ART - Adaptive Resonance Theory
Architekturen, Implementierung und Anwendung

Kai-Uwe Herrmann

SS 1992

Diese Arbeit wurde am Institut für parallele und verteilte Höchstleistungsrechner (IPVR) der Fakultät Informatik
(Universität Stuttgart) verfaßt.
Betreuer : Dr. Andreas Zell
Prüfer : Prof. Dr. Andreas Reuter

Inhaltsverzeichnis

Zusammenfassung

Thema dieser Arbeit ist die Adaptive Resonance Theory, kurz ART. Sie stellt für verschiedene Anwendungen eine Klasse von Netzwerkarchitekturen und Lernverfahren zur Verfügung. G.A. Carpenter und S. Grossberg entwickelten zunächst ART 1, ein selbstorganisierendes Verfahren zur Klassifikation binärer Eingabemuster. Da die meisten Klassifikationsaufgaben reellwertige Eingabevektoren benötigen, wurde das Modell in diese Richtung erweitert; es entstand ART 2. Ein drittes Modell – ARTMAP – erlaubt es die Adaptive Resonance Theory auch für überwachtes Lernen einzusetzen. Aufgabe dieser Arbeit ist es, für jedes der genannten Verfahren verschiedene Möglichkeiten der Realisierung gegeneinander abzuwägen und die jeweils am besten geeignete für den Stuttgarter Neuronale Netze Simulator (SNNS) zu implementieren. Schließlich sollen Stärken und Schwächen dieser Modelle untersucht werden.

Kapitel 1

Einführung in Neuronale Netze

1.1 Aufbau und Funktionalität von Neuronalen Netzen

In diesem einführenden Kapitel soll zunächst geklärt werden, was ein neuronales Netz eigentlich ist. Es ist der Versuch das Zusammenspiel von Nervenzellen (= *Neuronen*) wie wir sie bei Lebewesen finden, in sehr stark vereinfachter Form zu simulieren. Die hauptsächliche Parallelität zur Natur besteht darin, daß konnektionistische Modelle, wie natürliche neuronale Netze, aus vielen „Zellen" bestehen, die über gerichtete Verbindungen variabler Stärke miteinander verbunden sind.

Diese meist sehr zahlreichen Verbindungen sind gewichtet. Das heißt, daß eine Zelle eine Nachfolgerzelle über eine Verbindung mit einem hohen Gewicht stark, über eine mit einem geringen Gewicht schwächer erregen und über eine mit einem negativen Gewicht sogar hemmen wird. Über diese Verbindungen können die Zellen Informationen austauschen, sind also in der Lage miteinander zu kommunizieren. (Siehe Abbildung 1.1).

Der Grund, warum man versucht, die, verglichen mit einem Mikroprozessor, recht langsamen Zellen der Natur nachzuahmen, ist ihre große Zahl. Sie ermöglicht einen hohen Grad an Parallelverarbeitung und macht damit die vergleichsweise geringe Geschwindigkeit wett. Aber das ist natürlich nicht alles. Primäres Ziel ist es, daß ein Netz selbständig lernt, eine bestimmte Aufgabe zu erfüllen.

Man gibt dem Netz dazu ein Eingabemuster, das die Repräsentation eines Problems darstellt und läßt es diese Information verarbeiten. Das heißt, es verteilt die Eingabe über die Verbindungen im Netz, indem jede Zelle über ihre ausgehenden Verbindungen die Information *gewichtet* an ihre Nachfolger weitergibt und damit deren Aktivierung zu einem Teil beeinflußt. Zu einem Teil deshalb, weil jede Zelle natürlich mehrere Vorgängerzellen haben kann. Auf diesem Weg wird die Information bis zur Ausgabeschicht des Netzes *propagiert* (= bekanntgemacht, verteilt).

Hier teilt man ihm mit, ob oder wie weit die erzeugte Ausgabe, also die Reaktion des Netzes, von der erwarteten oder gewünschten Reaktion abweicht. Aufgrund dieses Fehlers werden dann die Gewichte der Verbindungen im Netzwerk nach bestimmten Verfahren angepaßt. Diese Verfahren nennt man *überwachte Lernalgorithmen*. Legt man das Muster nun erneut an, so sollte der sich ergebende Fehler an der Ausgabeseite des Netzes geringer sein als beim ersten Mal.

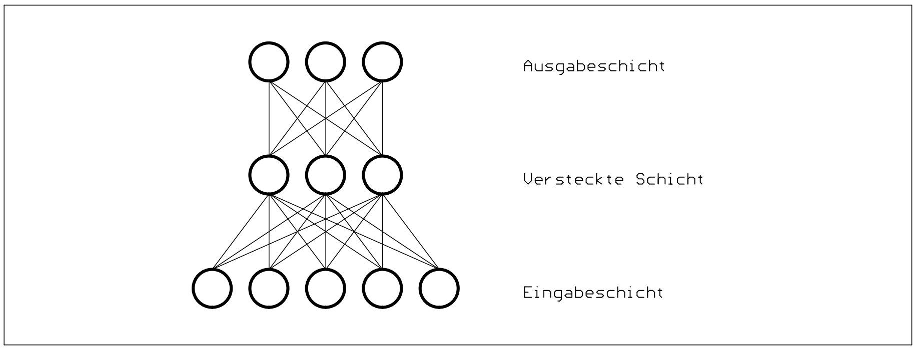


Abbildung 1.1: *Ein einfaches neuronales Netz, bestehend aus fünf Eingabe-, drei versteckten und drei Ausgabeneuronen. Die drei Schichten sind vollständig vernetzt, das heißt jedes Neuron der Vorgängerschicht hat eine Verbindung zu jedem Neuron der Nachfolgeschicht.*

