effektivere Verfahren oder alternative Lösungswege.

Folgende Modifikationsmöglichkeiten werden von MFOS–M–System und mindestens einem der anderen vorgestellten Optimierungs–Systeme unterstützt:

- Ändern von Regel–Konklusionen
- Erzeugen neuer Ausgabe–Partitions–Mengen
- Modifikation von Partitions–Fuzzy–Mengen
- Erzeugen neuer Regeln

Wie bereits in Abschnitt 7.3 dargestellt wurde, gibt es prinzipiell folgende Möglichkeiten, das MFOS–M–System mit einer Methode eines der anderen vorgestellten Optimierungs–Systeme zu ergänzen:

- Die hybride Methode von Lin und Lee bestimmt in mehreren Schritten für jede Regel die optimale Konklusion. Dabei werden mit Hilfe des Backpropagation–Algorithmus mehrere zusätzliche Ausgabe–Partitions–Mengen erzeugt, die nicht unbedingt benötigt werden, bis schließlich die optimale Konklusion gefunden wird.

  Mit der MFOS–M–Methode zum Korrigieren von Regeln wird sofort in einem Schritt eine geeignete Konklusion bestimmt. Bei Bedarf wird genau die benötigte Ausgabe–Partitions–Menge erzeugt. Abschließend erfolgt ein Fine–Tuning der Partitions–Fuzzy–Mengen mit dem Backpropagation–Algorithmus. Mit der hybriden Methode von Lin und Lee ist daher weder eine Erhöhung der Effektivität noch ein geringerer Fehler zu erwarten. Deshalb ist eine (prinzipiell mögliche) Übertragung dieser Methode auf das MFOS–M–System nicht sinnvoll.

- Die NEFCON–Methode zur Erzeugung einer Regelbasis bildet alle Regeln, die sich durch Kombination vorhandener Eingabe– und Ausgabe–Partitions–Mengen darstellen lassen. Anschließend werden "schlechte" und nicht benötigte Regeln entfernt. Diese Vorgehensweise ist grundsätzlich auch mit dem MFOS–M–System möglich. Zur Bewertung der Regeln wird dazu ein anderes Kriterium benötigt, da die Ausgabe beim MFOS–M–System nicht normiert sein muß. Hier bietet sich der von den Regeln verursachte Fehler als Kriterium an.

Da sich bereits mit der MFOS–M–Methode zum Erzeugen von Regeln alle benötigten Regeln erzeugen lassen, besteht grundsätzlich kein Bedarf an einem alternativen Verfahren. Mit der MFOS–M–Methode werden direkt in einem Schritt korrekte Regeln erzeugt, so daß die NEFCON–Methode keine größere Effektivität erwarten läßt. Daher scheint eine Übertragung dieser Methode auf das MFOS–M–System nicht sinnvoll.

- Der NEFCON–Lernalgorithmus 1 adaptiert die Partitions–Fuzzy–Mengen mit dem Ziel, den Einfluß "guter" Regeln zu vergrößern und den Einfluß "schlechter" Regeln zu verringern. Hierzu soll der Erfüllungsgrad der Prämissen "guter" Regeln und der Betrag ihres Beitrags zum Ergebnis vergrößert werden. Für "schlechte" Regeln soll das Gegenteil erreicht werden.

  Eine Übertragung dieser Vorgehensweise auf das MFOS–M–System erfordert einige Modifikationen. Zur Bewertung einer Regel muß ein alternatives Kriterium gewählt werden, z.B. der von der Regel verursachte Fehler. Eine Erhöhung des Erfüllungsgrades der Prämissen "guter" Regeln und eine Verringerung des Erfüllungsgrades der Prämissen "schlechter" Regeln ist in jedem Fall sinnvoll, da der Erfüllungsgrad der Prämisse den Einfluß einer Regel auf das Ergebnis steuert. Zur Anpassung der Ausgabe–Partitions–Mengen wäre die zusätzliche Berücksichtigung der korrekten Ausgabe wünschenswert.

  Der Nutzen einer Übertragung dieser Methode auf das MFOS–M–System ist fraglich. Mit dem Backpropagation–Algorithmus ist bereits eine Optimierung der Partitions–Fuzzy–Mengen möglich. Insbesondere werden im Gegensatz zur NEFCON–Methode auch die Modalwerte und somit die Positionen der Partitions–Fuzzy–Mengen angepaßt, wodurch eine wesentlich effektivere Einstellung gewährleistet wird. Daher ist durch die NEFCON–Methode weder eine höhere Effektivität noch ein geringerer Fehler zu erwarten, so daß eine Übertragung dieser Methode auf das MFOS–M–System nicht sinnvoll erscheint.

Teil III

Optimierungsmethoden für Sugeno–Controller

Kapitel 8

Das ANFIS–System

In der Literatur werden häufig Sugeno–Controller mit Hilfe von neuronalen Netzen simuliert und optimiert. Die Übertragung eines Sugeno–Controllers auf ein funktional äquivalentes neuronales Netz gestaltet sich einfacher als die Übetragung eines Mamdani–Controllers, da bei Sugeno–Controllern die Defuzzifizierung entfällt. Zudem gibt es natürliche Gemeinsamkeiten zwischen Sugeno–Controllern und sogenannten *RBF–Netzen* (Radiale–Basisfunktionen–Netze). Zur Beschreibung von RBF–Netzen und deren Vergleich mit Sugeno–Controllern s. [Zell, 1996] und [WuTam, 1999]. Stellvertretend für Optimierungs–Systeme für Sugeno–Controller soll in diesem Kapitel das ANFIS–System vorgestellt werden.

Das ANFIS–System (Adpative–Network–based Fuzzy Inference System, [Jang, 1993]) ist ein universelles Modell eines neuronalen Netzes zur Modellierung und Optimierung eines Fuzzy–Controllers. Im Gegensatz zum Verfahren von Lin und Lee (s. Kapitel 4) und zum NEFCON–Modell (s. Kapitel 5) wird hier ein Sugeno–Controller verwendet.

8.1 Aufbau des Systems

Zur Vereinfachung der Konstruktion wird vorausgesetzt, daß sich jede Regel nur auf eine Ausgabe–Dimension bezieht. Aufgrund äquivalenter Umformungsmöglichkeiten stellt dies keine Einschränkung dar (vgl. Kapitel 3). Eine Regel, die zwei Ausgabe–Dimensionen verwendet, wird durch zwei Regeln, die jeweils eine Ausgabe–Dimension verwenden, ersetzt. Ein ANFIS–Netz besteht aus fünf Schichten. In Schicht 1 gibt es ein Neuron für jede Eingabe–Partitions–Menge. Schicht 2, Schicht 3 und Schicht 4 enthalten jeweils für jede Regel ein Neuron. Jedes Neuron in Schicht 2 ist mit genau den Neuronen aus Schicht 1 verbunden, die bei der Prämisse der zugehörigen Regel verwendet werden. So wird die Prämisse der Regeln in diesen Verbindungen gespeichert. Schicht 3 und Schicht 2 sind total verbunden. Jedes Neuron aus Schicht 4 ist genau mit dem Neuron aus Schicht 3 verbunden, das dieselbe Regel repräsentiert. Jedes Ausgabe–Neuron (Schicht 5) ist mit allen Neuronen aus Schicht 4 verbunden, deren zugehörige Regel sich auf die entsprechende Ausgabe–Dimension bezieht.

Beispiel 8.1 *Abbildung 8.1 zeigt ein ANFIS–System für folgende Regeln:*

R_1 : *IF* $x_1 = \widetilde{A}_{11}$ *UND* $x_2 = \widetilde{A}_{21}$ *THEN* $y = c_1$
R_2 : *IF* $x_1 = \widetilde{A}_{12}$ *UND* $x_2 = \widetilde{A}_{22}$ *THEN* $y = c_2$
R_3 : *IF* $x_1 = \widetilde{A}_{13}$ *UND* $x_2 = \widetilde{A}_{22}$ *THEN* $y = c_3$

mit $c_i \in \mathbb{R}$ *den reellen Konklusionen.*

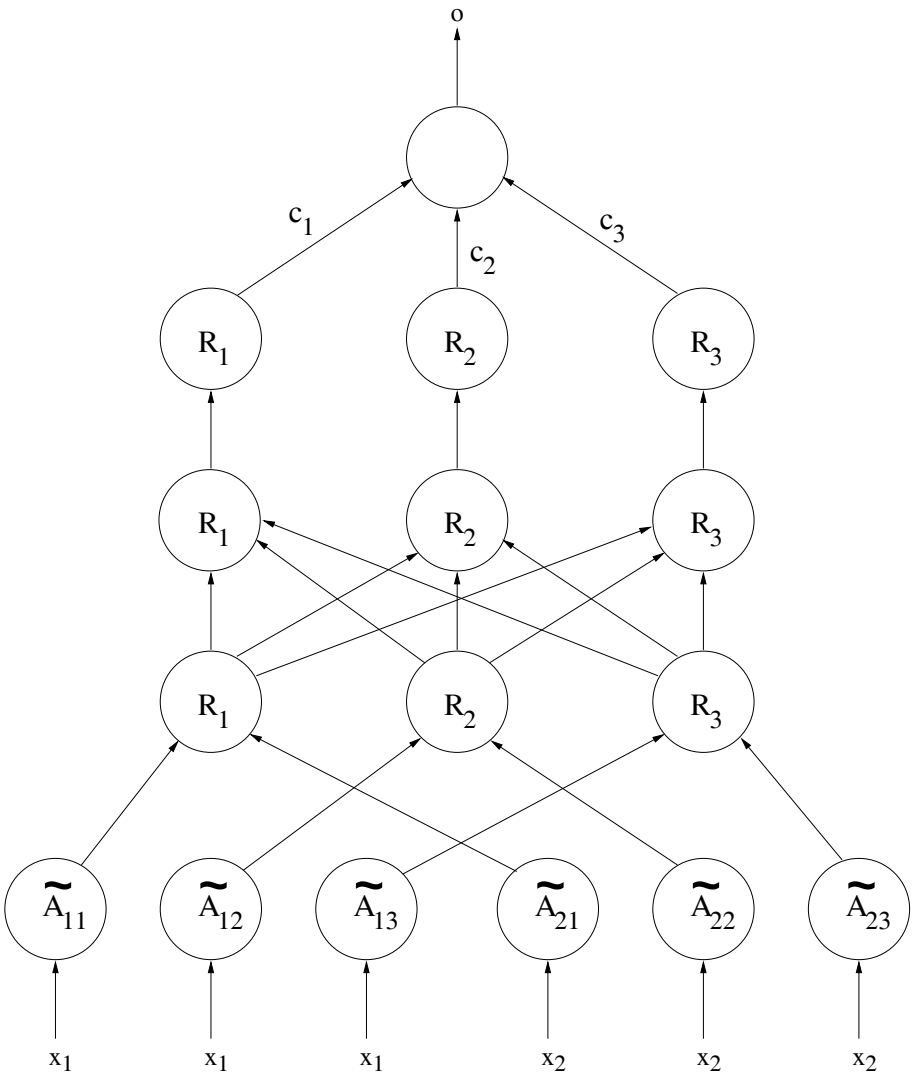

Abbildung 8.1: Aufbau eines ANFIS–Systems

Jedes Neuron in Schicht 1 repräsentiert eine Eingabe–Partitions–Menge. Als Parameter werden die Werte dieser Fuzzy–Menge verwendet, z.B. (m, w) (Modalweit und Weite) bei Gauß–Mengen. Als Ausgabe wird der Zugehörigkeitsgrad der Eingabe zu dieser Menge berechnet. Jedes Neuron aus Schicht 2 berechnet eine T–Norm seiner Eingaben, z.B. das Produkt der Eingaben. Damit entspricht die Ausgabe dieser Neuronen

dem Erfüllungsgrad E_k der Prämisse der zugehörigen Regel R_k. In Schicht 3 berechnet jedes Neuron $N_{3,k}$ den *gemittelten Erfüllungsgrad* ME_k von Regel R_k.

Definition 8.1 *Sei die Regelbasis eines Fuzzy–Controllers gegeben. Der gemittelte Erfüllungsgrad einer Regel R_k: ME_k wird berechnet gemäß:*

$$ME_k = \frac{E_k}{\sum\limits_{R_l \in Reg(j)} E_l}$$

mit $Reg(j)$ der Menge aller Regeln, deren Konklusion sich auf dieselbe Ausgabe–Dimension Y_j bezieht wie Regel R_k.

In Schicht 4 berechnet jedes Neuron das Produkt seiner Eingabe ME_k mit der Konklusion der zugehörigen Regel c_k (reell) bzw. f_k, falls die Konklusion eine lineare Funktion ist (vgl. Abschnitt 3.12). Die Ausgabe dieser Neuronen ist somit der gemittelte Anteil der zugehörigen Regel am Ergebnis. In Schicht 5 berechnet jedes Neuron die Summe seiner Eingaben, um die Netzausgabe zu erhalten. Auf diese Weise liefert das ANFIS–Netz zu den selben Eingabe–Werten dieselben Ausgabe–Werte, wie der modellierte Sugeno–Controller:

Die Erfüllungsgrade E_k der Prämissen werden offensichtlich korrekt berechnet. Für die Ausgabe des Sugeno–Controllers $(s_1, \ldots, s_m)$ gilt nach Definition (s. Abschnitt 3.12):

$$s_j = \frac{\sum\limits_{R_k \in Reg(j)} E_k \cdot f_k}{\sum\limits_{R_k \in Reg(j)} E_k}$$

somit gilt für die Netzsausgabe $(o_1, \ldots, o_m)$:

$$o_j = \sum_{R_k \in Reg(j)} \frac{E_k}{\sum\limits_{R_l \in Reg(j)} E_l} \cdot f_k = \sum_{R_k \in Reg(j)} \frac{E_k \cdot f_k}{\sum\limits_{R_l \in Reg(j)} E_l} = \frac{\sum\limits_{R_k \in Reg(j)} E_k \cdot f_k}{\sum\limits_{R_l \in Reg(j)} E_l} = s_j$$

D.h. das ANFIS–Netz führt exakt dieselben Berechnungen durch wie der Sugeno–Controller, lediglich die Reihenfolge ist verändert. Schicht 3 berechnet die Werte von $\frac{E_k}{\sum_{R_l \in Reg(j)} E_l}$, in Schicht 4 werden diese Werte mit den f_k multipliziert, Schicht 5 summiert diese Produkte auf. Demnach verhält sich das Netz so, wie der Fuzzy–Controller, den es repräsentiert. Zur Optimierung der berechneten Ausgaben werden Lernverfahren eingesetzt.

8.2 Lernverfahren und Anwendung

Für ANFIS–Netze ist ausschließlich ein Gradientenabstiegsverfahren und eine *lineare Methode* als Lernverfahren vorgesehen. Das heißt, es werden lediglich die Parameter der Fuzzy–Mengen auf dem Eingaberaum und die reellen Konklusionen angepaßt. Fehlerfunktion F für den Gradientenabstieg ist der mittlere quadratische Fehler. Die Anpassung der Parameter m_{i*} (Modalwert), w_{i*} (Weite) der Fuzzy–Mengen auf dem Eingaberaum und der Regel–Konklusionen c_k erfolgt somit gemäß:

$$\Delta m_{i*} = -\eta \cdot \frac{\partial F}{\partial m_{i*}}$$

$$\Delta w_{i*} = -\eta \cdot \frac{\partial F}{\partial w_{i*}}$$

$$\Delta c_{i*} = -\eta \cdot \frac{\partial F}{\partial c_{i*}}$$

mit $\eta > 0$ einer Lernrate.

Alternativ wird ein hybrides Verfahren aus dem Gradientenabstieg und einem linearem Lernverfahren eingesetzt: werden die Parameter der Fuzzy–Mengen auf dem Eingaberaum als fest vorausgesetzt, läst sich die Netzausgabe als Linearkombination der Regel–Konklusionen c_k und der gemittelten Erfüllungsgrade der Regeln ME_k (Ausgaben Schicht 3) darstellen:

$$o_j = \sum_{R_k \in Reg(j)} ME_k \cdot c_k$$

Unter dieser Voraussetzung sind für jedes Trainingsbeispiel die einmal berechneten Werte ME_k fest. Somit ist zur Bestimmung der optimalen Werte der Regel–Konklusionen c_k ein lineares Gleichungssystem zu lösen. Analog zum linearen Assoziierer (vgl. Abschnitt 2.5) wird der Fehler mit Hilfe der Pseudo–Inversen minimiert. Nach Bestimmung der optimalen Werte der Regel–Konklusionen c_k bei gegebenen Werten ME_k werden die Fuzzy–Mengen auf dem Eingaberaum angepaßt. Anschließend werden die neuen Werte ME_k bestimmt und ein neues lineares Gleichungssystem erstellt. Das hybride Verfahren ist ein online–Training mit zwei abwechselnden Phasen:

1. Forward–Pass: Anpassung der Werte von c_k mit Hilfe der Pseudo–Inversen

2. Backward–Pass: Anpassung der Werte von m_{i*} und w_{i_*} mit Hilfe des Gradientenabstiegs

Da im Forward–Pass die Werte der c_k für die jeweils aktuellen Werte von m_{i*} und w_{i*} sofort in einem Schritt optimiert werden, ermöglicht das hybride Verfahren ein schnellere Lernen als das reine Gradientenabstiegsverfahren.

Ein entscheidender Nachteil des ANFIS–Systems ist, daß ausschließlich die Parameter der vorhandenen Fuzzy–Mengen und der Regel–Konklusionen optimiert werden. Ein Erzeugen oder Korrigieren von Regeln ist nicht vorgesehen. Falls eine notwendige Regel bei der Initialisierung eines ANFIS–Netzes nicht berücksichtigt wurde, ist somit eine optimale Anpassung des Systems nicht möglich. Gleiches gilt, falls zu wenig Eingabe–Partitions–Mengen definiert wurden, da auch das Erzeugen neuer Fuzzy–Mengen nicht durchgeführt wird.

Für den erfolgreichen Einsatz eines ANFIS–Netzes muß die Anzahl der Partitions–Mengen und Regeln vorher bestimmt werden. Eine Möglichkeit ist, nach Festlegung der Eingabe–Partitionierungen *jede* Kombination der Eingabe–Fuzzy–Mengen als Regel–Prämisse zu verwenden. Bei drei Eingabe–Dimensionen mit jeweils sieben Mengen ergibt dies bereits $7 \cdot 7 \cdot 7 = 343$ mögliche Kombinationen. Die meisten davon stellen allerdings Situationen dar, die bei der Anwendung nicht auftreten, so daß auf diese Weise viele unnötige Regeln erzeugt werden. Die Autoren empfehlen in [Jang, 1993] die geeignete Struktur des Netzes (Anzahl Neuronen Schicht 1 = Anzahl Eingabe–Partitions–Mengen, Anzahl Neuronen Schicht 2, 3 und 4 = Anzahl Regeln) durch ausprobieren herauszufinden. Dies entspricht der Vorgehensweise beim Einsatz von Standard–Multilayer–Perzeptrons. Hier wird versucht, basierend auf Erfahrungswerten, durch ausprobieren eine hinreichend gute Netzstruktur zu finden.

Ein klassisches MLP ist jedoch nicht so stark abhängig von der Anzahl der Neuronen, d.h. der Anwender hat mehr Spielraum. Grund hierfür ist der Aufbau der Neuronen eines MLP: jedes verborgene Neuron berechnet die gewichtete Summe seiner Eingaben und setzt diesen Wert in eine sigmoide Funktion ein (s. Abschnitt 2.6). Falls ein verborgenes Neuron entfernt wird, fehlt bei den gewichteten Summen der nächsten Schicht jeweils ein Summand. Dies wird durch entsprechend angepaßte Gewichtswerte weitestgehend ausgeglichen, so daß eine hinreichende Fehlerminimierung immer noch möglich ist. Konsequenz ist, daß die Anzahl der Schichten und verborgenen Neuronen beim MLP relativ unkritisch ist.

Bei einem ANFIS–Netz verhält sich dies anders: jedes Neuron in Schicht 1 repräsentiert eine Eingabe–Partitions–Menge. Die Partitionierungen teilen die Eingabe–Dimensionen in verschiedene Abschnitte auf, in denen unterschiedliche Ausgaben gewünscht sind. Falls in einem Bereich, der von einer einzigen Fuzzy–Menge überdeckt wird, verschiedene Ausgaben korrekt sind, ist dies mit nur einer Fuzzy–Mengen nicht zu erreichen. In den Schichten 2 bis 4 repräsentiert jedes Neuron eine Regel. Um die richtigen Ausgaben zu berechnen, muß für jede praktisch mögliche Situation die korrekte Regel

vorhanden sein. Eine *andere* Regel wird in einer *anderen* Situation aktiv. Somit ist es beim ANFIS–Netz nicht möglich, analog zum MLP das Fehlen einzelner verborgener Neuronen durch Adaption der Gewichts–Werte zu kompensieren.

Eine genauere Bewertung dieses Verfahrens sowie ein Vergleich zu den anderen vorgestellten Optimierungs–Systemen und ein weiteres Beispiel zu diesem Verfahren (Beispiel 10.1) werden in Kapitel 7 vorgestellt.

Kapitel 9

MFOS für Sugeno–Controller

Das in Kapitel 6 vorgestellte MFOS–M–System ist ausschließlich zur Repräsentation und Optimierung eines Fuzzy–Controllers nach Mamdani vorgesehen. In diesem Kapitel wird ein alternatives MFOS–System zur Repräsentation und Optimierung eines Sugeno–Controllers (MFOS–S) entwickelt. Da ein Sugeno–Controller keine Ausgabe–Partitions–Mengen verwendet, muß das verwendete neuronale Netz anders aufgebaut sein als ein MFOS–M–Netz. Insbesondere müssen aus dem gleichen Grund die einzelnen Lernverfahren modifiziert werden.

9.1 Aufbau und Arbeitsweise des MFOS–S–Netzes

Das MFOS–System für Sugeno–Controller (MFOS–S) soll die gleichen Einsatz– und Optimierungsmöglichkeiten wie das MFOS–M–System bieten. Daher wurden dieselben Voraussetzungen an den Sugeno–Controller gemacht, wie an den Fuzzy–Controller für den Einsatz des MFOS–M–Systems (vgl. Kapitel 6.1):

- UND–Verknüpfung der Prämisse
- eine Ausgabe–Dimension pro Regel
- Singleton–Fuzzifizierer
- Gauß– und Dreiecks–Mengen
- Minimum Implikation
- (Defuzzifizierung entfällt bei Sugeno–Controllern)

Im Gegensatz zum vierschichtigen MFOS–M–Netz ist das MFOS–S–Netz für Sugeno–Controller dreischichtig (s. Abbildung 9.1). Da die Prämissen der Regeln und die Berechnung der Erfüllungsgrade der Prämissen bei Fuzzy–Controllern nach Mamdani und Sugeno–Controllern exakt gleich sind (s. Kapitel 3), werden die Schichten 1 und 2

direkt vom MFOS–M–Netz übernommen. Beim MFOS–M–Netz enthält Schicht 3 für jede Ausgabe–Partitions–Menge ein Neuron (s. Kapitel 6.2); diese Schicht entfällt beim MFOS–S–Netz. Stattdessen ist nun Schicht 3 die Ausgabeschicht, jedoch mit geänderten Aktivierungs– und Ausgabe–Funktionen.

Definition 9.1 *Abbildung–3: SC $\longrightarrow$ MFOS–S ist die im Folgenden beschriebene Portierung eines Sugeno–Controllers auf das MFOS–S–System.*

Das Netz ist wie folgt aufgebaut:

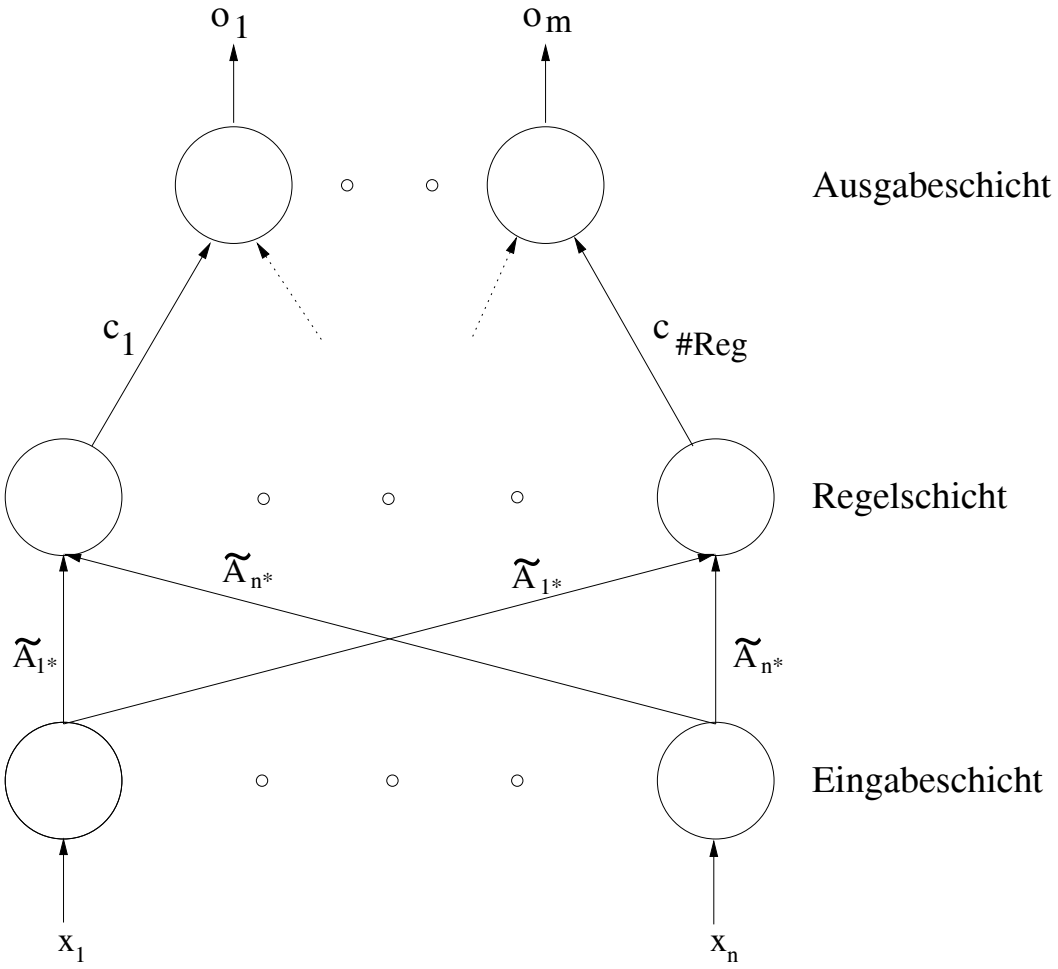

Abbildung 9.1: Aufbau eines MFOS–S–Netzes

In der Eingabeschicht gibt es für jede Eingabe–Dimension ein Neuron, welches nur seine Eingabe weiterleitet.

In Schicht 2, der *Regelschicht*, gibt es für jede Regel R_k ein Neuron, welches ebenfalls mit R_k bezeichnet wird. Dieses ist mit allen Neuronen aus Schicht 1 verbunden, deren

zugehörige Eingaben bei dieser Regel verwendet werden. Somit lassen sich auch linguistische Regeln einsetzen, die nicht alle Eingabe–Werte berücksichtigen. Als Gewicht wird für jede Verbindung die Fuzzy–Menge genommen, die den entsprechenden linguistischen Term aus der Prämisse von Regel R_k für die zugehörige Eingabe–Dimension repräsentiert.

Jedes Neuron R_k in Schicht 2 berechnet den Erfüllungsgrad der Prämisse von Regel R_k. Dazu werden zunächst die Zugehörigkeitsgrade der Eingabewerte zu den jeweiligen Fuzzy–Mengen der Verbindungen mit Schicht 1 berechnet. Anschließend werden diese Werte rekursiv mit einem UND–Operator zum Erfüllungsgrad verknüpft. Dieser ist dann die Ausgabe des Neurons.

In der Ausgabeschicht gibt es für jede Dimension des Ausgaberaumes ein Neuron. Dieses ist mit allen Neuronen aus Schicht 2 verbunden, deren zugehörige Regel sich auf diese Ausgabe–Dimension bezieht. Als Gewicht wird jeweils die reelle Konklusion dieser Regel genommen. Jedes Neuron in der Ausgabeschicht berechnet den Stellwert für seine Ausgabe–Dimension gemäß folgender Formeln:

Aktivitätsfunktion von Neuron j der Ausgabeschicht ist:

$$f_{a\_3,j} = \sum_{k \in V_2(j)} o_{2,k} \cdot c_k$$

Ausgabefunktion von Neuron j der Ausgabeschicht ist:

$$f_{o\_3,j} = \frac{f_{a\_3,j}}{\sum_{k \in V_2(j)} o_{2,k}}$$

mit

- $V_2(j)$ der Menge der Verbindungen zwischen Neuron j der Ausgabeschicht und den Neuronen aus Schicht 2
- $o_{2,k}$ der Ausgabe von Neuron k aus Schicht 2
- c_k der Konklusion von Regel R_k
  (Regel R_k wird durch Neuron k aus Schicht 2 repräsentiert)

Die Struktur des Netzes repräsentiert somit die Regelbasis des Fuzzy–Controllers vollständig. In den Gewichten werden die Eingabe–Partitionierungen und Regel–Konklusionen ebenfalls vollständig gespeichert. D.h. bei der Übertragung eines Sugeno–Controllers auf das MFOS–S–Netz gehen keine Informationen verloren.

Beispiel 9.1 *Hat Regel R_4 die Gestalt*

IF $x_1 = \widetilde{A}_{13}$ *UND* $x_2 = \widetilde{A}_{21}$ *THEN* $y_1 = c_4$

so hat das Neuron R_4 aus Schicht 2 folgende Verbindungen (s. Abb. 9.2):

- *es gibt je eine Verbindung mit Neuron 1 und Neuron 2 aus Schicht 1 mit den Eingabe–Partitions–Mengen $\widetilde{A}_{13}$ bzw. $\widetilde{A}_{21}$ als Gewicht*
- *es gibt eine Verbindung mit Ausgabe–Neuron 1, die als Gewicht c_4 erhält*

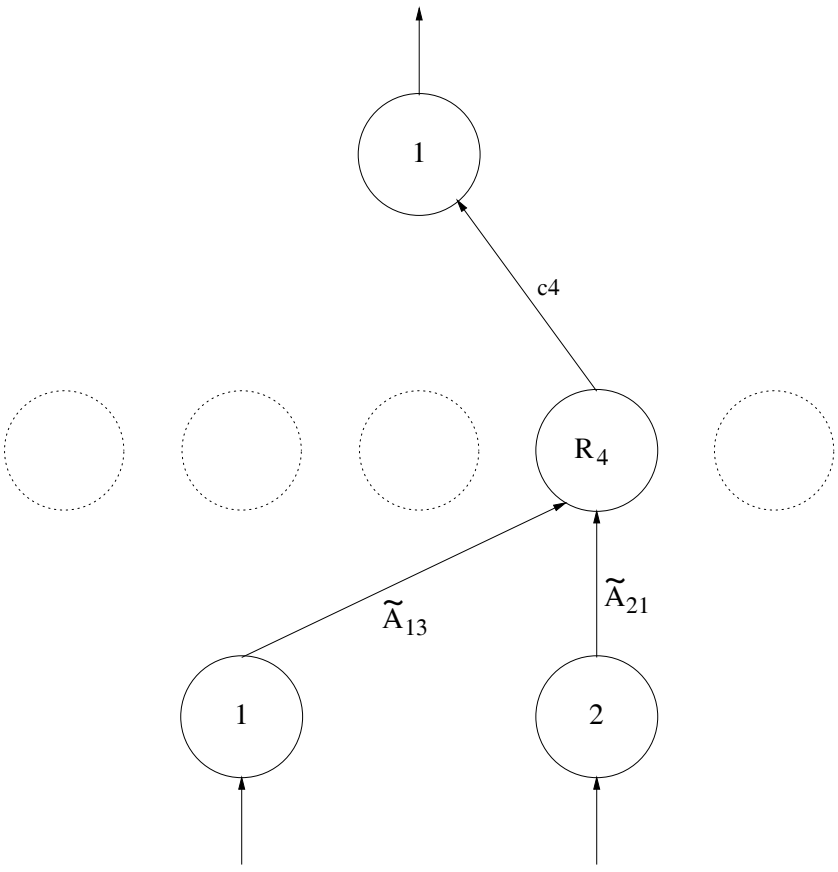

Abbildung 9.2: Regel 4 repräsentierende Verbindungen

Das Netz verhält sich nun so, wie der Sugeno–Controller, den es repräsentiert. D.h. es werden exakt dieselben Berechnungen durchgeführt, wie im ursprünglichen Sugeno–Controller. Durch das MFOS–S–Netz steht somit eine Methode zur Verfügung, einen potentiell beliebigen Sugeno–Controller auf ein funktional äquivalentes neuronales Netz zu portieren. Der folgende Satz zeigt, daß diese Portierung (Abbildung–3) korrekt ist:

Satz 9.1
Abbildung–3 ist korrekt, d.h. das Ein– Ausgabe–Verhalten eines Sugeno–Controllers mit den o.g. Spezifikationen (s. Abschnitt 9.1) und das Ein– Ausgabe–Verhalten des zugehörigen MFOS–S–Netzes sind identisch.

Beweis 9.1
Sei SC ein Sugeno–Controller, der die o.g. Spezifikationen erfüllt.
Seien $R_1, \ldots, R_{\#Reg}$ *die Regeln von SC und jeweils*
$\widetilde{A}_{1*}, \ldots, \widetilde{A}_{n*}$ *die Eingabe–Partitions–Mengen von Eingabe–Dimension 1 bis* n.

Regel R_k $(k = 1, \ldots, \#Reg)$ *hat die Gestalt:*

`IF` $x_1 = \widetilde{A}_{1*}$ `UND ... UND` $x_n = \widetilde{A}_{n*}$ `THEN` $y_j = c_k$

mit $\widetilde{A}_{i*}$ *einer Partitions–Menge aus Eingabe–Dimension* i *und* $c_k \in \mathbb{R}$ *der Konklusion von Regel* R_k.

Sei NN das mit Abbildung–3 zu SC generierte MFOS–S–Netz.

Seien $r_1, \ldots, r_n \in \mathbb{R}^n$ *die Eingaben von SC bzw. NN, seien* $s_1, \ldots, s_m$ *die berechneten Ausgaben von SC und* $o_{3,1}, \ldots, o_{3,m}$ *die berechneten Ausgaben von NN zu den Eingaben* $r_1, \ldots, r_n$.

Zu zeigen ist: $o_{3,j} = s_j$ *für* $j = 1, \ldots, m$.

Das zugehörige MFOS–S–Netz NN enthält nach Definition zu jeder Regel R_k *ein gleichnamiges Neuron in Schicht 2. Die Gewichte der Verbindungen zu Schicht 1 sind jeweils dieselben Fuzzy–Mengen, die in der Prämisse der Regeln verwendet werden.*

Im Korrektheitsbeweis zu Abbildung–1 (s. [Tenhagen, 2000]) wurde bereits gezeigt, daß die berechnete Ausgabe von jedem Neuron R_k *aus Schicht 2 bei einem MFOS–M–Netz gleich dem Erfüllungsgrad von Regel* R_k *des zugehörigen Fuzzy–Controllers nach Mamdani ist.*

Da

1. *die Erfüllungsgrade der Prämissen der Regeln bei einem Sugeno–Controller auf dieselbe Weise berechnet werden wie bei einem Fuzzy–Controller nach Mamdani und*

2. *Schicht 1 und Schicht 2 beim MFOS–M–Netz und beim MFOS–S–Netz identisch sind,*

folgt:

die berechnete Ausgabe von jedem Neuron R_k aus Schicht 2 ist beim MFOS–S–Netz gleich dem Erfüllungsgrad von Regel R_k des zugehörigen Sugeno–Controllers. D.h. es gilt für jedes R_k aus Schicht 2:

$$o_{2,k} = E_k$$

mit $o_{2,k}$ der berechneten Ausgabe von Neuron k und E_k dem Erfüllungsgrad der Prämisse von Regel R_k.

Nach Definition wird die Ausgabe s_j, $j = 1, \ldots, m$, eines Sugeno–Controllers wie folgt berechnet (vgl. Kapitel 3.12):

$$s_j = \frac{\sum\limits_{k \in V_2(j)} E_k \cdot c_k}{\sum\limits_{k \in V_2(j)} E_k}$$

Daraus folgt für die berechnete Netzausgabe $o_{3,j}$, $j = 1, \ldots, m$:

$$o_{3,j} = f_{o\_3,j}(f_{a\_3,j}) = \frac{f_{a\_3,j}}{\sum\limits_{k \in V_2(j)} o_{2,k}} = \frac{\sum\limits_{k \in V_2(j)} o_{2,k} \cdot c_k}{\sum\limits_{k \in V_2(j)} o_{2,k}} = \frac{\sum\limits_{k \in V_2(j)} E_k \cdot c_k}{\sum\limits_{k \in V_2(j)} E_k} = s_j$$

□

Mit dem MFOS–S–System ist es somit möglich, einen gegebenen Sugeno–Controller auf ein funktional äquivalentes neuronales Netz zu übertragen. Wie beim MFOS–M–Netz sollen verschiedene Lernverfahren den Sugeno–Controller optimieren. Nach dem Training ist eine Rück–Portierung des MFOS–S–Netzes auf einen gewöhnlichen Sugeno–Controller möglich. Diese Abbildung wird im Anschluß an die Vorstellung der eingesetzten Lernverfahren in Abschnitt 9.3 beschrieben. Zunächst werden im folgenden Abschnitt die eingesetzten Lernverfahren vorgestellt.

9.2 Die Lernverfahren des MFOS–S–Systems

Das MFOS–M–System stellt diverse Lernverfahren zur Verfügung, um die einzelnen Bestandteile einen Fuzzy–Controller nach Mamdani zu optimieren (s. Kapitel 6). In diesem Abschnitt sollen einzelne dieser Verfahren auf das MFOS–S–System übertragen werden.

Dies ist nicht ohne einige Modifikationen möglich, da beim Sugeno–Controller —und somit beim MFOS–S–System— prinzipiell keine Ausgabe–Partitions–Mengen und keine Schnitthöhen dieser Mengen zur Verfügung stehen. Damit entfällt ein wichtiges Kriterium zur Bewertung von Regeln bzw. Konklusionen, welches bei einigen Verfahren des MFOS–M–Systems verwendet wird. Daher müssen für die Lernverfahren des MFOS–S–Systems Alternativen gefunden werden. Grundsätzlich sind mit dem MFOS–S–System dieselben Modifikationen möglich wie mit dem MFOS–M–System:

- Modifikation bestehender Regeln
- Erzeugen neuer Regeln
- Löschen vorhandener Regeln
- Modifikation bestehender Fuzzy–Mengen
- Erzeugen neuer Fuzzy–Mengen
- Löschen vorhandener Fuzzy–Mengen

Wie das MFOS–M–System bietet das MFOS–S–System die Möglichkeit, alle dieser Modifikationen durchzuführen. Dabei bleibt die Entscheidung, welche Modifikationen in welcher Reihenfolge angewendet werden, dem User überlassen. Sämtliche Verfahren basieren auf der Repräsentation von Trainingsbeispielen.

Korrigieren von Regeln:

Die Vorüberlegungen zur Beurteilung einer Regel sind dieselben wie beim MFOS–M–System (s. Kapitel 6.3.1). Jedoch muß das Kriterium zur Bewertung des Fehlers und zur Bestimmung der korrekten Konklusion modifiziert werden.

1. Regeln, die alle Eingabe–Variablen verwenden:

Hat die Prämisse einer Regel, die alle Eingabe–Variablen verwendet, einen hohen Erfüllungsgrad, so ist das zu steuernde System zu einem genau so hohen Grad in dem Zustand, den diese Regel repräsentiert. Die Regel mit dem höchsten Erfüllungsgrad der Prämisse beschreibt die entsprechende Situation am besten, andere Regeln sind für

andere Situationen vorgesehen. Deshalb werden für jedes Trainingsbeispiel nur die Regeln mit dem höchsten Erfüllungsgrad der Prämisse überprüft. Aufgrund der Form der Regeln (eine Ausgabe–Dimension pro Regel, s. Abschnitt 9.1) können mehrere Regeln die gleiche Prämisse und somit den gleichen Erfüllungsgrad haben. Von den Regeln mit maximalem Erfüllungsgrad der Prämisse wird jede separat getestet.

Falls die reelle Konklusion einer solchen Regel eine hohe Abweichung von der gewünschten Ausgabe hat, ist die Konklusion falsch. Daher wird als neue Konklusion die korrekte Ausgabe des aktuellen Trainingsbeispiels eingesetzt. Zur Durchführung dieses Verfahrens werden für jede Regel, die alle Eingabe–Variablen verwendet, folgende Schritte ausgeführt:

- bestimme das Trainingsbeispiel, welches den maximalen Erfüllungsgrad bewirkt
- falls der maximale Erfüllungsgrad über einer Schranke liegt: berechne die Abweichung der Regel–Konklusion von der gewünschten Ausgabe des gewählten Trainingsbeispiels
- falls die Abweichung über einer Schranke liegt: wähle als neue Konklusion die korrekte Ausgabe des gewählten Trainingsbeispiels

Bemerkung 9.1 *Diese Vorgehensweise entspricht dem Verfahren zum Korrigieren von Regeln des MFOS–M–Systems, da bei einem Sugeno–Controller das einzeln mit einer Regel berechnete Ergebnis immer der Regel–Konklusion entspricht: sei*

R_k: `IF` $x_1 = \widetilde{A}_{1*}$ `UND` ... `UND` $x_n = \widetilde{A}_{n*}$ `THEN` $y_j = c_k$

eine Regel eines Sugeno–Controllers mit Erfüllungsgrad $E_k > 0$. Dann gilt für die nur mit dieser Regel berechnete Ausgabe s_j entsprechend der Definition eines Sugeno–Controllers:

$$s_j = \frac{E_k \cdot c_k}{E_k} = c_k$$

Beispiel 9.2 *Die folgenden Fuzzy–Mengen und Regeln ergeben die Partitionierungen und die Regelbasis für einen einfachen Sugeno–Controller zur Steuerung eines Heizgeräts:*

Eingabe (Temperatur in °C): Dreiecks–Mengen auf dem Grundraum $[13, 23]$*:*

$$\begin{aligned} \texttt{sehr kalt} &\mathrel{\widehat{=}} \widetilde{A}_1 = (13, 15, 17) \\ \texttt{kalt} &\mathrel{\widehat{=}} \widetilde{A}_2 = (16, 18, 20) \\ \texttt{warm} &\mathrel{\widehat{=}} \widetilde{A}_3 = (19, 21, 23) \end{aligned}$$

Die linguistischen Variablen sind x für die Temperatur und y für die Heizleistung.

Die richtigen Regeln sind:

$$\begin{aligned} &\textit{IF } x = \texttt{sehr kalt } \textit{THEN } y = 8 \\ &\textit{IF } x = \texttt{kalt } \textit{THEN } y = 5 \\ &\textit{IF } x = \texttt{warm } \textit{THEN } y = 2 \end{aligned}$$

Durch versehentliches Vertauschen der reellen Konklusionen 8 und 2 ergeben sich folgende Regeln:

$$\begin{aligned} &\textit{IF } x = \texttt{sehr kalt } \textit{THEN } y = 2 \\ &\textit{IF } x = \texttt{kalt } \textit{THEN } y = 5 \\ &\textit{IF } x = \texttt{warm } \textit{THEN } y = 8 \end{aligned}$$

Das oben beschriebene Verfahren erkennt diese Fehler und korrigiert sie, so daß genau die richtigen Regeln wieder hergestellt werden.

2. Regeln, die nicht alle Eingabe–Variablen verwenden:

Falls die reelle Konklusion einer Regel, die nicht alle Eingabe–Variablen verwendet, bei *jedem* Trainingsbeispiel, das einen hohen Erfüllungsgrad der Prämisse bewirkt, eine hohe Abweichung von der gewünschten Ausgabe hat, dann liegt die Konklusion dieser Regel in einem falschen Bereich. Als richtige Konklusion wird der Durchschnitt der korrekten Ausgaben von Trainingsbeispielen, die einen Erfüllungsgrad der Prämisse dieser Regel über einer Schranke bewirken, gewählt. Zur Durchführung dieses Verfahrens werden für jede Regel, die nicht alle Eingabe–Variablen verwendet, folgende Schritte ausgeführt:

- gehe alle Trainingsbeispiele durch, die einen Erfüllungsgrad der Prämisse über einer Schranke bewirken

- berechne jeweils bei der zu überprüfenden Regel die Abweichung der Regel–Konklusion von der gewünschten Ausgabe des aktuellen Trainingsbeispiels
- falls die Abweichung jedesmal über einer Schranke liegt: bestimme die neue Konklusion
- neue Konklusion ist der Durchschnitt der korrekten Ausgaben von Trainingsbeispielen, die einen Erfüllungsgrad der Prämisse dieser Regel über einer Schranke bewirken

Beispiel 9.3 *Die folgenden Fuzzy–Mengen und Regeln ergeben die Partitionierungen und die Regelbasis für einen einfachen Sugeno–Controller zur Steuerung eines Heizgeräts. Im Gegensatz zu Beispiel 9.2 wird berücksichtigt, ob es Nacht ist oder nicht (codiert durch "1" für Tag und "3" für Nacht). Dabei gilt, daß nachts grundsätzlich nur mit geringer Leistung geheizt werden soll:*

Eingabe 1 (Temperatur in °C): Dreiecks–Mengen auf dem Grundraum $[13, 23]$*:*

$$\begin{aligned}
\texttt{sehr kalt} &\mathrel{\widehat{=}} \widetilde{A}_{11} = (13, 15, 17) \\
\texttt{kalt} &\mathrel{\widehat{=}} \widetilde{A}_{12} = (16, 18, 20) \\
\texttt{warm} &\mathrel{\widehat{=}} \widetilde{A}_{13} = (19, 21, 23)
\end{aligned}$$

Eingabe 2 (Tag / Nacht): Dreiecks–Mengen auf dem Grundraum $[0, 4]$*:*

$$\begin{aligned}
\texttt{Tag} &\mathrel{\widehat{=}} \widetilde{A}_{21} = (0, 1, 2) \\
\texttt{Nacht} &\mathrel{\widehat{=}} \widetilde{A}_{22} = (2, 3, 4)
\end{aligned}$$

Die linguistischen Variablen sind x_1 *für die Temperatur,* x_2 *für* `Tag / Nacht` *und* y *für die Heizleistung.*

Die richtigen Regeln sind:

$$\begin{aligned}
&\texttt{IF } x_1 = \texttt{sehr kalt } \texttt{UND } x_2 = \texttt{Tag } \texttt{THEN } y = 8 \\
&\texttt{IF } x_1 = \texttt{kalt } \texttt{UND } x_2 = \texttt{Tag } \texttt{THEN } y = 5 \\
&\texttt{IF } x_1 = \texttt{warm } \texttt{UND } x_2 = \texttt{Tag } \texttt{THEN } y = 2 \\
&\texttt{IF } x_2 = \texttt{Nacht } \texttt{THEN } y = 2
\end{aligned}$$

Ist die letzte Regel versehentlich als

IF $x_2 =$ *Nacht THEN* $y = 8$

erzeugt worden, so erkennt das oben beschriebene Verfahren diesen Fehler und stellt die korrekte Regel wieder her.

Erzeugen von Regeln:

Das Verfahren zur Erzeugung neuer Regeln basiert auf folgender Heuristik: hat bei einem Trainingsbeispiel die Prämisse von *jeder* Regel einen geringen Erfüllungsgrad, dann gibt es keine passende Regel für diese Situation. Also muß eine neue Regel erzeugt werden. Für die Prämisse wird die Eingabe–Partitions–Menge bestimmt, die am besten die Eingabe–Werte repräsentiert. Als Konklusion wird die korrekte Ausgabe des Trainingsbeispiels gewählt. Zur Durchführung dieses Verfahrens werden für jedes Trainingsbeispiel die folgenden Schritte ausgeführt:

- berechne für jede Regel den Erfüllungsgrad der Prämisse
- ist der Erfüllungsgrad bei jeder Regel unterhalb einer Schranke, erzeuge eine neue Regel
  - bestimme für jeden Eingabe–Wert die Partitions–Menge aus der zugehörigen Eingabe–Dimension, die den höchsten Zugehörigkeitsgrad ergibt
  - die so bestimmten Eingabe–Partitions–Mengen ergeben die Prämisse der zu erzeugenden Regel
  - Konklusion der zu erzeugenden Regel ist die korrekte Ausgabe des aktuellen Trainingsbeispiels

Beispiel 9.4 *Die folgenden Fuzzy–Mengen und Regeln ergeben die Partitionierungen und die Regelbasis für einen einfachen Sugeno–Controller zur Steuerung eines Heizgeräts:*

Eingabe (Temperatur in °C): Dreiecks–Mengen auf dem Grundraum $[13, 23]$*:*

sehr kalt $\widehat{=} \widetilde{A}_1 = (13, 15, 17)$
kalt $\widehat{=} \widetilde{A}_2 = (16, 18, 20)$
warm $\widehat{=} \widetilde{A}_3 = (19, 21, 23)$

Die linguistischen Variablen sind x für die Temperatur und y für die Heizleistung.

Die richtigen Regeln sind:

IF $x =$ *sehr kalt THEN* $y = 8$
IF $x =$ *kalt THEN* $y = 5$
IF $x =$ *warm THEN* $y = 2$

Wird bei Erstellung der Regelbasis die Regel

IF $x =$ *sehr kalt THEN* $y = 8$

vergessen, so erzeugt das oben beschriebene Verfahren genau diese Regel, und die Regelbasis ist vervollständigt.

Löschen von Regeln:

Beim MFOS–M–System wird mit Hilfe der unterschiedlichen Schnitthöhen einer Ausgabe–Partitions–Menge bestimmt, welche Regeln niemals einen Einfluß auf das Ergebnis haben und daher gelöscht werden können (s. Kapitel 6.3.1). Beim MFOS–S–System entfällt dieses Kriterium. Stattdessen werden hier (neben dem Löschen eventuell vorhandener doppelter Regeln) zwei alternative Verfahren durchgeführt:

1. Verfahren:

Falls der Erfüllungsgrad der Prämisse einer Regel bei jedem Trainingsbeispiel unter einer Schranke liegt, ist der Einfluß dieser Regel so gering, daß sie entfernt werden kann. Zur Durchführung dieses Verfahrens werden für jede Regel die folgenden Schritte ausgeführt:

- bestimme für jedes Trainingsbeispiel den Erfüllungsgrad der Prämisse der aktuellen Regel
- falls der Erfüllungsgrad der Prämisse bei jedem Beispiel unter einer Schranke ist, lösche dieses Regel

Bemerkung 9.2 *Bei diesem Verfahren ist eine repräsentative Trainingsmenge, die jede mögliche Situation abdeckt, besonders wichtig. Falls für eine typische Situation kein Trainingsbeispiel vorhanden ist, wird eventuell eine Regel gelöscht, die nicht wegfallen darf. Die Verfügbarkeit geeigneter Trainingsbeispiele ist generell notwendig für die Anwendung von neuronalen Lernverfahren und nicht immer unproblematisch, vgl. z.B. [Rojas, 1993].*

Beispiel 9.5 *Seien die folgenden Fuzzy–Mengen und Regeln definiert:*

Eingabe 1: Dreiecks–Mengen auf dem Grundraum $[1, 11]$*:*

$\texttt{klein} \;\widehat{=}\; \widetilde{A}_{11} = (1, 3, 5)$
$\texttt{mittel} \;\widehat{=}\; \widetilde{A}_{12} = (4, 6, 8)$
$\texttt{groß} \;\widehat{=}\; \widetilde{A}_{13} = (7, 9, 11)$

Eingabe 2: Dreiecks–Mengen auf dem Grundraum $[2, 12]$*:*

$\texttt{niedrig} \;\widehat{=}\; \widetilde{A}_{21} = (2, 4, 6)$
$\texttt{hoch} \;\widehat{=}\; \widetilde{A}_{22} = (5, 7, 9)$
$\texttt{sehr hoch} \;\widehat{=}\; \widetilde{A}_{23} = (8, 10, 12)$

Die linguistischen Variablen sind x_1 *und* x_2 *für die Eingaben sowie* y *für die Ausgabe.*

Die richtigen Regeln sind:

$\texttt{IF}\ x_1 = \texttt{klein}\ \texttt{UND}\ x_2 = \texttt{niedrig}\ \texttt{THEN}\ y = 10$
$\texttt{IF}\ x_1 = \texttt{klein}\ \texttt{UND}\ x_2 = \texttt{hoch}\ \texttt{THEN}\ y = 10$
$\texttt{IF}\ x_1 = \texttt{mittel}\ \texttt{UND}\ x_2 = \texttt{niedrig}\ \texttt{THEN}\ y = 10$
$\texttt{IF}\ x_1 = \texttt{mittel}\ \texttt{UND}\ x_2 = \texttt{hoch}\ \texttt{THEN}\ y = 18$
$\texttt{IF}\ x_1 = \texttt{mittel}\ \texttt{UND}\ x_2 = \texttt{sehr hoch}\ \texttt{THEN}\ y = 18$
$\texttt{IF}\ x_1 = \texttt{groß}\ \texttt{UND}\ x_2 = \texttt{hoch}\ \texttt{THEN}\ y = 26$
$\texttt{IF}\ x_1 = \texttt{groß}\ \texttt{UND}\ x_2 = \texttt{sehr hoch}\ \texttt{THEN}\ y = 26$

Eingabewerte am entgegengesetzten Ende der Eingabe–Dimensionen (z.B. $x_1 = 3$ *und* $x_2 = 10$*) kommen nicht vor und gehören daher nicht zu den Trainingsbeispielen. Bei der Erstellung einer Regelbasis werden häufig aufgrund unzureichender Kenntnisse über mögliche Eingabewerte alle Kombinationen von Eingabe–Partitions–Mengen als Regel–Prämisse verwendet. Definiert ein Anwender bei diesem Beispiel zusätzlich folgende Regeln:*

$\texttt{IF}\ x_1 = \texttt{klein}\ \texttt{UND}\ x_2 = \texttt{sehr hoch}\ \texttt{THEN}\ y = 16$
$\texttt{IF}\ x_1 = \texttt{groß}\ \texttt{UND}\ x_2 = \texttt{niedrig}\ \texttt{THEN}\ y = 20$

so erkennt das oben beschrieben Verfahren, daß diese Regeln nutzlos sind und entfernt sie.

2. Verfahren:

Falls Regeln, die alle Eingabe–Variablen verwenden, einen Erfüllungsgrad der Prämisse über einer Schranke haben und die *Varianz* ihrer reellen Konklusionen gering ist, lassen sich diese Regeln zu einer Regel zusammenfassen. Die Prämisse wird von der Regel mit maximalem Erfüllungsgrad der Prämisse übernommen, Konklusion ist der Durchschnittswert der Konklusionen zusammengefaßter Regeln.

Regeln, die nicht alle Eingabe–Variablen berücksichtigen, lassen sich auf diese Weise nicht zusammenfassen, da sie je nach Trainingsbeispiel mit unterschiedlichen Regeln gemeinsam einen hohen Erfüllungsgrad der Prämisse besitzen.

Definition 9.2
Seien $c_1, \ldots, c_l$ die reellen Konklusionen (auf der selben Ausgabe–Dimension) von l ausgewählten Regeln.

Der Durchschnittswert D dieser Konklusionen ist:

$$D = \frac{c_1 + \ldots + c_l}{l}$$

Die Varianz V dieser Konklusionen ist:

$$V = \Big(c_1 - D\Big)^2 + \ldots + \Big(c_l - D\Big)^2$$

Beispiel 9.6 *Betrachte die reellen Konklusionen $\{5.1, 4.9, 4.8, 5.3\}$:*

Es ist:

$$D = \frac{5.1 + 4.9 + 4.8 + 5.3}{4} = \frac{20.1}{4} = 5.025$$

und

$$V = \Big(5.1 - 5.025\Big)^2 + \Big(4.9 - 5.025\Big)^2 + \Big(4.8 - 5.025\Big)^2 + \Big(5.3 - 5.025\Big)^2$$

$$= 0.005625 + 0.015625 + 0.050625 + 0.075625 = 0.1475$$

Zur Durchführung dieses Verfahrens werden für jedes Trainingsbeispiel die folgenden Schritte ausgeführt:

- bestimme die Erfüllungsgrade der Prämissen aller Regeln
- betrachte jeweils alle Regeln gemeinsam, die alle Eingabe–Variablen verwenden, deren Prämissen einen Erfüllungsgrad über einer Schranke haben und deren Konklusionen sich auf dieselbe Ausgabe–Dimension beziehen
- ist die Varianz der Konklusionen der betrachteten Regeln unter einer Schranke, fasse diese Regeln zusammen
  - Prämisse ist die Prämisse der Regel mit maximalem Erfüllungsgrad der Prämisse unter den betrachteten Regeln
  - Konklusion ist der Durchschnitt der Konklusionen der betrachteten Regeln

Beispiel 9.7 *Sei die gleiche Situation wie in Beispiel 9.3 gegeben.*
Ersetzt man die Regel

IF $x_1 =$ *sehr kalt UND* $x_2 =$ *Tag THEN* $y = 8$

unnötigerweise durch zwei Regeln, ergibt sich z.B. folgende Regelbasis:

IF $x_1 =$ *sehr kalt UND* $x_2 =$ *Tag THEN* $y = 8.1$
IF $x_1 =$ *sehr kalt UND* $x_2 =$ *Tag THEN* $y = 7.9$
IF $x_1 =$ *kalt UND* $x_2 =$ *Tag THEN* $y = 5$
IF $x_1 =$ *warm UND* $x_2 =$ *Tag THEN* $y = 2$
IF $x_2 =$ *Nacht THEN* $y = 2$

Das oben beschriebene Verfahren faßt die ersten beiden Regeln zu einer Regel zusammen:

IF $x_1 =$ *sehr kalt UND* $x_2 =$ *Tag THEN* $y = 8$

Damit entspricht die Regelbasis der ursprünglich in Beispiel 9.3 verwendeten.

Erzeugen von Fuzzy–Mengen:

Neue Fuzzy–Mengen werden analog zum MFOS–M–System erzeugt, jedoch systembedingt nur Eingabe–Partitions–Mengen. Daher ist das Verfahren zum Erzeugen neuer Fuzzy–Mengen in das Verfahren zum Erzeugen neuer Regeln integriert.

Bei der Erzeugung einer Regel werden für die Prämisse die Eingabe–Partitions–Mengen bestimmt, bei denen die Eingabe–Werte des betrachteten Trainingsbeispiels den höchsten Zugehörigkeitsgrad ergeben. Falls für einen Wert der Zugehörigkeitsgrad bei keiner Partitions–Menge aus der zugehörigen Eingabe–Dimension über einer Schranke liegt, ist keine passende Partitions–Menge für diesen Wert vorhanden. Daher wird während der Anwendung des Verfahrens zur Erzeugung neuer Regeln eine geeignete Fuzzy–Menge erzeugt und als Menge aus der Prämisse der Regel festgelegt:

Neue Partitions-Menge:

- Modalwert: betroffener Eingabe–Wert
- Breite: 1.2 mal Abstand zum Modalwert der nächsten Nachbar–Menge

Beispiel 9.8 *Werden in der Situation von Beispiel 9.2 die Eingabe–Partitions–Menge*

$\texttt{kalt} \mathrel{\widehat{=}} \widetilde{A}_2 = (16, 18, 20)$

und die Regel

IF $x =$ `kalt` *THEN* $y = 5$

vergessen, so wird durch das oben beschriebene Verfahren bei der Erzeugung der vergessenen Regel die Partitions–Menge $(16.2, 18, 19.8)$ *generiert, die nun für* `kalt` *steht. Gleichzeitig wird die vergessene Regel erstellt. Mit der erzeugten Fuzzy–Menge liefert der Sugeno–Controller genauso gute Ergebnisse wie mit der ursprünglichen Fuzzy–Menge* $(16, 18, 20)$.

Löschen von Fuzzy–Mengen:

Das MFOS–M–System überprüft, ob zwei benachbarte Fuzzy–Mengen gleichwertig sind, in dem festgestellt wird, ob zu allen Eingabe–Werten, die in einer von zwei Nachbar–Mengen liegen, die korrekte Ausgabe in der selben Ausgabe–Partitions–Menge liegt. Ist dies der Fall, werden die beiden Nachbar–Mengen zu einer Fuzzy–Menge zusammengefaßt. Beim MFOS–S–System entfällt das Kriterium "Zugehörigkeit zur Ausgabe–Partitions–Menge". Daher wird stattdessen die *Varianz* der korrekten Ausgabe–Werte zum Vergleich von Nachbar–Mengen herangezogen.

Definition 9.3
Seien $y_j^{(1)}, \ldots, y_j^{(l)}$ *die gewünschten Ausgabe–Werte für Ausgabe–Dimension* Y_j *von* l *ausgewählten Trainingsbeispielen.*

Der Durchschnittswert D *dieser Ausgabe–Werte ist:*

$$D = \frac{y_j^{(1)} + \ldots + y_j^{(l)}}{l}$$

Die Varianz V dieser Ausgabe–Werte ist:

$$V = \left(y_j^{(1)} - D\right)^2 + \ldots + \left(y_j^{(l)} - D\right)^2$$

Beispiel 9.9 *Betrachte die gewünschten Ausgabe–Werte* $\{3.1, 3.9, 2.8, 3.3\}$*:*

Es ist:

$$D = \frac{3.1 + 3.9 + 2.8 + 3.3}{4} = \frac{13.1}{4} = 3.275$$

und

$$V = \left(3.1 - 3.275\right)^2 + \left(3.9 - 3.275\right)^2 + \left(2.8 - 3.275\right)^2 + \left(3.3 - 3.275\right)^2$$

$$= 0.030625 + 0.390625 + 0.225625 + 0.000625 = 0.6475$$

Die Varianz ist ein geeignetes Kriterium, die Abweichungen von gewünschten Ausgaben ausgesuchter Trainingsbeispiele zu bewerten. Ist die Varianz und damit die Stärke der Abweichungen ausgesuchter Trainingsbeispiele gering, ist es nicht notwendig, zur Unterscheidung dieser Trainingsbeispiele verschiedene Eingabe–Partitions–Mengen zu verwenden. Falls dennoch unnötigerweise verschiedene Eingabe–Partitions–Mengen zur Unterscheidung dieser Trainingsbeispiele definiert wurden, faßt das MFOS–S–Verfahren diese zu einer gemeinsamen Fuzzy–Menge zusammen.

Es werden jeweils alle Trainingsbeispiele gemeinsam betrachtet, bei denen ein Eingabe–Wert in einer von zwei Nachbar–Mengen liegt, und alle anderen Eingabe–Werte jeweils in derselben Menge. Dabei wird die Partitions–Menge, in der ein Wert liegt, als diejenige bestimmt, bei der der Zugehörigkeitsgrad am höchsten ist. Falls bei allen gemeinsam betrachteten Beispielen jedesmal die gewünschte Ausgabe ähnlich ist, ist es gleichwertig, in welcher der beiden überprüften Nachbar–Mengen sich der Eingabe–Wert befindet. Somit können diese Mengen zusammengefaßt und die dann überflüssige Menge gelöscht werden.

Als Modalwert der neuen Partitions–Menge wird das arithmetische Mittel der Modalwerte von den beiden Nachbar–Mengen festgelegt. Die Breite wird so gewählt, daß der gesamte Bereich beider Mengen abgedeckt wird. Anschließend werden noch alle Regeln angepaßt, die sich nur bei den betrachteten Nachbar–Mengen unterscheiden. Zur Durchführung dieses Verfahrens werden die folgenden Schritte ausgeführt:

- betrachte jeweils alle Trainingsbeispiele gemeinsam, bei denen ein Eingabe–Wert in einer von zwei Nachbar–Mengen liegt, und alle anderen Eingabe–Werte in derselben Menge
- bestimme für alle betrachteten Beispiele —für jede Ausgabe–Dimension getrennt— die Varianz der gewünschten Ausgaben
- wenn die Varianz für jede Ausgabe–Dimension unter einer Schranke liegt, fasse die beiden Nachbar–Mengen zusammen:
  - Modalwert: arithmetisches Mittel der Modalwerte der Nachbar–Mengen
  - Breite: Breite des gemeinsam überdeckten Bereiches
- passe danach alle Regeln an, die sich nur in den beiden Nachbar–Mengen unterscheiden
  - Eingaben, die einer der Nachbar–Mengen zugeordnet sind, werden der zusammengefaßten Menge zugeordnet
  - die Konklusion bleibt unverändert

Bemerkung 9.3 *Eine geeignete Festlegung der Konklusionen zusammengefaßter Regeln ist bereits in das Verfahren zum Löschen von Regeln integriert und muß daher an dieser Stelle nicht zusätzlich berücksichtigt werden (s. Abschnitt 9.2):*

- *falls die zusammengefaßten Regeln dieselbe Konklusion besitzen, sind sie nun vollkommen gleich und überzählige Exemplare werden gelöscht*
- *falls die Varianz der Konklusionen zusammengefaßter Regeln unter einer Schranke liegt, werden dieses Regeln zusammengefaßt (s. Beispiel 9.7)*

Beispiel 9.10 *Sei die gleiche Situation wie in Beispiel 9.3 gegeben. Ersetzt man die Partitions–Menge für* `sehr kalt` $\widehat{=} \widetilde{A}_{11} = (13, 15, 17)$ *durch zwei Mengen, ergeben sich z.B. folgende Partitionierungen:*

Eingabe 1 (Temperatur in °C): Dreiecks–Mengen auf dem Grundraum $[13, 23]$*:*

$$\begin{aligned}
\texttt{besonders kalt} &\mathrel{\widehat{=}} \widetilde{A}_{11} = (13, 14, 15)\\
\texttt{sehr kalt} &\mathrel{\widehat{=}} \widetilde{A}_{12} = (15, 16, 17)\\
\texttt{kalt} &\mathrel{\widehat{=}} \widetilde{A}_{12} = (16, 18, 20)\\
\texttt{warm} &\mathrel{\widehat{=}} \widetilde{A}_{13} = (19, 21, 23)
\end{aligned}$$

Eingabe 2 (Tag / Nacht): Dreiecks–Mengen auf dem Grundraum $[0, 4]$*:*

$$\begin{aligned}
\texttt{Tag} &\mathrel{\widehat{=}} \widetilde{A}_{21} = (0, 1, 2)\\
\texttt{Nacht} &\mathrel{\widehat{=}} \widetilde{A}_{22} = (2, 3, 4)
\end{aligned}$$

Die linguistischen Variablen sind x_1 *für die Temperatur,* x_2 *für* `Tag / Nacht` *und* y *für die Heizleistung.*

Die verwendeten Regeln sind:

$$\begin{aligned}
&\texttt{IF } x_1 = \texttt{besonders kalt } \texttt{UND } x_2 = \texttt{Tag } \texttt{THEN } y = 8.1\\
&\texttt{IF } x_1 = \texttt{sehr kalt } \texttt{UND } x_2 = \texttt{Tag } \texttt{THEN } y = 7.9\\
&\texttt{IF } x_1 = \texttt{kalt } \texttt{UND } x_2 = \texttt{Tag } \texttt{THEN } y = 5\\
&\texttt{IF } x_1 = \texttt{warm } \texttt{UND } x_2 = \texttt{Tag } \texttt{THEN } y = 2\\
&\texttt{IF } x_2 = \texttt{Nacht } \texttt{THEN } y = 2
\end{aligned}$$

Das oben beschriebene Verfahren erkennt, daß die Aufteilung des unteren Temperaturbereiches in `besonders kalt` *und* `sehr kalt` *nicht nötig ist und faßt beide Partitions–Mengen zur Partitions–Menge* $(13, 15, 17)$ *zusammen, die nun für* `sehr kalt` *steht. Gleichzeitig werden die Regeln*

$$\begin{aligned}
&\texttt{IF } x_1 = \texttt{besonders kalt } \texttt{UND } x_2 = \texttt{Tag } \texttt{THEN } y = 8.1\\
&\texttt{IF } x_1 = \texttt{sehr kalt } \texttt{UND } x_2 = \texttt{Tag } \texttt{THEN } y = 7.9
\end{aligned}$$

angepaßt zu

$$\begin{aligned}
&\texttt{IF } x_1 = \texttt{sehr kalt } \texttt{UND } x_2 = \texttt{Tag } \texttt{THEN } y = 8.1.\\
&\texttt{IF } x_1 = \texttt{sehr kalt } \texttt{UND } x_2 = \texttt{Tag } \texttt{THEN } y = 7.9.
\end{aligned}$$

Die nun überflüssige Partitions–Menge für `besonders kalt` *wird gelöscht.*

Schließlich werden die beiden angepaßten Regeln zu einer Regel zusammengefaßt:

IF $x_1 =$ *sehr kalt UND* $x_2 =$ *Tag THEN* $y = 8$ *hoch.*

Damit entsprechen die Partitions–Mengen und Regeln den ursprünglich in Beispiel 9.3 verwendeten.

Modifikation von Fuzzy–Mengen und Regel–Konklusionen:

Zur Modifikation der Fuzzy–Mengen (Modalwert und Weite) sowie der reellen Regel–Konklusionen wird das Gradientenabstiegsverfahren eingesetzt. Die Aktivitäts– und Ausgabe–Funktionen in Schicht 1 und Schicht 2 sind exakt dieselben wie bei MFOS–M–Netzen (s. Abschnitt 9.1). In Schicht 3 unterscheiden sich MFOS–M– und MFOS–S–Netze (eine vierte Schicht existiert bei MFOS–S–Netzen nicht). Analog zum MFOS–M–System werden Dreiecks– und Gauß–Mengen verwendet. Dies ist bei der Herleitung zu unterscheiden. Da keine Defuzzifizierung vorgenommen wird, gibt es somit zwei zu betrachtende Fälle:

- MFOS–S–System mit Gauß–Mengen
- MFOS–S–System mit Dreiecks–Mengen

Die zu ändernden Parameter sind die Modalwerte m und Weiten w der Eingabe–Partitions–Mengen sowie die reellen Konklusionen c der Regeln. Daher werden die partiellen Ableitungen der Fehlerfunktion F nach m, w und c separat berechnet und die Werte entsprechend folgender Formeln angepaßt:

$$\Delta m = -\eta \cdot \frac{\partial F}{\partial m}$$

$$\Delta w = -\eta \cdot \frac{\partial F}{\partial w}$$

$$\Delta c = -\eta \cdot \frac{\partial F}{\partial c}$$

mit $\eta > 0$ der üblichen Lernrate.

Bemerkung 9.4 *Analog zu einem MFOS–M–Netz muß auch hier sichergestellt werden, daß die Weiten der Fuzzy–Mengen nicht negativ werden (vgl. Kapitel 6). Daher gilt für die tatsächlich durchgeführte Änderung von w:*

$$\Delta w = -\eta \cdot \frac{\partial F}{\partial w}, \; falls \; \eta \cdot \frac{\partial F}{\partial w} > w$$

Schicht 3:

In Schicht 3 gibt es für jede Dimension des Ausgabe–Raumes ein Neuron.

Aktivitätsfunktion von Neuron j der Ausgabeschicht ist:

$$f_{a\_3,j} = \sum_{k \in V_2(j)} o_{2,k} \cdot c_k$$

Ausgabefunktion von Neuron j der Ausgabeschicht ist:

$$f_{o\_3,j} = \frac{f_{a\_3,j}}{\sum\limits_{k \in V_2(j)} o_{2,k}}$$

mit

- $V_2(j)$ der Menge der Verbindungen zwischen Neuron j der Ausgabeschicht und den Neuronen aus Schicht 2
- $o_{2,k}$ der Ausgabe von Neuron k aus Schicht 2
- c_k der Konklusion von Regel R_k
  (Regel R_k wird durch Neuron k aus Schicht 2 repräsentiert)

Die Fehlerfunktion ist:

$$F = \frac{1}{2} \cdot \sum_{j=1}^{m} \left(y_j - o_{3,j}\right)^2$$

mit y_j der gewünschten und $o_{3,j}$ der tatsächlichen Ausgabe von Neuron j aus Schicht 3.

Da sich jede Regel R_k auf eine Ausgabe–Dimension j bezieht, werden zur Bestimmung der partiellen Ableitungen nach der Regel–Konklusion c_k jeweils die Aktivitäts– und Ausgabe–Funktionen des zugehörigen Ausgabe–Neurons j verwendet. Für die partielle Ableitung der Fehlerfunktion F nach der Regel–Konklusion von Regel R_k, c_k, gilt daher:

$$\frac{\partial F}{\partial c_k} = \frac{\partial F}{\partial f_{a\_3,j}} \cdot \frac{\partial f_{a\_3,j}}{\partial c_k} = \frac{\partial F}{\partial f_{o\_3,j}} \cdot \frac{\partial f_{o\_3,j}}{\partial f_{a\_3,j}} \cdot \frac{\partial f_{a\_3,j}}{\partial c_k} \qquad \text{mit:}$$

$$\frac{\partial F}{\partial f_{o\_3,j}} = \left(\frac{1}{2} \cdot \left(y_j - o_{3,j}\right)^2\right)' = -(y_j - o_{3,j})$$

(eindimensionale Kettenregel, $o_{3,j}$ als Variable)

$$\frac{\partial f_{o\_3,j}}{\partial f_{a\_3,j}} = \frac{1}{\sum\limits_{k \in V_2(j)} o_{2,k}}$$

$$\frac{\partial f_{a\_3,j}}{\partial c_k} = o_{2,k}$$

Schicht 2:

In dieser Schicht gibt es für jede Regel ein Neuron. Dieses ist mit *dem* Neuron aus Schicht 3 verbunden, welches die Ausgabe–Dimension repräsentiert, auf die sich diese Regel bezieht. Sei j der Index des Neurons aus Schicht 3, das mit Neuron Nr. i aus Schicht 2 verbunden ist. Die Prämisse der jeweiligen Regel ist durch die Verbindungen mit den Neuronen aus Schicht 1 realisiert. Dabei werden als Gewichte gerade die Eingabe–Partitions–Mengen verwendet, die bei der Prämisse berücksichtigt werden. Falls eine Regel nicht alle Eingabe–Variablen berücksichtigt, gibt es zu den entsprechenden Eingabe–Neuronen (Schicht 1) keine Verbindung. Seien $l_1, \ldots, l_{r(i)}$ die Indizes der Neuronen aus Schicht 1, die mit Neuron Nr. i aus Schicht 2 verbunden sind.

Die Ausgabe von Neuron Nr. i dieser Schicht wird berechnet durch:

$$f_{o\_2,i} = \min\left(\mu_{\widetilde{A}_{il_1}}(o_{1,l_1}), \ldots, \mu_{\widetilde{A}_{il_{r(i)}}}(o_{1,l_{r(i)}})\right)$$

wobei $\widetilde{A}_{il_*}$ die Fuzzy–Mengen aus der Prämisse von Regel Nr. i sind, und o_{1,l_*} die Ausgaben der entsprechenden Neuronen aus Schicht 1.

Für die partielle Ableitung der Fehlerfunktion nach den Parametern der Eingabe–Fuzzy–Mengen gilt daher:

$$\frac{\partial F}{\partial m_{ik}} = \frac{\partial F}{\partial f_{o\_2,i}} \cdot \frac{\partial f_{o\_2,i}}{\partial m_{ik}} = \frac{\partial F}{\partial f_{a\_3,j}} \cdot \frac{\partial f_{a\_3,j}}{\partial f_{o\_2,i}} \cdot \frac{\partial f_{o\_2,i}}{\partial m_{ik}}$$

$$= \frac{\partial F}{\partial f_{o\_3,j}} \cdot \frac{\partial f_{o\_3,j}}{\partial f_{a\_3,j}} \cdot \frac{\partial f_{a\_3,j}}{\partial f_{o\_2,i}} \cdot \frac{\partial f_{o\_2,i}}{\partial m_{ik}}$$

mit:

$$\frac{\partial F}{\partial f_{o\_3,j}} = \left(\frac{1}{2} \cdot \left(y_j - o_{3,j} \right)^2 \right)' = -(y_j - o_{3,j}) =: \delta_{3,j}$$

(Dies ist der Fehleranteil von Neuron Nr. j aus Schicht 3. $\delta_{3,j}$ wird daher direkt in diesem Neuron berechnet.)

$$\frac{\partial f_{o\_3,j}}{\partial f_{a\_3,j}} = \frac{1}{\sum\limits_{k \in V_2(j)} o_{2,k}}$$

$$\frac{\partial f_{a\_3,j}}{\partial f_{o\_2,i}} = c_i$$

Analog gilt:

$$\frac{\partial F}{\partial w_{ik}} = \frac{\partial F}{\partial f_{o\_3,j}} \cdot \frac{\partial f_{o\_3,j}}{\partial f_{a\_3,j}} \cdot \frac{\partial f_{a\_3,j}}{\partial f_{o\_2,i}} \cdot \frac{\partial f_{o\_2,i}}{\partial w_{ik}}$$

Lediglich für die Berechnung von $\frac{\partial f_{o\_2,i}}{\partial m_{ik}}$ und $\frac{\partial f_{o\_2,i}}{\partial w_{ik}}$ muß zwischen Gauß– und Dreiecks–Mengen unterschieden werden.

Bei Verwendung von Gauß–Mengen gilt:

$$\frac{\partial f_{o\_2,i}}{\partial m_{ik}} = \begin{cases} 0: & \text{falls } \mu_{\widetilde{A}_{ik}}(z_{1k}) \text{ nicht minimal} \\ \left(e^{\frac{-(z_{1k}-m_{ik})^2}{w_{ik}^2}}\right)' & \text{sonst} \end{cases}$$

dabei ist:

$$\left(e^{\frac{-(z_{1k}-m_{ik})^2}{w_{ik}^2}}\right)' = e^{\frac{-(z_{1k}-m_{ik})^2}{w_{ik}^2}} \cdot \frac{2 \cdot (z_{1k} - m_{ik})}{w_{ik}^2}$$

(eindimensionale Kettenregel, m_{ik} als Variable)

$$\frac{\partial f_{o\_2,i}}{\partial w_{ik}} = \begin{cases} 0: & \text{falls } \mu_{\widetilde{A}_{ik}}(z_{1k}) \text{ nicht minimal} \\ \left(e^{\frac{-(z_{1k}-m_{ik})^2}{w_{ik}^2}}\right)' & \text{sonst} \end{cases}$$

dabei ist:

$$\left(e^{\frac{-(z_{1k}-m_{ik})^2}{w_{ik}^2}}\right)' = e^{\frac{-(z_{1k}-m_{ik})^2}{w_{ik}^2}} \cdot \frac{2 \cdot (z_{1k} - m_{ik})^2}{w_{ik}^3}$$

(eindimensionale Kettenregel, w_{ik} als Variable)

Bei Verwendung von Dreiecks–Mengen gilt:

$$\frac{\partial f_{o\_2,i}}{\partial m_{ik}} = \begin{cases} 0: & \text{falls } \mu_{\widetilde{A}_{ik}}(z_{1k}) \text{ nicht minimal} \\ \left(1 - \frac{m_{ik}-z_{1k}}{w_{ik}}\right)' & \text{falls } m_{ik} - w_{ik} \leq z_{1k} \leq m_{ik} \\ \left(1 + \frac{m_{ik}-z_{1k}}{w_{ik}}\right)' & \text{falls } m_{ik} < z_{1k} \leq m_{ik} + w_{ik} \end{cases}$$

dabei ist:

$$\left(1 - \frac{m_{ik} - z_{1k}}{w_{ik}}\right)' = -\frac{1}{w_{ik}}$$

$$\left(1 + \frac{m_{ik} - z_{1k}}{w_{ik}}\right)' = \frac{1}{w_{ik}}$$

(m_{ik} als Variable)

und:

$$\frac{\partial f_{o\_2,i}}{\partial w_{ik}} = \begin{cases} 0: & \text{falls } \mu_{\widetilde{A}_{ik}}(z_{1k}) \text{ nicht minimal} \\ \left(1 - \frac{m_{ik} - z_{1k}}{w_{ik}}\right)' & \text{falls } m_{ik} - w_{ik} \leq z_{1k} \leq m_{ik} \\ \left(1 + \frac{m_{ik} - z_{1k}}{w_{ik}}\right)' & \text{falls } m_{ik} < z_{1k} \leq m_{ik} + w_{ik} \end{cases}$$

dabei ist:

$$\left(1 - \frac{m_{ik} - z_{1k}}{w_{ik}}\right)' = \frac{m_{ik} - z_{1k}}{w_{ik}^2}$$
$$\left(1 + \frac{m_{ik} - z_{1k}}{w_{ik}}\right)' = -\frac{m_{ik} - z_{1k}}{w_{ik}^2}$$

(w_{ik} als Variable)

Beispiel 9.11 *Sei die gleiche Situation wie in Beispiel 9.2 gegeben. Bei den folgenden Fuzzy–Mengen und Regeln sind die Modalwerte und die reellen Konklusionen ungünstig gewählt und führen zu Fehlern bei der Ausgabe:*

Eingabe (Temperatur in °C): Dreiecks–Mengen auf dem Grundraum $[12, 24]$*:*

sehr kalt $\widehat{=} \widetilde{A}_1 = (12, 14, 16)$
kalt $\widehat{=} \widetilde{A}_2 = (17, 19, 21)$
warm $\widehat{=} \widetilde{A}_3 = (20, 22, 24)$

Die linguistischen Variablen sind x für die Temperatur und y für die Heizleistung.

Die verwendeten Regeln sind:

IF $x =$ *sehr kalt THEN* $y = 7$
IF $x =$ *kalt THEN* $y = 4$
IF $x =$ *warm THEN* $y = 1$

Nach der Durchführung des Gradientenabstiegsverfahrens ergeben sich folgende Fuzzy–Mengen und Regeln, mit denen die berechneten Ausgabewerte korrekt sind:

$$\begin{aligned} \textit{sehr kalt} &\mathrel{\widehat{=}} \widetilde{A}_1 = (14.56, 15.8965, 17.24) \\ \textit{kalt} &\mathrel{\widehat{=}} \widetilde{A}_2 = (16.51, 18.2862, 20.06) \\ \textit{warm} &\mathrel{\widehat{=}} \widetilde{A}_3 = (18.86, 21.0604, 23.26) \end{aligned}$$

$$\begin{aligned} &\textit{IF } x = \textit{sehr kalt } \textit{THEN } y = 8 \\ &\textit{IF } x = \textit{kalt } \textit{THEN } y = 5 \\ &\textit{IF } x = \textit{warm } \textit{THEN } y = 2 \end{aligned}$$

9.3 Rückportierung auf einen Sugeno–Controller

Sämtliche vorgestellten Lernverfahren für ein MFOS–S–Netz verändern einzelne Parameter (Gewichte) des Netzes oder die Struktur des Netzes. Dabei entsteht jedoch immer ein Netz, daß der Definition eines MFOS–S–Netzes entspricht:

- Beim Korrigieren einer Regel wird der Wert der reellen Konklusion dieser Regel geändert. Eine Strukturänderung findet nicht statt.
- Beim Erzeugen einer Regel wird ein neues Neuron in Schicht 2 eingefügt und mit Schicht 1 und Schicht 3 verbunden. Als Gewichte werden geeignete Eingabe–Partitions–Mengen und eine geeignete reelle Konklusion gewählt. Somit entspricht das neu entstandene Netz der Definition eines MFOS–S–Netzes.
- Beim Löschen einer Regel wird das zugehörige Neuron aus Schicht 2 sowie seine Verbindungen zu Schicht 1 und Schicht 3 entfernt. Auch hier entspricht das neu entstandene Netz weiterhin der Definition eines MFOS–S–Netzes.
- Neue Fuzzy–Mengen werden parallel beim Erzeugen neurer Regeln erzeugt und als Gewichte der Verbindungen zwischen Schicht 1 und Schicht 2 eingefügt. Auch dies entspricht der Definition eines MFOS–S–Netzes.
- Das Löschen von Fuzzy–Mengen ergibt dieselben strukturellen Änderungen wie das Löschen von Regeln.
- Die Modifikation von Fuzzy–Mengen und der reellen Regel–Konklusionen ändert Gewichte. Eine Strukturänderung findet nicht statt.

Nach Anwendung der Lernverfahren steht somit in jedem Fall ein korrektes MFOS–S–Netz zur Verfügung, das einen Sugeno–Controller repräsentiert. Daher ist eine Rücktransformation dieses Netzes auf einen Sugeno–Controller möglich.

Definition 9.4 *Abbildung–4: MFOS–S $\longrightarrow$ SC ist die im Folgenden beschriebene Portierung eines MFOS–S–Netzes auf einen Sugeno–Controller.*

Der Sugeno–Controller wird wie folgt aus einem MFOS–S–Netz gebildet:

Jedes Neuron in Schicht 2 repräsentiert eine Regel. Die Verbindungen zu Schicht 1 ergeben die Prämisse. Die Gewichte dieser Verbindungen entsprechen den Eingabe–Partitions–Mengen. Die Verbindung zu Schicht 3 mit dem reellen Gewicht entspricht der Konklusion der Regel.

Ist Neuron R_k aus Schicht 2 mit den Neuronen $1, \ldots, n$ aus Schicht 1 über die Gewichte $\widetilde{A}_{1*}, \ldots, \widetilde{A}_{n*}$ verbunden und über das Gewicht c_k mit Ausgabe–Neuron 1, so ergibt sich Regel R_k wie folgt:

R_k : `IF` $x_1 = \widetilde{A}_{1*}$ `UND` ... `UND` $x_n = \widetilde{A}_{n*}$ `THEN` $y_1 = c_k$

Falls Neuron R_k mit einzelnen Eingabe–Neuronen nicht verbunden ist, entfallen die entsprechenden Eingaben bei der Regel. D.h. die Regel berücksichtigt nicht alle Eingabe–Werte.

Beispiel 9.12 *Sei das Neuron R_4 aus Schicht 2 mit folgenden Verbindungen gegeben:*

- *es gibt je eine Verbindung mit Neuron 1 und Neuron 2 aus Schicht 1, mit den Eingabe–Partitions–Mengen $\widetilde{A}_{13}$ bzw. $\widetilde{A}_{21}$ als Gewicht*
- *es gibt eine Verbindung mit Ausgabe–Neuron 1, die als Gewicht c_4 erhält*

Daraus läßt sich Regel R_4 extrahieren:

`IF` $x_1 = \widetilde{A}_{13}$ *`UND`* $x_2 = \widetilde{A}_{21}$ *`THEN`* $y_1 = c_4$

Satz 9.2
Abbildung–4 ist korrekt, d.h. das Ein– Ausgabe–Verhalten eines MFOS–S–Netzes und das Ein– Ausgabe–Verhalten des zugehörigen Sugeno–Controller sind identisch.

Beweis 9.2
Da Abbildung–4 die Umkehrabbildung von Abbildung–3 ist, erfolgt der Beweis analog zum Beweis von Satz 9.1.

Sei NN ein korrektes (d.h. der Definition entsprechendes) MFOS–S–Netz.
Seien $R_1, \ldots, R_{\#Reg}$ die Neuronen aus Schicht 2 von NN und jeweils

$\widetilde{A}_{1*}, \ldots, \widetilde{A}_{n*}$ die Gewichte der Verbindungen mit Schicht 1 bis n.

Sei SC der mit Abbildung–4 zu NN generierte Sugeno–Controller.

Seien $r_1, \ldots, r_n \in \mathbb{R}^n$ die Eingaben von NN bzw. SC, seien $o_{3,1}, \ldots, o_{3,m}$ die berechneten Ausgaben von NN und $s_1, \ldots, s_m$ die berechneten Ausgaben von SC zu den Eingaben $r_1, \ldots, r_n$.

Zu zeigen ist: $o_{3,j} = s_j$ *für* $j = 1, \ldots, m$.

Der zugehörige Sugeno–Controller SC enthält nach Definition zu jedem Neuron R_k aus Schicht 2 eine gleichnamige Regel. Die Fuzzy–Mengen, die in der Prämisse der Regeln verwendet werden, sind jeweils die Gewichte der Verbindungen zu Schicht 1. Im Korrektheitsbeweis zu Abbildung–2 (s. [Tenhagen, 2000]) wurde bereits gezeigt, daß der Erfüllungsgrad von Regel R_k bei einem Mamdani–Controller gleich der berechneten Ausgabe von Neuron R_k aus Schicht 2 des zugehörigen MFOS–M–Netzes ist.

Da

1. *die Erfüllungsgrade der Prämissen der Regeln bei einem Sugeno–Controller auf dieselbe Weise berechnet werden wie bei einem Fuzzy–Controller nach Mamdani und*
2. *Schicht 1 und Schicht 2 beim MFOS–M–Netz und beim MFOS–S–Netz identisch sind,*

folgt:

Der Erfüllungsgrad von Regel R_k von SC ist beim MFOS–S–Netz gleich der berechneten Ausgabe von Neuron R_k aus Schicht 2. D.h. es gilt für jede Regel R_k:

$$E_k = o_{2,k}$$

mit E_k dem Erfüllungsgrad der Prämisse von Regel R_k und $o_{2,k}$ der berechneten Ausgabe von Neuron k aus Schicht 2.

Daraus folgt mit Hilfe der Definition eines Sugeno–Controller (s. Kapitel 3.12) für die berechnete Ausgabe von SC s_j, $j = 1, \ldots, m$:

$$s_j = \frac{\sum\limits_{k \in V_2(j)} E_k \cdot c_k}{\sum\limits_{k \in V_2(j)} E_k} = \frac{\sum\limits_{k \in V_2(j)} o_{2,k} \cdot c_k}{\sum\limits_{k \in V_2(j)} o_{2,k}}$$

$$= \frac{f_{a\_3,j}}{\sum\limits_{k \in V_2(j)} o_{2,k}} = f_{o\_3,j}(f_{a\_3,j}) = o_{3,j}$$

□

Das MFOS–S–System stellt somit eine Methode zur Verfügung, einen potentiell beliebigen Sugeno–Controller auf ein funktional äquivalentes neuronales Netz zu übertragen, dieses zu optimieren, und anschließend wieder auf einen funktional äquivalenten (optimierten) Sugeno–Controller zu übertragen. Analog zum MFOS–M–System läßt sich auch beim MFOS–S–System Pre–Tuning und Fine–Tuning unterscheiden:

- Pre–Tuning sind die Verfahren zum Korrigieren, Erzeugen und Löschen von Regeln sowie zum Erzeugen und Löschen von Fuzzy–Mengen.
- Fine–Tuning sind die Verfahren zur Modifikation von Fuzzy–Mengen und der reellen Regel–Konklusionen mit dem Gradientenabstiegsverfahren.

Analog zum MFOS–S–System ist folgender Ablauf (s. Abb. 9.3) üblich: ein gegebener Sugeno–Controller SC 1 wird auf das MFOS–S–Netz NN 1 übertragen, in zwei Phasen optimiert, und das entstandene MFOS–S–Netz NN 3 wird zurücktransformiert in den optimierten Sugeno–Controller SC 3.

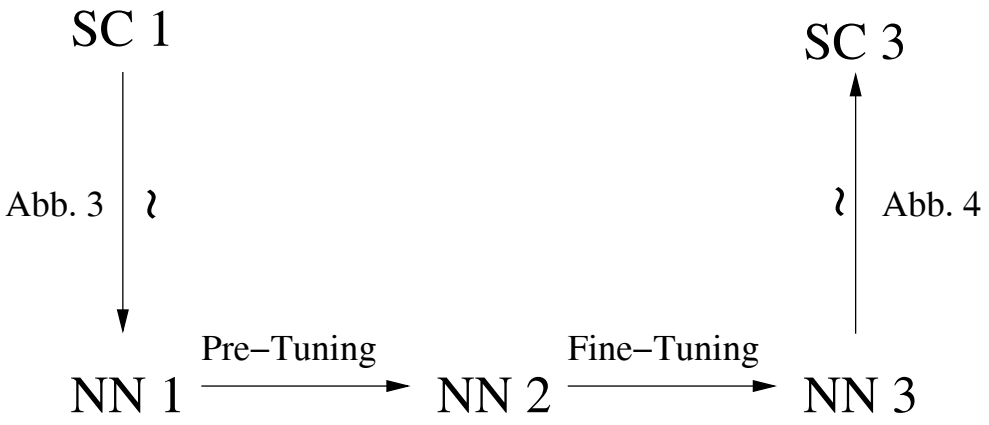

Abbildung 9.3: Diagramm der MFOS–S–Abbildungen

Kapitel 10

Vergleich der Optimierungs–Systeme

In Teil III dieser Arbeit wurde stellvertretend für Optimierungs–Systeme für Sugeno–Controller das ANFIS–System (Kapitel 8) vorgestellt. Weiterhin wurde mit dem MFOS–S–System (Kapitel 9) eine Version des MFOS für Sugeno–Controller entwickelt. In diesem Kapitel wird eine vergleichende Bewertung dieser Ansätze vorgestellt.

10.1 Einsetzbare Sugeno–Controller

Das ANFIS–System:

Das ANFIS–System ermöglicht die Übertragung eines gegebenen Sugeno–Controllers auf ein funktional äquivalentes neuronales Netz. Die Anzahl der Eingabe– und Ausgabe–Dimensionen ist beliebig, ebenso die Art der Eingabe–Partitions–Mengen. Auch die verwendete T–Norm ist frei wählbar. Eine Defuzzifizierung findet bei Sugeno–Controllern definitionsbedingt nicht statt.

Das MFOS–S–System:

Das MFOS–S–System ermöglicht die Übertragung eines gegebenen Sugeno–Controllers auf ein funktional äquivalentes neuronales Netz. Dabei gibt es keine besonderen Einschränkungen. Es sind beliebig viele Eingabe– und Ausgabe–Dimensionen möglich. Als Fuzzy–Mengen werden prinzipiell Gauß– bzw. Dreiecks–Mengen verwendet. Die T–Norm ist frei wählbar.

Vergleich und Bewertung:

Beide vorgestellten Systeme ermöglichen prinzipiell die Übertragung eines vorgegebenen Sugeno–Controllers auf ein funktional äquivalentes neuronales Netz. Dabei werden keine besonderen Voraussetzungen gemacht. Die verwendeten neuronalen Netze unter-

scheiden sich lediglich in der konkreten Art, einen Sugeno–Controller zu repräsentieren. D.h. die Struktur der verwendeten Netze und die Anzahl der verwendeten Schichten ist unterschiedlich. Die Berechnung der Netzausgabe ist bei beiden Systemen nachweislich (s. Kapitel 8 und 9) äquivalent zur Berechnung der Ausgabe des verwendeten Sugeno–Controllers.

10.2 Optimierungsmöglichkeiten

Die beiden vorgestellten Optimierungs–Systeme übertragen einen Sugeno–Controller auf ein spezielles neuronales Netz (jedes System mit einer individuellen Methode), um dieses zu trainieren. Während des Trainings werden strukturelle Änderungen des Netzes bzw. Änderungen der Gewichte durchgeführt. Diese Adaptionen des neuronalen Netzes korrelieren mit Anpassungen der zu optimierenden Bestandteile des verwendeten Sugeno–Controllers. D.h. es werden durch das Training die Regeln, Konklusionen und die Eingabe–Partitionierungen des Sugeno–Controllers eingestellt.

Jedoch werden die gleichen Adaptionen mit dem selben Ziel von den einzelnen Optimierungs–Systemen z.T. mit unterschiedlichen Methoden realisiert. Auch ist nicht mit jedem System jede theoretisch mögliche Modifikation durchführbar.

Das ANFIS–System:

Folgende Modifikationen sind prinzipiell möglich:

- Modifikation von reellen Konklusions–Werten
- Modifikation von Eingabe–Partitions–Mengen (Modalwert und Weite)
- Ausgabe–Partitions–Mengen sind definitionsbedingt nicht vorhanden (Sugeno–Controller)

Es findet ausschließlich eine Adaption der Parameter statt. Die reellen Regel Konklusionen werden mit Hilfe des Backpropagation–Algorithmus bzw. mit einer linearen Methode verändert. Die Modalwerte und Weiten der Eingabe–Partitions–Mengen werden nur mit Hilfe des Backpropagation–Algorithmus eingestellt. Ein Erzeugen neuer Partitions–Fuzzy–Mengen oder neuer Regeln etc. ist grundsätzlich nicht vorgesehen (s. Kapitel 8).

Das MFOS–S–System:

Folgende Modifikationen sind prinzipiell möglich:

- Neufestlegung von Regel–Konklusionen

- Erzeugen neuer Regeln
- Löschen unnötiger Regeln
- Erzeugen neuer Eingabe–Partitions–Mengen
- Löschen unnötiger Partitions–Fuzzy–Mengen
- Modifikation von reellen Konklusions–Werten
- Modifikation von Eingabe–Partitions–Mengen (Modalwert und Weite)
- Ausgabe–Partitions–Mengen sind definitionsbedingt nicht vorhanden (Sugeno–Controller)

Die Auswahl und Reihenfolge der Modifikationen durch das MFOS–S–System ist analog zum MFOS–M–System dem User überlassen (vgl. Kapitel 6 und 9). Für die genaue Darstellung der einzelnen Methoden s. Kapitel 9. Zur Neufestlegung von Regel–Konklusionen (d.h. Korrigieren von Regeln) wird die korrekte Ausgabe des Trainingsbeispiels ausgewählt, welches den maximalen Erfüllungsgrad der Prämisse der überprüften Regel bewirkt.

Neue Regeln werden erzeugt, falls zu einem Trainingsbeispiel keine geeignete Regel vorhanden ist. Kriterium hierfür ist der Erfüllungsgrad der Prämisse der Regeln. Zur Definition der Prämisse einer neuen Regel werden diejenigen Eingabe–Partitions–Mengen ausgewählt, die das aktuelle Trainingsbeispiel am besten repräsentieren. Kriterium hierfür ist der Zugehörigkeitsgrad der Eingabe–Werte zu den Eingabe–Partitions–Mengen. Die Konklusion wird mit Hilfe der korrekten Ausgaben des aktuellen Trainingsbeispiels bestimmt. Falls Regeln einen sehr geringen Einfluß auf das Ergebnis haben, werden sie gelöscht bzw. zusammengefaßt. Kriterium hierfür ist der Erfüllungsgrad der Prämisse der Regeln bzw. die Varianz der reellen Konklusionen.

Falls beim Erzeugen von Regeln festgestellt wird, daß mit keiner vorhandenen Eingabe–Partitions–Menge der Erfüllungsgrad groß genug ist, werden neue Eingabe– Partitions–Mengen erzeugt. Als Modalwert werden die Eingabe–Werte des aktuellen Trainingsbeispiels verwendet. Die Weite wird so gewählt, das eine Überlappung zu eventuellen Nachbar–Mengen gegeben ist. Auf diese Weise wird erstens sichergestellt, daß die erzeugten Partitions–Fuzzy–Mengen perfekt zum aktuellen Trainingsbeispiel passen. Zweitens wird erreicht, daß sich die neu erzeugten Partitions–Fuzzy–Mengen in die vorhandenen Partitionierungen einfügen. Somit sind neu erzeugte Partitions–Mengen auch für andere Traininsgbeispiele mit ähnlichen Werten geeignet.

Neben dem Erzeugen neuer Partitions–Fuzzy–Mengen ist auch das Löschen unnötiger Partitions–Fuzzy–Mengen möglich. Gleichwertige Eingabe–Partitions–Mengen lassen sich zu einer Menge zusammenfassen. Kriterium hierfür ist, ob zwei benachbarte Eingabe–Partitions–Mengen zur Unterscheidung verschiedener Ausgabe–Werte notwendig sind oder nicht. Falls in dem von zwei Nachbar–Mengen überdeckten Bereich

die korrekte Ausgabe nur wenig variiert, genügt eine größere Menge für diesen Bereich. Anderenfalls sind zwei einzelne Fuzzy–Mengen unverzichtbar.

Zusätzlich zu diesen als Pre–Tuning bezeichneten Modifikationen ist auch ein Fine–Tuning möglich. Als Fine–Tuning wird beim MFOS–S–System die Anpassung der Parameter der Eingabe–Partitions–Mengen (Modalwert und Weite) und der reellen Konklusionen durchgeführt. Dies geschieht mit dem Gradientenabstiegsverfahren.

Vergleich und Bewertung:

Beide Optimierungs–Systeme ermöglichen die Anpassung ausgesuchter Komponenten eines Sugeno–Controllers. Jedoch ist es nur mit dem MFOS–S–System möglich, jede Komponente eines Sugeno–Controllers nach Bedarf optimieren zu lassen. Das ANFIS–System beschränkt sich auf die Modifikation vorhandener reeller Werte.

Mit dem ANFIS–System werden ausschließlich die Parameter der Eingabe–Partitions–Mengen und die reellen Regel–Konklusionen angepaßt. Eine Korrektur oder Erzeugung von neuen Regeln bzw. ein Erzeugen zusätzlicher Partitions–Fuzzy–Mengen ist nicht vorgesehen. Dies führt zu den bekannten Enschränkungen, falls bei der Konfiguration des zugehörigen Netzes zur korrekten Steuerung unverzichtbare Partitions–Fuzzy–Mengen bzw. Regeln vergessen wurden.

Beispiel 10.1 *Die folgenden Fuzzy–Mengen und Regeln ergeben die Partitionierungen und die Regelbasis für einen einfachen Sugeno–Controller zur Steuerung eines Heizgeräts:*

Eingabe (Temperatur in °C): Gauß–Mengen (Modalwert, Weite):

$$\begin{aligned} \texttt{sehr kalt} &\mathrel{\widehat{=}} \widetilde{A}_1 = (15,3) \\ \texttt{kalt} &\mathrel{\widehat{=}} \widetilde{A}_2 = (18,3) \\ \texttt{warm} &\mathrel{\widehat{=}} \widetilde{A}_3 = (21,3) \end{aligned}$$

Die linguistischen Variablen sind x für die Temperatur und y für die Heizleistung.

Die richtigen Regeln sind:

$$\begin{aligned} &\texttt{IF } x = \texttt{sehr kalt } \texttt{THEN } y = 8 \\ &\texttt{IF } x = \texttt{kalt } \texttt{THEN } y = 5 \\ &\texttt{IF } x = \texttt{warm } \texttt{THEN } y = 2 \end{aligned}$$

Wird nun bei der Erstellung des Netzes die Eingabe–Partitions–Menge **kalt** *und die Regel*

IF $x =$ **kalt** *THEN* $y = 5$

vergessen, so ist es mit den ANFIS–Lernverfahren nicht möglich, die gegebenen Fuzzy–Mengen und Konklusionen so einzustellen, daß die Steuerung in jeder Situation funktioniert. Hierfür wäre eine zusätzliche Eingabe–Partitions–Menge und eine zusätzliche Regel erforderlich, die jedoch nicht erzeugt werden können.

Den Autoren ist dieser Nachteil durchaus bewußt (s. [Jang, 1993]). Sie empfehlen, die optimale Struktur des zugehörigen Netzes durch ausprobieren basierend auf Erfahrungswerten herauszufinden, wie dies z.B. beim klassischen Multilayer–Perzeptron üblich ist. Allerdings ist ein ANFIS–Netz wesentlich stärker von der gewählten Struktur abhängig als ein MLP (s. Kapitel 8). Daher ist das Herausfinden der geeigneten Netzstruktur i.A. nicht unproblematisch.

Eine automatische Unterstüzung bei der Festlegung der Struktur eines ANFIS–Netzes wäre deshalb wünschenswert. Eventuell ist daher der Einsatz von anderen Systemen zur automatischen Generierung von Fuzzy–Regeln sinnvoll, um mit den erzeugten Regeln ein ANFIS–Netz zu erstellen und mit den ANFIS–Methoden weiter zu optimieren.

Das MFOS–S–System bietet alle Modifikationsarten, die prinzipiell möglich sind. Vorhandene Regeln werden korrigiert, nach Bedarf werden zusätzliche Regeln erzeugt bzw. überflüssige Regeln gelöscht. Ebenso ist das Erzeugen und Löschen von Partitions–Fuzzy–Mengen möglich.

Bei der Konfiguration eines MFOS–S–Netzes wird ein vom Anwender vorgegebener Sugeno–Controller übernommen und nach Bedarf optimiert. Dabei ist es unbedeutend, ob einzelne Regeln oder Partitions–Fuzzy–Mengen noch nicht definiert sind, oder nur vorhandene Partitions–Fuzzy–Mengen verändert werden müssen.

Beispiel 10.2 *Sei die gleiche Situation wie in Beispiel 7.1 gegeben. Die folgenden Fuzzy–Mengen und Regeln ergeben die Partitionierungen und die Regelbasis für einen einfachen Sugeno–Controller zur Steuerung eines Heizgeräts:*

Eingabe (Temperatur in °C): Dreiecks–Mengen auf dem Grundraum $[13, 23]$*:*

sehr kalt $\widehat{=} \widetilde{A}_1 = (13, 15, 17)$
kalt $\widehat{=} \widetilde{A}_2 = (16, 18, 20)$
warm $\widehat{=} \widetilde{A}_3 = (19, 21, 23)$

Die linguistischen Variablen sind x für die Temperatur und y für die Heizleistung.

Die richtigen Regeln sind:

IF $x =$ ***sehr kalt*** *THEN* $y = 8$
IF $x =$ ***kalt*** *THEN* $y = 5$
IF $x =$ ***warm*** *THEN* $y = 2$

Wird nun bei der Erstellung des MFOS–S–Systems die Eingabe–Partitions–Menge ***kalt*** *auf dem Eingaberaum und die Regel*

IF $x =$ ***kalt*** *THEN* $y = 5$

vergessen, so erzeugt das MFOS–S–Verfahren die fehlende Regel und eine neue Eingabe–Partitions–Menge

kalt $\widehat{=} \widetilde{A}_2 = (16.2, 18.0, 19.8)$

mit der die berechneten Ergebnisse genau so gut sind wie mit der ursprünglichen Menge $(16, 18, 20)$.

Die einzelnen Modifikationen werden nach Wahl des Benutzers in Abhängigkeit der Trainingsdaten durchgeführt. Zur Bewertung einer Regel wird der Fehler berücksichtigt, den die Regel verursacht. Neue Partitions–Mengen werden nach Bedarf erzeugt und von den Regeln verwendet. Dabei wird vermieden, mehrere ähnliche Partitions–Fuzzy–Mengen zu erzeugen, von denen eine genügen würde.

Zur Feinabstimmung der Parameter von Partitions–Mengen und Regeln werden die Modalwerte und Weiten sowie die reellen Konklusionen mit Hilfe des Backpropagation–Algorithmus angepaßt. Aufgrund der Berücksichtigung sämtlicher prinzipiell möglicher Modifikationen gibt es somit keine besonderen Einschränkungen oder Voraussetzungen für die Optimierung eines Sugeno–Controllers mit dem MFOS–S–System.

Insgesamt läßt sich folgendes Aussagen:

Mit dem MFOS–S–System lassen sich sämtliche Komponenten eines Sugeno–Controllers nach Bedarf optimieren. Eine konkrete Einschränkung oder notwendige Voraussetzung wie beim ANFIS–System läßt sich nicht feststellen. Daher ist zu erwarten, daß mit dem MFOS–S–System in nahezu allen Fällen die Optimierung eines vorgegebenen Sugeno–Controllers zu erreichen ist, unabhängig von der Konfiguration, die der Anwender vorgibt.

Das ANFIS–System ist von folgender Voraussetzung abhängig:

- für den erfolgreichen Einsatz des ANFIS–Systems sind bis auf Feinabstimmung korrekte Eingabe–Partitionierungen und Regeln unbedingt erforderlich

Diese Bedingungen müssen vom Anwender sichergestellt werden. Dabei sind eventuell andere Systeme zur automatischen Generierung von Fuzzy–Mengen oder Fuzzy–Regeln hilfreich. Wenn die genannten Bedingungen erfüllt werden, ist mit jedem dieser Systeme die Optimierung eines Sugeno–Controllers zu erreichen.

10.3 Vergleich gemeinsamer Verfahren

Beide vorgestellten Optimierungs–Systeme stellen Methoden zur Verfügung, Sugeno–Controller zu optimieren. Manche Methoden ähneln sich im Ziel und Durchführung, andere sind nur in einem System vorhanden. Zum Teil werden die gleichen Ziele mit unterschiedlichen Methoden realisiert. Welche Methoden sind besser geeignet? Lassen sich bestimmte Methoden eines Systems auf ein anderes System übertragen?

Modifikation von reellen Konklusions–Werten:

Das ANFIS–System und das MFOS–S–System ermöglichen das Ändern von Regel–Konklusionen. Beim ANFIS–System wird lediglich mit dem Backpropagation–Algorithmus bzw. einer linearen Methode der Wert der reellen Konklusionen verändert (s. Kapitel 8). Beim MFOS–S–System besteht zusätzlich zur Anwendung des Backpropagation–Algorithmus die Möglichkeit, unter Verwendung der Trainingsdaten gezielt eine neue Konklusion einzusetzen.

Der Vorteil des MFOS–S–Systems ist, daß bei fehlerhaften Konklusions–Werten in einem Schritt die korrekte Konklusion bestimmt wird. Das Backpropagation–Verfahren dient hier vorzugsweise der Feinabstimmung der Konklusions–Werte. Auch mit dem ANFIS–System ist eine korrekte Einstellung der Konklusions–Werte möglich. Durch die Kombination mit der linearen Methode ist das ANFIS–System schneller als bei ausschließlicher Verwendung von Backpropagation. Daher ist die potentiell mögliche Übertragung der MFOS–S–Methode zur direkten Festlegung der korrekten Konklusion nicht notwendig. Die Übertragung der linearen Methode auf das MFOS–S–System ist nicht möglich, da die Ausgabefunktion bei diesem System keine Linearkombination der Eingaben ist (vgl. Kapitel 8 und 9).

Modifikation von Eingabe–Partitions–Mengen

Beide vorgestellten Optimierungs–Systeme ermöglichen die Modifikation von Eingabe–Partitions–Mengen unter Verwendung des Backpropagation–Verfahrens. Dabei wird der Gradient jeweils mit den in den einzelnen Neuronen verwendeten (in jedem System unterschiedlichen) Funktionen berechnet. Eine Übertragung dieser Verfahren von einem System auf das andere ist daher offensichtlich nicht sinnvoll.

Weitere von mehreren der vorgestellten Optimierungs–Systemen durchgeführten Verfahren gibt es nicht. Die Möglichkeiten

- Neufestlegung von Regel–Konklusionen
- Erzeugen neuer Regeln
- Löschen unnötiger Regeln
- Erzeugen neuer Eingabe–Partitions–Mengen
- Löschen unnötiger Partitions–Fuzzy–Mengen

stellt nur das MFOS–S–System zur Verfügung. Daher gibt es keine weiteren Verfahren, die verglichen werden können.

10.4 Übertragung von Verfahren

Zum Abschluß dieser Untersuchungen soll nun überprüft werden, in wie weit sich die einzelnen Optimierungs–Verfahren auf das jeweils andere System übertragen lassen. Insbesondere geht es dabei um die Frage, ob sich die erwähnten Defizite der einzelnen Optimierungs–Systeme beheben lassen bzw. ob sich die Optimierungs–Ziele effektiver erreichen lassen.

Das ANFIS–System:

Mit dem ANFIS–System lassen sich keine neuen Eingabe–Partitions–Mengen und keine neuen Regeln erzeugen. Deshalb ist eine optimale Einstellung mit dem ANFIS–System nicht möglich, falls eine zur korrekten Steuerung unverzichtbare Eingabe–Partitions–Menge oder Regel nicht berücksichtigt wird (s. Beispiel 10.1). Daher würde eine Erweiterung um ein Verfahren zum Erzeugen von Regeln bzw. Eingabe–Partitions–Mengen die Einsatzmöglichkeiten dieses Optimierungs–Systems erweitern.

Da mit dem ANFIS–System Sugeno–Controller modelliert werden, werden prinzipiell keine Ausgabe–Partitionierungen definiert. Eine Übertragung der MFOS–S–Methoden zum Erzeugen von Regeln und Eingabe–Partitions–Mengen ist ohne große Modifikationen möglich. Ein ANFIS–Netz berechnet in Schicht 2 den Erfüllungsgrad der Prämisse der einzelnen Regeln. Somit ist es ohne besonderen Aufwand überprüfbar, ob bei einem Trainingsbeispiel der Erfüllungsgrad der Prämisse von jeder Regel unter einer Schranke liegt. In diesem Fall läßt sich eine neue Regel wie folgt erzeugen:

- Die Prämisse wird analog zur MFOS–S–Methode definiert. D.h. es werden diejenigen Eingabe–Partitions–Mengen ausgewählt, zu denen die Eingabe–Werte des aktuellen Trainingsbeispiels den höchsten Zugehörigkeitsgrad haben. Falls zu einem Eingabe–Wert der Zugehörigkeitsgrad zu jeder vorhandenen Eingabe–Partitions–Menge zu gering ist, wird analog zur MFOS–S–Methode mit Hilfe der Trainingsdaten eine geeignete Fuzzy–Menge erzeugt und in der Prämisse der zu erzeugenden Regel verwendet.

- Ebenso wird die Konklusion analog zur MFOS–S–Methode bestimmt. D.h. die gewünschte Ausgabe des aktuellen Trainingsbeispiels wird als reelle Konklusion der erzeugten Regel eingesetzt.

Auf diese Weise ist es auch mit dem ANFIS–System möglich, neue Eingabe–Partitions–Mengen und Regeln zu erzeugen. Damit sind die Einschränkungen für den Einsatz des ANFIS–Systems behoben.

Das MFOS–S–System:

Die Untersuchungen in Abschnitt 10.2 haben gezeigt, daß für die Optimierung eines Sugeno–Controllers mit dem MFOS–S–System keine besonderen Einschränkungen oder Voraussetzungen bestehen. Somit gibt es keinen grundsätzlichen Bedarf, die Einsatzmöglichkeiten dieses Optimierungs–Systems zu erweitern.

Dennoch stellt sich die Frage, ob es sinnvoll ist, eine Methode des ANFIS–Systems auf das MFOS–S–System zu übertragen. Möglicherweise ergeben sich dadurch effektivere Verfahren oder alternative Lösungswege.

Folgende Modifikationsmöglichkeiten werden vom MFOS–S–System und dem ANFIS–System unterstützt:

- Modifikation von reellen Konklusions–Werten
- Modifikation von Eingabe–Partitions–Mengen (Modalwert und Weite)

Wie bereits in Abschnitt 10.3 dargestellt wurde, passen beide vorgestellten Systeme die Eingabe–Partitions–Mengen mit Hilfe des Backpropagation–Verfahrens an. Da dieses Verfahren den Gradienten der in den einzelnen Neuronen verwendeten Funktionen benötigt und diese sich bei beiden Systemen unterscheiden, ist eine Übertragung nicht sinnvoll.

Das ANFIS–System adaptiert die reellen Konklusionen der Regeln mit dem Backpropagation–Algorithmus und mit einer linearen Methode. Voraussetzung für die Anwendung der linearen Methode ist, daß die in der Ausgabeschicht verwendete Ausgabefunktion als Linearkombination der zugehörigen Eingaben darstellbar ist. Da dies beim MFOS–S–System nicht der Fall ist, ist eine Übertragung dieser Methode auf das MFOS–S–System grundsätzlich nicht möglich. Somit ergeben sich keine Möglichkeiten, ein Verfahren des ANFIS–Systems auf das MFOS–S–System zu übertragen.

Fazit und Ausblick

In dieser Arbeit wurden verschiedene hybride Neuro–Fuzzy–Systeme zur Optimierung von Fuzzy–Controllern nach Mamdani bzw. Sugeno–Controllern vorgestellt und untersucht. Motiviert von den festgestellten einschränkenden Voraussetzungen für den Einsatz der untersuchten Systeme sowie Defiziten bei den verfügbaren Optimierungsmöglichkeiten wurde ein universelles konnektionistisches Modell entwickelt, welches nahezu keine einschränkenden Voraussetzungen mehr macht und jede Modifikationsmöglichkeit für die Regelbasis und die Partitionierungen eines Fuzzy–Controllers nach Mamdani bzw. eines Sugeno–Controllers zur Verfügung stellt: das Münsteraner–Fuzzy–Optimierungs–System MFOS.

Die wesentlichen Kriterien der durchgeführten Untersuchungen waren:

1. Einsetzbare Controller
2. Optimierungsmöglichkeiten
3. Vergleich gemeinsamer Verfahren
4. Übertragung von Verfahren

Punkt 1 und 2 sind primär aus der Sicht des Anwenders von Interesse, Punkt 3 und 4 betrachten Ansatzpunkte, bestehende Systeme zu verbessern.

1. Einsetzbare Controller

Für den Anwender ist es zunächst einmal wichtig, welche Systeme zur Optimierung eines Fuzzy–Controllers nach Mamdani bzw. Sugeno–Controllers eingesetzt werden können. Im Normalfall wählt der Anwender die Art des gewünschten Controllers aus. Damit ist automatisch vorgegeben, ob ein System zur Optimierung von Fuzzy–Controllern nach Mamdani oder ein System zur Optimierung von Sugeno–Controllern zu verwenden ist.

In einer typischen Anwendungssituation erstellt der Benutzer einen vorläufigen Controller, der von einem System optimiert werden soll. Dabei werden z.B. die Art der

verwendeten Fuzzy–Mengen, die Anzahl der Regeln und die Defuzzifizierungsmethode (außer bei Sugeno–Controllern) vom Benutzer ausgesucht. Falls ein System diese vom Anwender gewählten Bestandteile des Controllers nicht unterstützt, kann es nicht ohne Modifikationen des Controllers zur Optimierung eingesetzt werden. Unterstützt ein System z.B. nur Gauß–Mengen, läßt sich ein Controller mit Dreiecks–Mengen nicht äquivalent auf dieses System übertragen. Gleiches gilt, falls die Maximum–Methode zur Defuzzifizierung festgelegt wurde, das gewählte System jedoch ausschließlich die Schwerpunkt–Methode verwendet.

Daher müssen abhängig vom verwendeten Optimierungs–System einige Voraussetzungen an den Controller erfüllt werden, damit eine Übertragung auf das System durchführbar ist. Dies schränkt die Einsatzmöglichkeiten ein, falls ein bereits erstellter Controller zu optimieren ist. Im einzelnen müssen die folgenden Punkte beachtet werden:

- Das Verfahren von Lin und Lee stellt ausschließlich die Schwerpunkt–Methode zur Verfügung.
- Das NEFCON–Modell verwendet Dreiecks–Mengen für die Eingabe–Dimensionen und Zacken–Mengen für die Ausgabe–Dimension. Die Stellwerte müssen normiert sein. Die Ausgabe wird als gewichtete Summe mit Hilfe der Umkehrabbildung der Zugehörigkeitsfunktionen der Ausgabe–Partitions–Mengen berechnet.
- Das MFOS–M–System macht keine besonderen Voraussetzungen.
- Das ANFIS–System macht keine besonderen Voraussetzungen.
- Das MFOS–S–System macht keine besonderen Voraussetzungen.

Es zeigt sich, daß lediglich das NEFCON–Modell nicht unerhebliche Voraussetzungen an den zu optimierenden Fuzzy–Controller macht. Das Verfahren von Lin und Lee stellt ausschließlich die Schwerpunkt–Methode zur Verfügung, dies ist in manchen Fällen ein Nachteil. Die anderen Systeme ermöglichen die Verwendung eines nahezu beliebigen, vorgegebenen Controllers.

2. Optimierungsmöglichkeiten:

Sämtliche vorgestellten Optimierungs–Systeme übertragen einen Fuzzy–Controller nach Mamdani bzw. einen Sugeno–Controller auf ein neuronales Netz, um dieses zu trainieren. Durch spezielle Lernverfahren werden die einzelnen Bestandteile des Controllers angepaßt. Jedoch ist nicht mit jedem System jede mögliche Modifikation des Controllers durchführbar. Daher ist in manchen Fällen eine erfolgreiche Optimierung nicht möglich. Dies ist z.B. der Fall, wenn eine wichtige Regel oder Partitions–Menge bei der Konfiguration des Controllers nicht berücksichtigt wurde und das Optimierungs–System keine neuen Regeln oder Partitions–Mengen erzeugen kann.

Folgende Modifikationen sind mit den einzelnen Optimierungs–Systemen prinzipiell durchführbar:

- Verfahren von Lin und Lee
  - Neufestlegung von Regel–Konklusionen
  - Erzeugen neuer Ausgabe–Partitions–Mengen
  - Modifikation von Eingabe–Partitions–Mengen (Modalwert und Weite)
  - Modifikation von Ausgabe–Partitions–Mengen (Modalwert und Weite)
- NEFCON–Modell
  - Erzeugen neuer Regeln
  - Modifikation von Eingabe–Partitions–Mengen (nur Weite)
  - Modifikation von Ausgabe–Partitions–Mengen (nur Weite)
- MFOS–M–System
  - Neufestlegung von Regel–Konklusionen
  - Erzeugen neuer Regeln
  - Löschen unnötiger Regeln
  - Erzeugen neuer Eingabe–Partitions–Mengen
  - Erzeugen neuer Ausgabe–Partitions–Mengen
  - Löschen unnötiger Partitions–Fuzzy–Mengen
  - Modifikation von Eingabe–Partitions–Mengen (Modalwert und Weite)
  - Modifikation von Ausgabe–Partitions–Mengen (Modalwert und Weite)
- ANFIS–System
  - Modifikation von reellen Konklusions–Werten
  - Modifikation von Eingabe–Partitions–Mengen (Modalwert und Weite)
  - Ausgabe–Partitions–Mengen sind definitionsbedingt nicht vorhanden (Sugeno–Controller)
- MFOS–S–System
  - Neufestlegung von Regel–Konklusionen
  - Erzeugen neuer Regeln
  - Löschen unnötiger Regeln
  - Erzeugen neuer Eingabe–Partitions–Mengen

- Löschen unnötiger Partitions–Fuzzy–Mengen
- Modifikation von reellen Konklusions–Werten
- Modifikation von Eingabe–Partitions–Mengen (Modalwert und Weite)
- Ausgabe–Partitions–Mengen sind definitionsbedingt nicht vorhanden (Sugeno–Controller)

Es zeigt sich, daß lediglich das MFOS–M– und das MFOS–S–System sämtliche prinzipiell möglichen Modifikationen der Bestandteile des zu optimierenden Controllers durchführen können. Bei den anderen Systemen führt insbesondere die fehlende Möglichkeit, neue Regeln bzw. neue Partitions–Mengen zu erzeugen, dazu, daß in einigen Fällen eine Optimierung des vorgegebenen Controllers scheitert.

Daher ist nur das MFOS–M– bzw. MFOS–S–System ohne besondere Einschränkungen geeignet, in nahezu allen Fällen einen beliebigen, vorgegebenen Controller zu optimieren. Bei den anderen vorgestellten Systemen müssen die folgenden Voraussetzungen erfüllt werden:

- Für den erfolgreichen Einsatz des Verfahrens von Lin und Lee müssen die Anzahl der Eingabe–Partitions–Mengen und die Anzahl der notwendigen Regeln vom Anwender korrekt festgelegt werden.
- Für den erfolgreichen Einsatz des NEFCON–Modells müssen bis auf Feinabstimmung die Partitionierungen vom Anwender korrekt vorgegeben werden.
- Für den erfolgreichen Einsatz des ANFIS–Systems müssen die Anzahl der notwendigen Regeln und Eingabe–Fuzzy–Mengen vom Anwender korrekt festgelegt werden.

Die Erfüllung dieser Voraussetzungen ist jedoch nicht trivial. Daher ist es empfehlenswert, zunächst andere Systeme zur Generierung von Regeln bzw. Partitionierungen einzusetzen und anschließend das Verfahren von Lin und Lee, das NEFCON–Modell oder das ANFIS–System zur weiteren Optimierung einzusetzen.

3. Vergleich gemeinsamer Verfahren

Da es Optimierungs–Systeme gibt, die die gleichen Optimierungsziele mit unterschiedlichen Methoden erreichen, wurden diese Methoden miteinander verglichen.

Das Verfahren von Lin und Lee und das MFOS–M–System ermöglichen das Ändern von Regel–Konklusionen und das Erzeugen neuer Ausgabe–Partitions–Mengen. Das Verfahren von Lin und Lee erzeugt mit Hilfe des Backpropagation–Verfahrens eine ganze Anzahl neuer Ausgabe–Partitions–Mengen, bis die optimale Menge gefunden

wurde. Dies führt dazu, daß viele eigentlich unnötige Fuzzy–Mengen erzeugt werden. Das MFOS–M–System wählt gezielt eine geeignete Partitions–Menge aus bzw. erzeugt bei Bedarf sofort die benötigte Fuzzy–Menge. Daher ist diese Methode effektiver.

Das NEFCON–Modell erzeugt neue Regeln durch Kombination aller vorhandenen Partitions–Mengen. Daher ist es abhängig von korrekt vorgegebenen Partitionierungen. Das MFOS–M–System erzeugt bei Bedarf gezielt zusätzlich benötigte Regeln. Dies scheint weniger umständlich als die NEFCON–Methode.

Weiterhin führen sämtliche vorgestellten Systeme Modifikationen der Partitions–Mengen bzw. der reellen Regel–Konklusionen (Sugeno–Controller) durch. Das NEFCON–Modell verwendet dazu ein Fuzzy–Fehlermaß, die anderen Systeme das Backpropagation–Verfahren bzw. Modifikationen von Backpropagation.

4. Übertragung von Verfahren

Nicht jedes vorgestellte Optimierungs–System stellt jede mögliche Modifikation der Bestandteile des zu optimierenden Controllers zur Verfügung. Daher wurde untersucht, ob sich einzelne Verfahren eines Systems auf ein anderes System übertragen lassen, um vorhandene Defizite zu beheben bzw. höhere Effektivität zu erreichen. Insbesondere die folgenden Verbesserungen sind dazu geeignet:

- Mit dem Verfahren von Lin und Lee lassen sich keine neuen Regeln und keine Eingabe–Partitions–Mengen erzeugen. Daher wäre es sinnvoll, die NEFCON–Methode zum Erzeugen von Regeln bzw. die MFOS–M–Methoden zum Erzeugen von Regeln und Partitions–Mengen auf das Verfahren von Lin und Lee zu übertragen.
- Mit dem NEFCON–Modell lassen sich keine neuen Partitions–Mengen erzeugen. Daher wäre es sinnvoll, die MFOS–M–Methode zum Erzeugen neuer Partitions–Mengen auf das NEFCON–Modell zu übertragen.
- Mit dem ANFIS–System lassen sich keine neuen Regeln und keine Eingabe–Partitions–Mengen erzeugen. Daher wäre es sinnvoll, die MFOS–S–Methoden zum Erzeugen von Regeln und Partitions–Mengen auf das ANFIS–System zu übertragen.

Auf diese Weise lassen sich die wesentlichen Einschränkungen für den erfolgreichen Einsatz dieser Verfahren beseitigen.

Fazit:

Neben der Vorstellung und den Untersuchungen hybrider Neuro–Fuzzy–Systeme war ein zentraler Bestandteil dieser Arbeit die Entwicklung des universellen konnektionistischen Modelles MFOS, des Münsteraner Fuzzy–Optimierungs–Systems. Motiviert von den einschränkenden Voraussetzungen und Defiziten bekannter Systeme war es das Ziel, dem Anwender ein Werkzeug zur Verfügung zu stellen, um einen beliebigen, vorgegebenen Fuzzy–Controller nach Mamdani bzw. Sugeno–Controller auf ein funktional äquivalentes neuronale Netz zu übertragen und die Bestandteile des Controllers nach Wunsch zu optimieren.

Die Untersuchungen haben gezeigt, daß dieses Ziel erreicht wurde. Mit dem MFOS–M– bzw. MFOS–S–System werden im Gegensatz zu den anderen vorgestellten Systemen keine besonderen Voraussetzungen an den zu optimierenden Controller gemacht. Nur mit dem MFOS–M– bzw. MFOS–S–System lassen sich sämtliche möglichen Modifikationen der Bestandteile des Controllers durchführen. Daher ist zu erwarten, daß mit dem MFOS–M– bzw. MFOS–S–System in nahezu allen Fällen die Optimierung eines vorgegebenen Fuzzy–Controllers zu erreichen ist. Somit sind das MFOS–M– und das MFOS–S–System geeignete Modelle zur Optimierung von Fuzzy–Controllern, die den anderen vorgestellten Modellen in einigen wesentlichen Punkten überlegen sind.

Ausblick:

Für das MFOS–M– und das MFOS–S–Netz wurden verschiedene Lernverfahren entwickelt. Neben dem Gradientenabstiegsverfahren und seinen Modifikationen stellen insbesondere die Pre–Tuning–Verfahren neue Ansätze für Lernalgorithmen zur Verfügung. Sämtliche Pre–Tuning–Verfahren wurden speziell für ein MFOS–Netz entwickelt. Dabei wurde das konkrete Ziel, einen Fuzzy–Controller zu optimieren, stets berücksichtigt. D.h. Ausgangspunkt bei der Entwicklung von jedem einzelnen dieser Verfahren war die Optimierung eines konkreten Bestandteils eines Fuzzy–Controllers unter Einbeziehung der verwendeten speziellen Netzarchitekturen. Es bleibt zu untersuchen, ob sich dennoch ausgesuchte Pre–Tuning Verfahren auf andere (herkömmliche) neuronale Netze übertragen lassen.

Da das Thema dieser Arbeit die Optimierung von Fuzzy–Controllern ist, soll dies als Anregung für weitere Arbeiten verstanden werden.

Mit der Entwicklung des MFOS–M– und MFOS–S–Systems ist das wesentliche Ziel dieser Arbeit erreicht. Die Übertragung eines vorgegebenen Fuzzy–Controllers auf ein funktional äquivalentes neuronales Netz sowie die Optimierung der Bestandteile dieses Controllers sind mit den beiden Systemen in nahezu allen Fällen erfolgreich durchführbar.

Anleitung zum MFOS–Programm

.1 MFOS–M–System

Das MFOS–M–System wurde objektorientiert in C++ implementiert. Da bei der Entwicklung des Systems die Algorithmen im Vordergrund standen, wurde zunächst auf eine GUI verzichtet. Die Entwicklung einer GUI war Bestandteil der Diplomarbeiten von M. Deimann [Deimann, 2002] und von H. Brinkschulte [Brinkschulte, 2002]. Die Optimierung dieser GUI sowie die Kombination mit dem MFOS–S–System ist Bestandteil der Diplomarbeiten von J. Thielemeyer und W. Urich, die zum Zeitpunkt der Erstellung dieser Arbeit noch nicht vorliegen.

Auf der beiliegenden CD befinden sich die MFOS–M–Versionen mit GUI von H. Brinkschulte und M. Deimann in den Verzeichnissen `niendieck/jmfos/brinkschulte` bzw. `niendieck/jmfos/deimann`.

Sämtliche C++ Dateien für das MFOS–M–System befinden sich auf der beigefügten CD unter `niendieck/mfos_m`. Im folgenden wird zunächst die Anwendung des menügesteuerten MFOS–M–Systems beschrieben:

Das Hauptprogramm ist `mfos_mamdani.cpp`

Folgende Klassen werden eingefügt und enthalten das Bildschirmmenü, die Partitionierungen, Neuronen, Regeln, Datensätze und die Lernverfahren:

```
menue.h, partmengen.h, neuronen.h, regelbas.h, datensae.h,
entfer.h, neureg.h, korreg.h, zusammen.h
```

Das Hauptprogramm wird unter Sun Solaris wie folgt compiliert:

```
environ SunONE <RETURN>

CC -o mfos_mamdani.exe mfos_mamdani.cpp <RETURN>
```

Gestartet wird das Programm durch Aufruf von

```
mfos_mamdani.exe <RETURN>
```

Daraufhin erscheint das Hauptmenü:

Hauptmenü – MFOS–Mamdani

(L) Laden
(S) Speichern
(A) Anzeigen
(R) Regelbasis einfügen
(M) Methode zur Defuzzifizierung festlegen
(E) Eingabe eines Datensatzes in das Netz
(D) Datensatz aus der Datei in das Netz einfügen
(B) Berechnung der Ausgabe
(K) Komplette Berechnung für alle Datensätze
(T) Training
(H) Halt - Programm beenden

Sämtliche Menüpunkte werden durch Eingabe des Buchstabens und (RETURN) aufgerufen.

(L)aden und (S)peichern dient zum Laden bzw. Speichern von Partitionierungen, Regeln und Datensätzen. (A)nzeigen zeigt die entsprechenden Werte am Bildschirm an. (R)egelbasis einfügen initialisiert das Netz mit einer geladenen Regelbasis und den geladenen Partitionierungen. (R) muß immer aufgerufen werden, nachdem Regeln oder Partitionierungen geladen wurden. (M)ethode zur Defuzzifizierung festlegen erlaubt die Wahl zwischen (S)chwerpunkt–Methode und (M)aximum–Methode.

(E)ingabe eines Datensatzes ermöglicht die Eingabe von eigenen Datensätzen, die nicht in einer Datei vorliegen. (D)atensatz aus der Datei in das Netz einfügen fügt einen Datensatz aus einer geladenen Datei als Eingaben in das Netz ein.

(B)erechnung der Ausgabe berechnet die Ausgabe eines zuvor von Hand eingegebenen oder aus einer Datei eingefügten Datensatzes. (K)omplette Berechnung berechnet der Reihe nach zu allen geladenen Datensätzen die Ausgaben. Das Ergebnis kann danach durch (A)nzeigen: (D)atensätze angezeigt werden.

(T)raining führt zum Trainingsmenü:

(V) Voreinstellungen für Training
(N) Neue Regeln erzeugen
(K) Korrigieren von Regeln
(E) Entfernen von Regeln
(P) Partitions–Mengen zusammenfassen

(B) Backpropagation durchführen
(Z) Zurück zum Hauptmenü

(V)oreinstellungen dient der Eingabe der Trainings–Parameter Erfüllungsgrad–Grenze, Fehler–Grenze, Aeta (Backpropagation), Wiederholungen sowie der Festlegung, ob neue Partitions–Mengen erzeugt werden dürfen oder nicht. (Die Bedeutung dieser Parameter wird bei der Beschreibung der einzelnen Lernverfahren erklärt.)

Die Punkte (N) Neue Regeln erzeugen, (K) Korrigieren von Regeln, (E) Entfernen von Regeln, (P) Partitions–Mengen zusammenfassen und (B) Backpropagation durchführen starten die einzelnen Lernverfahren.

Beispiel .1 *Auf der beigefügten CD befindet sich folgendes Beispiel, das hier schrittweise vorgeführt wird:*

1. (L)aden: (E)ingabe–Partitionierungen `beispielein`
2. (L)aden: (A)usgabe–Partitionierungen `beispielaus`
3. (L)aden: (R)egelbasis `beispielreg`
4. (L)aden: (D)atensätze `beispielsatz`

5. (R)egelbasis einfügen

6. (A)nzeigen: (E)ingabe–Partionierungen:

Dreiecks–Mengen (m,w): (14, 2), (19, 2), (22, 2)

7. (A)nzeigen: (A)usgabe–Partionierungen:

Dreiecks–Mengen (m,w): (1, 2), (4, 2), (7, 2)

8. (A)nzeigen: (R)egelbasis: 1 | 1 | 3, 2 | 1 | 2

Die erste Zahl bezeichnet die Nr. der Eingabe–Partitions–Menge, die zweite Zahl die Nr. der zugehörigen Ausgabe–Dimension und die letze Zahl die Nr. der Ausgabe–Partitions–Menge. D.h. die Regeln sind:

```
IF x = (14, 2) THEN y = (7, 2)
IF x = (19, 2) THEN y = (4, 2)
```

9. (K)omplette Berechnung

10. (A)nzeigen: (D)atensätze: Eingabe | gewünschte Ausgabe | berechnete Ausgabe: 15 | 8 | 7, 18 | 5 | 4, 21 | 2 | 0

Die Fehler beim ersten und zweiten Datensatz werden durch die ungünstigen Ausgabe–Partitions–Mengen verursacht, der Fehler beim dritten Datensatz entsteht, weil keine Regel zur Eingabe 21 vorhanden ist. Deshalb soll nun das Netz trainiert werden:

11. (T)raining: (V)oreinstellungen:
(E)rfüllungsgrad–Grenze = 0.5, (F)ehler–Grenze = 1

12. (T)raining: (N)eue Regeln erzeugen erzeugt die fehlende Regel

13. (R)egelbasis einfügen erstellt das neue Netz

14. (A)nzeigen: (R)egelbasis zeigt die Regeln an: 3 | 1 | 1 steht für die neue Regel:

```
IF x = (22, 2) THEN y = (1, 2)
```

15. (K)omplette Berechnung

16. (A)nzeigen Datensätze:

15 | 8 | 7, 18 | 5 | 4, 21 | 2 | 1

Somit ist der Fehler bei Eingabe 21 verbessert worden, da nun eine passende Regel zu dieser Eingabe vorhanden ist. Die weitere Optimierung erfolgt nun mit dem Backpropagation–Verfahren:

17. (T)raining: (V)oreinstellungen: (A)eta jeweils 0.1, (W)iederholungen 100

18. (T)raining: (B)ackpropagation durchführen

19. (K)omplette Berechnung

20. (A)nzeigen: (A)usgabe–Partitionierungen zeigt die optimierten Ausgabe–Partitionierungen:

Dreiecks–Mengen (m,w): (1.99997, 2), (4.99997, 2), (7.99997, 2)

21. (A)nzeigen: (D)atensätze zeigt die Datensätze mit den berechneten Ausgaben, die nun alle nahezu korrekt sind:

15 | 8 | 7.99997, 18 | 5 | 4.99997, 21 | 2 | 1.99997

Somit ist nun das Netz optimal eingestellt. Zunächst wurde eine fehlende Regel erzeugt, anschließend die ungünstigen Ausgabe–Partitions–Mengen angepaßt.

Mit den Hilfsprogrammen `erzpart.exe`, `erzreg.exe` und `erzsatz.exe` lassen sich weitere Beispiele von Partitionierungen, Regeln und Datensätzen erzeugen.

Eine MFOS–M–Version mit GUI bzw. eine gemeinsame Version von MFOS–M und MFOS–S mit GUI werden wie oben erwähnt zum Zeipunkt der Erstellung dieser Arbeit entwickelt.

In Zukunft werden diese Versionen zugänglich sein unter:

`http://wwwmath.uni-muenster.de/info/Professoren/Lippe`
`/lehre/skripte/index.html`

Ein Beispiel zum Programm von H. Brinkschulte befindet sich auf der CD unter `niendieck/jmfos/brinkschulte`.

Aufgerufen wird das Programm unter Sun Solaris durch:

```
environ Java_1.3.1 <RETURN>

java -jar mfos.jar <RETURN>
```

Im Verzeichnis `niendieck/jmfos/brinkschulte/beispiele` befinden sich folgende Beispieldateien, die unter dem Menüpunkt `Datei` nacheinander geladen werden:

Datensätze: `beispieldat.dtn`
Eingabe–Partitionierung: `beispielein.ept`
Ausgabe–Partitionierung: `beispielaus.apt`
Regelbasis: `beispielreg.rba`

Da das Programm keine Daten von der CD lesen kann, müssen diese zuvor in ein eigenes Verzeichnis kopiert werden.

Mit `Berechnung: Komplette Berechnung` werden die Ausgaben berechnet.

Mit `Training: Voreinstellungen` werden die Fehler–Grenze auf 1 und die Erfüllungsgrad–Grenze auf 0.5 festgelegt.

`Training: Neue Regeln erzeugen` erzeugt die fehlende Regel analog zum vorherigen Beispiel.

Eine Anleitung zu dieser MFOS–Version mit GUI findet sich unter:

`http://wwwmath.uni-muenster.de/info/Professoren/Lippe/`
`diplomarbeiten/html/brinkschulte/Handbuch_html/index.htm`

.2 MFOS–S–System

Das MFOS–S–System wurde objektorientiert in C++ implementiert. Da bei der Entwicklung des Systems die Algorithmen im Vordergrund standen, wurde wie beim MFOS–M–System zunächst auf eine GUI verzichtet. Die Erstellung einer GUI und das Kombinieren der beiden Systeme ist Bestandteil der Diplomarbeiten von J. Thielemeyer und W. Urich, die zum Zeitpunkt der Erstellung dieser Arbeit noch nicht vorliegen. Daher ist das MFOS–S–Programm menügesteuert.

Sämtliche C++ Dateien befinden sich auf der beigefügten CD unter `mfos_s`.

Das Hauptprogramm ist `mfos_sugeno.cpp`

Folgende Klassen werden eingefügt und enthalten das Bildschirmmenü, die Partitionierungen, Neuronen, Regeln, Konklusionen, Datensätze und die Lernverfahren:

`menue.h, partmengen.h, neuronen.h, regelbas.h, konklus.h, datensae.h, entfer.h, neureg.h, korreg.h, zusammen.h`

Das Hauptprogramm wird unter Sun Solaris wir folgt compiliert:

```
environ SunONE <RETURN>

CC -o mfos_sugeno.exe mfos_sugeno.cpp <RETURN>
```

Gestartet wird das Programm durch Aufruf von

```
mfos_sugeno.exe <RETURN>
```

Daraufhin erscheint das Hauptmenü:

Hauptmenü – Münsteraner Fuzzy Optimierungs System MFOS–S

(L) Laden
(S) Speichern
(A) Anzeigen
(R) Regelbasis einfügen
(E) Eingabe eines Datensatzes in das Netz
(D) Datensatz aus der Datei in das Netz einfügen
(B) Berechnung der Ausgabe
(K) Komplette Berechnung für alle Datensätze
(T) Training
(H) Halt - Programm beenden

Sämtliche Menüpunkte werden durch Eingabe des Buchstabens und (RETURN) aufgerufen.

(L)aden und (S)peichern dient zum Laden bzw. Speichern von Partitionierungen, Regeln und Datensätzen. (A)nzeigen zeigt die entsprechenden Werte am Bildschirm an. (R)egelbasis einfügen initialisiert das Netz mit einer geladenen Regelbasis und den geladenen Partitionierungen. (R) muß immer aufgerufen werden, nachdem Regeln oder Partitionierungen geladen wurden.

(E)ingabe eines Datensatzes ermöglicht die Eingabe von eigenen Datensätzen, die nicht in einer Datei vorliegen. (D)atensatz aus der Datei in das Netz einfügen fügt einen Datensatz aus einer geladenen Datei als Eingaben in das Netz ein.

(B)erechnung der Ausgabe berechnet die Ausgabe eines zuvor von Hand eingegebenen oder aus einer Datei eingefügten Datensatzes. (K)omplette Berechnung berechnet der Reihe nach zu allen geladenen Datensätzen die Ausgaben. Das Ergebnis kann danach durch (A)nzeigen: (D)atensätze angezeigt werden.

(T)raining führt zum Trainingsmenü:

(V) Voreinstellungen für Training
(N) Neue Regeln erzeugen
(K) Korrigieren von Regeln
(E) Entfernen von Regeln
(P) Partitions–Mengen zusammenfassen
(B) Backpropagation durchführen
(M) Mitteln der Partitions–Mengen
(Z) Zurück zum Hauptmenü

(V)oreinstellungen dient der Eingabe der Trainings–Parameter Erfüllungsgrad–Grenze, Fehler–Grenze, Varianz–Grenze, Aeta (Backpropagation), Wiederholungen sowie der Festlegung, ob neue Partitions–Mengen erzeugt werden dürfen oder nicht. (Die Bedeutung dieser Parameter wird bei der Beschreibung der einzelnen Lernverfahren erklärt.)

Die Punkte (N) Neue Regeln erzeugen, (K) Korrigieren von Regeln, (E) Entfernen von Regeln, (P) Partitions–Mengen zusammenfassen und (B) Backpropagation durchführen starten die Lernverfahren. (M)itteln führt das abschließende Mitteln der gekoppelten Gewichte durch.

Beispiel .2 *Auf der beigefügten CD befindet sich folgendes Beispiel, das hier schrittweise vorgeführt wird:*

1. (L)aden: (E)ingabe–Partitionierungen `beispielpart`
2. (L)aden: (R)egelbasis `beispielreg` *und Konklusionen* `beispielkonk`

3. (L)aden: (D)atensätze `beispielsatz`

4. (R)egelbasis einfügen

5. (A)nzeigen: (E)ingabe–Partionierungen:

Dreiecks–Mengen (m,w): (14, 2), (19, 2), (22, 2)

6. (A)nzeigen: (R)egelbasis: 1 | 1 | 7, 2 | 1 | 4

Die erste Zahl bezeichnet die Nr. der Partitons–Menge, die zweite Zahl die Nr. der zugehörigen Ausgabe–Dimension und die letze Zahl die reelle Konklusion. D.h. die Regeln sind:

```
IF x = (14, 2) THEN y = 7
IF x = (19, 2) THEN y = 4
```

7. (K)omplette Berechnung

8. (A)nzeigen: (D)atensätze: Eingabe | gewünschte Ausgabe | berechnete Ausgabe:

15 | 8 | 7, 18 | 5 | 4, 21 | 2 | 0

Die Fehler beim ersten und zweiten Datensatz werden durch die ungünstigen Konklusionen verursacht, der Fehler beim dritten Datensatz entsteht, weil keine Regel zur Eingabe 21 vorhanden ist. Deshalb soll nun das Netz trainiert werden:

9. (T)raining: (V)oreinstellungen:
(E)rfüllungsgrad–Grenze = 0.5, (F)ehler–Grenze = 1

10. (T)raining: (N)eue Regeln erzeugen erzeugt die fehlende Regel

11. (R)egelbasis einfügen erstellt das neue Netz

12. (A)nzeigen: (R)egelbasis zeigt die Regeln an: 3—1—2 steht für die neue Regel:

```
IF x = (22, 2) THEN y = 2
```

13. (K)omplette Berechnung

14. (A)nzeigen Datensätze:

15 | 8 | 7, 18 | 5 | 4, 21 | 2 | 2

Somit ist der Fehler bei Eingabe 21 verbessert worden, da nun eine passende Regel zu dieser Eingabe vorhanden ist. Die weitere Optimierung erfolgt nun mit dem Backpropagation–Verfahren:

15. (T)raining: (V)oreinstellungen: (A)eta jeweils 0.1, (W)iederholungen 10
16. (T)raining: (B)ackpropagation durchführen

17. (R)egelbasis einfügen

18. (K)omplette Berechnung

19. (A)nzeigen: (R)egelbasis zeigt die endgültige Regelbasis:

```
IF x = (14, 2) THEN y = 8
IF x = (19, 2) THEN y = 5
IF x = (22, 2) THEN y = 2
```

20. (A)nzeigen: (D)atensätze zeigt die Datensätze mit den berechneten Ausgaben, die nun alle korrekt sind:

15 | 8 | 8, 18 | 5 | 5, 21 | 2 | 2

Somit ist nun das Netz optimal eingestellt. Zunächst wurde eine fehlende Regel erzeugt, anschließend die ungünstigen Konklusions–Werte der vorhandenen Regeln angepaßt.

Mit folgenden Hilfsprogrammen lassen sich weitere Beispiele von Partitionierungen, Regeln, Konklusionen und Datensätzen erzeugen:

`erzpart.exe`, `erzreg,exe`, `erzkonk.exe` und `erzsatz.exe`.

Abbildungsverzeichnis

Index

Literaturverzeichnis

[Black, 1937] M. Black:
Vaguenes: An exercise in logical analysis,
Philosophy of Science 4, pp. 427-455 (1937)

[Brinkschulte, 2002] H. Brinkschulte:
Einbindung von Fuzzy–Implikationen
und Defuzzifizierungsmethoden in das MFOS,
Diplomarbeit, Westf. Wilhelms–Universität Münster (2002)

[Brockhaus, 1998] Der Brockhaus in 15 Bänden,
5. Band, F.A. Brockhaus GmbH (1998)

[Buckley, 1993] J.J. Buckley:
Sugeno–type controllers are universal approximators,
Fuzzy sets and systems 53, pp. 299-303 (1993)

[CanoNava, 2002] J.C. Cano, P. Nava:
A Fuzzy Method for Automatic Generation Of Membership Function
Using Fuzzy Relations from Training Examples,
Proc. of the 2002 Annual Meeting of the North American
Fuzzy Information Processing Society,
New Orleans, LA, USA, pp. 158-162 (2002)

[ChangSun, 2003] W.-J. Chang, C.-C. Sun:
Constrained fuzzy controller design of discrete Takagi–Sugeno fuzzy models,
Fuzzy sets and systems 133, pp. 37-55 (2003)

[Ciftcioglu, 2002] Ö. Ciftcioglu:
Ordering Rules and Complexity Reduction for Fuzzy Models,
Proc. of the 2002 Annual Meeting of the North American
Fuzzy Information Processing Society,
New Orleans, LA, USA, pp. 535-540 (2002)

[Deimann, 2002] M. Deimann:
Visualisierung einer grafischen Benutzeroberfläche für einen Fuzzy Optimierer,
Diplomarbeit, Westf. Wilhelms–Universität Münster (2002)

[Fahlman, 1988] S.E. Fahlman:
An empirical study of learning speed in back–propagation networks,
in D. Touretzky, G. Hinton, T. Sejnowski (Eds.):
Proc. of the 1988 Connectionist Models Summer School,
Carnegie Mellon University (1988)

[Feuring, 1996] T. Feuring:
Fuzzy–Systeme, Vorlesungsscript,
Westf. Wilhelms–Universität Münster (1996)
Zugänglich unter: http://wwwmath.uni-muenster.de/math/inst/
info/Professoren/Lippe/lehre/skripte/index.html

[Fukushima, 1980] K. Fukushima:
Neocognitron: A self–organizing neural network model for a mechanism
of pattern recognition unaffected by shift in position,
Biological Cybernetics 20, pp. 121-136 (1980)

[Grossberg, 1976] S. Grossberg:
Adaptive pattern classification and universal recoding:
I. Parallel development and coding of neural feature detectors,
Biological Cybernetics 23, pp. 121-134 (1976)

[Kruse, 1995] R. Kruse, J. Gebhardt, F. Klawonn:
Fuzzy–Systeme,
Teubner (1995)

[Hebb, 1949] D.O. Hebb:
The Organization of Behaviour,
Wiley, New York, Introduction and Chapter 4:
The first stage of perception: growth of an assembly, pp. xi-xix, 60-78 (1949)

[Hecht–Nielsen, 1987] R. Hecht–Nielsen:
Kolmogorov's mapping neural network existence theorem,
Proc. of the 1st IEEE Int. Conf. on Neural Networks, 3, pp. 11-14 (1987)

[Hecht–Nielsen, 1991] R. Hecht–Nielsen:
Neurocomputing,
Addison–Wesley (1991)

[Heinemann, 1992] B. Heinemann, K. Weihrauch:
Logik für Informatiker,
Teubner (1992)

[Hopfield, 1982] J.J. Hopfield:
Neural Networks and physical systems with emergent collective
computational abilities,
Proc. of the National Academy of Sciences, USA, Vol. 79, pp. 2554-2558 (1982)

[Jacobs, 1988] R. A. Jacobs:
Increased Rates of Convergence Through Learning Rate Adaption,
Neural Networks 1/1988, pp. 295-307 (1988)

[Jang, 1993] R. Jang, J. Shing:
ANFIS: Adaptive–Network–Based Fuzzy Inference System,
IEEE Trans. on Systems, Man and Cybernetics, Vol. 23, no. 3, pp. 665-685 (1993)

[Kohonen, 1972] T. Kohonen:
Correlation Matrix Memories,
IEEE Transactions on Computers C-21, pp. 353-359 (1972)

[Kohonen, 1982] T. Kohonen:
Self–organized formation of topologically correct feature maps,
Biological Cybernetics 43, pp. 59-69 (1982)

[LiDengWei, 2002] Y. Li, J.-M. Deng, M.-Y. Wei:
Meaning and precision of adaptive fuzzy systems with
Gaussian–type membership functions,
Fuzzy sets and systems 127, pp. 85-97 (2002)

[LinLee, 1991] C.T. Lin, C.S.G. Lee:
Neural–network–based fuzzy logic controll and decision system,
IEEE Transactions on Computers, Vol. C-40,
Nr. 12, pp. 1320-1336, Dec. 1991 (1991)

[Lippman, 1991] S.B. Lippman:
C++, Einführung und Leitfaden,
Addison–Wesley (1991)

[LitRez, 2000] A. Little, L. Reznik:
Implementation of Fuzzy Controllers with Radial Basis Neural Networks,
Proc. of the 9th IEEE Int. Conf. on Fuzzy Systems,
San Antonio, Texas, USA, pp. 581-586 (2000)

[Lukasiewicz, 1957] J. Lukasiewicz:
Aristoteles syllogistic: From the standpoint of modern formal logic,
Clarendon Press Oxford, 2. Ed. (1957)

[Mamdani, 1974] E. H. Mamdani:
Applications of Fuzzy Algorithms for Simple Dynamic Plant,
Proc. of the IEEE, Volume 121, pp. 1585-1588 (1974)

[Mamdani, 1975] E. H. Mamdani:
An experiment in linguistic synthesis with a fuzzy logic controller,
Int. Journal of Man–Maschines Studies 7, pp. 1-13 (1975)

[MatAlJim, 2002] F. Matia, B.M. Al-Hadithi, A. Jimenez:
Generalization of stability criterion for Takagi–Sugeno continuous fuzzy model,
Fuzzy sets and systems 129, pp. 295-309 (2002)

[McCPit, 1943] W. S. McCulloch, W. Pitts:
A logical calculus of the ideas immanent in nervous activity,
Bulletin of Mathematical Biophysics 5, pp. 115-113 (1943)

[Mendel, 1992] J.M. Mendel, L.-X. Wang:
Fuzzy basis functions, universal approximation and orth. least squares learning,
IEEE Trans. on Neural Networks 3, pp. 807-814 (1992)

[MinPap, 1969] M. Minsky, S. Papert:
Perceptrons,
MIT Press, Cambridge, MA, Introduction, pp. 1-20 und p. 73 (1969)

[Nauck, 1996] D. Nauck, F. Klawonn, R. Kruse:
Neuronale Netze und Fuzzy–Systeme,
Vieweg (1996)

[Niendieck, 1998] S. Niendieck:
Eine universelle Repräsentation von Fuzzy–Controllern durch neuronale Netze,
Diplomarbeit, Westf. Wilhelms–Universität Münster (1998)

[Nilson, 1965] N.J. Nilson:
Learning Machines – Foundations of Trainable Pattern Classifying Systems,
McGraw–Hill, New York (1995)

[Nissen, 1997] V. Nissen:
Einführung in Evolutionäre Algorithmen,
Vieweg (1997)

[PalRay, 2000] T. Pal, N. Pal, S. D. Ray:
A Self–organized Rule Generation Scheme for Fuzzy Controllers,
Proc. of the 9th IEEE Int. Conf. on Fuzzy Systems,
San Antonio, Texas, USA, pp. 13-18 (2000)

[PatMoh, 2002] A.V. Patel, B.M. Mohan:
Some numerical aspects of center of area defuzzification method,
Fuzzy sets and systems 132, pp. 401-409 (2002)

[Rojas, 1993] R. Rojas:
Theorie der neuronalen Netze,
Springer (1993)

[Rosenblatt, 1958] F. Rosenblatt:
The perceptron: a probabilistic model for information storage
and organization in the brain,
Psychological review 65, pp. 386-408 (1958)

[Rosenblatt, 1962] F. Rosenblatt:
Principles of Neurodynamics,
Spartan Books, New York (1962)

[RuHiWi, 1986] D.E. Rumelhart, G.E. Hinton, R.J. Williams:
Learning representations by back–propagating errors,
Nature 323, pp. 533-536 (1986)

[Rumbaugh, 1991] J. Rumbaugh:
Object–Orientated Modelling and Design,
Prentice Hall (1991)

[Russel, 1923] B. Russel:
Vagueness,
The Australien Journal of Psychology and Philosophy 1, pp. 84-92 (1923)

[Schäfer, 1994] S. Schäfer:
Objektorientierte Entwurfsmethoden,
Addison–Wesley (1994)

[Schöneburg, 1994] E. Schöneburg, F. Heinzmann, S. Feddersen:
Genetische Algorithmen und Evolutionsstrategien,
Addison–Wesley (1994)

[Silipo, 2000] R. Silipo:
Extracting Information from Fuzzy Models,
Proc. of the 19th Int. Conf. of the North American
Fuzzy Information Processing Society,
Atlanta, Georgia, USA, pp. 44-48 (2000)

[Der Spiegel, 2002] Der Spiegel, Ausgabe 27, 2002,
Wie funktioniert das Lernen? pp. 68-80 (2002)

[Stroustrup, 1991] B. Stroustrup:
The C++ Programming Language,
Addison–Wesley (1991)

[Sugeno, 1985] M. Sugeno:
An introductory survey of fuzzy control,
Information Sciences, Volume 36, pp. 59-83 (1985)

[Tenhagen, 2000] A. Tenhagen:
Optimierung von Fuzzy–Entscheidungssystemen mittels
konnektionistischer Methoden,
Dissertation, Westf. Wilhelms–Universität Münster (2000)

[Tilli, 1993] T. Tilli:
Fuzzy–Logik,
Franzis (1993)

[Wang, 1992] L.-X. Wang:
Fuzzy Systems are universal approximators,
Proc. of the 1st IEEE Int. Conf. on Fuzzy Systems,
San Diego CA, USA, pp. 1163-1170 (1992)

[Watts, 2003] M. Watts:
Fuzzy Rule Extraction from Simple Evolving Connectionist Systems,
Proc. of the Conf. on Neuro–Computing and Evolving Intelligence 2003,
Auckland, Neuseeland, pp. 45-46 (2003)

[Werbos, 1974] P.J. Werbos:
Beyond regression: new tools for prediction and analysis in the behavioral sciences,
Ph. D. thesis, Harvard University, Cambridge, MA (1974)

[WidHof, 1960] B. Widrow, M.E. Hoff:
Adaptive switching circuits,
1960 IRE WESCON Convention Record, New York, IRE, pp. 96-104 (1960)

[WuTam, 1999] A. Wu, P.K.S. Tam:
A simplified model of Fuzzy Inference System constructed by using RBF Neurons,
Proc. of the IEEE Int. Fuzzy Systems Conf., Seoul, Korea, pp. 50-54 (1999)

[XioLitz, 2002] N. Xiong, L. Litz:
Reduction of fuzzy control rules by means of premise learning –
method and case study,
Fuzzy sets and systems 132, pp. 217-231 (2002)

[YingHaj, 2002] H. Ying, A. Haj–Ali:
Structure Analysis of Mamdani Fuzzy PID Controllers with
Nonlinear Input Fuzzy Sets,
Proc. of the 2002 Annual Meeting of the North American
Fuzzy Information Processing Society,
New Orleans, LA, USA, pp. 19-21 (2002)

[Zadeh, 1965] L. Zadeh:
Fuzzy sets,
Information and Control 8, pp. 338-353 (1965)

[Zell, 1996] A. Zell:
Simulation neuronaler Netze,
Addison–Wesley (1996)

[Zinth, 1996] C. Zinth:
Modifikation des Verfahrens von Lin und Lee basierend
auf dem Quickprop–Algorithmus,
Diplomarbeit, Westf. Wilhelms-Universität Münster (Juli 1996)

Lebenslauf

Name:	Steffen Niendieck
Geburtsdatum:	04.10.1970
Geburtsort:	Rheine
Familienstand:	ledig
Eltern:	Hubert Niendieck Dagmar Niendieck, geb. Stax
Schulbildung:	August 1977 – Juli 1981: Grundschule: Marienschule Roxel August 1981 – Mai 1990: Friedensschule Münster: Bischöfliche Gesamtschule mit gymnasialer Oberstufe
Abitur:	am 21.05.1990 in Münster
Zivildienst:	05.06.1990 – 31.08.1991: Arbeiterwohlfahrt Münster
Studium:	Mathematik mit Nebenfach Informatik (Diplom) Westfälische Wilhelms–Universität Münster: Oktober 1991 – Februar 1999
Prüfungen:	Diplom in Mathematik mit Nebenfach Informatik am 20.02.1999
Tätigkeiten:	Wissenschaftliche Hilfskraft am Institut für Informatik 01.04.1999 – 30.04.1999 Wissenschaftlicher Mitarbeiter am Institut für Informatik 01.05.1999 – 31.12.2003 jeweils Westfälische Wilhelms–Universität Münster
Beginn der Dissertation:	April 1999 am Institut für Informatik der Westfälischen Wilhelms–Universität Münster bei Prof. Dr. W.-M. Lippe