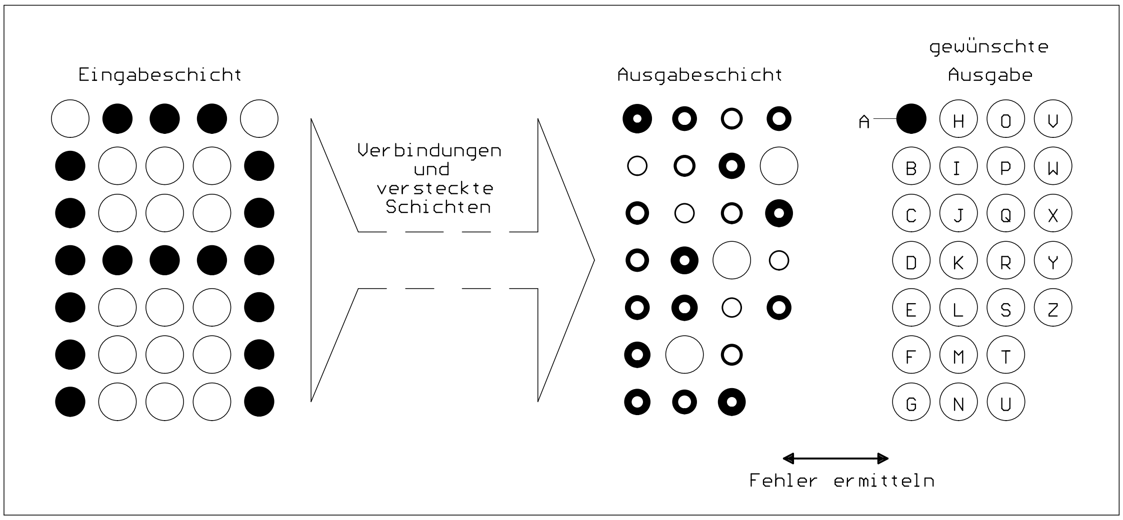


Abbildung 1.2: *Schematische Darstellung eines neuronalen Netzes zur Erkennung von Buchstaben. Je größer der Ausfüllungsgrad eines Kreises, desto stärker die Aktivierung. Ganz ausgefüllte Kreise stellen Units mit einer Aktivierung von 1.0 dar. Jedes Neuron der Ausgabeschicht repräsentiert einen Buchstaben des Alphabets. Dargestellt ist das Eingabemuster, das tatsächliche Ausgabemuster und das gewünschte Aktivierungsmuster für die angelegte Eingabe (Pixelmatrix für den Buchstaben A).*

Ein Beispiel

Um dies etwas zu veranschaulichen stelle man sich eine Pixelmatrix vor, die für den Menschen erkennbar, Buchstaben repräsentiert. Unser Ziel sei es, dem Netz beizubringen, die 26 Buchstaben des Alphabets korrekt zu klassifizieren.

Auf der Eingabeseite des Netzes wird direkt die Pixelmatrix angelegt. An der Ausgabeseite befinden sich 26 Zellen, von der jede einen Buchstaben repräsentiert, die erste das A, die zweite das B usw. Siehe dazu Abbildung 1.2.

Man legt nun nacheinander die Eingabemuster der verschiedenen Buchstaben an und paßt nach jeder Präsentation die Gewichte der Verbindungen an. Dieser Trainingszyklus wird solange wiederholt, bis das Netz in der Lage ist, die Buchstaben zu unterscheiden.

Geht man einen Schritt weiter, so liegt die Idee nahe, daß verschiedene Schriften erkannt werden sollen und zwar vom gleichen Netz. Dazu trainiert man es mit verschiedenen Zeichensätzen. Dadurch wird erreicht, daß das Netz eine Art Durchschnitt aus allen präsentierten A's bzw. B's usw. bildet und so nicht nur den einen, sondern mehrere Schriftsätze erkennen kann.

Mehr noch: werden an ein solches trainiertes Netz in *Testläufen* Zeichen eines untrainierten Zeichensatzes angelegt, dann erkennt es auch diese mit einer gewissen Zuverlässigkeit. Das heißt, es wird unempfindlich gegenüber Verzerrungen, leichten Verdrehungen, Rauschen etc. Um eine solche Unempfindlichkeit zu erreichen muß jedoch die Auflösung wesentlich höher gewählt werden, als in Abbildung 1.2 dargestellt.

Leider hängt der Lernerfolg und das Verhalten eines Netzes von vielen Parametern, wie Netzwerkarchitektur, Initialisierung der Gewichte, gewähltem Lernalgorithmus, sowie dessen Parametern ab. Ein erfolgreich trainiertes Netz ist also nicht selten ein Produkt vieler *trial-and-error*-Zyklen, die der Konstrukteur in oft zeitraubender Arbeit durchgeführt hat.

Neuronale Netze dienen in der oben beschriebenen Form vor allem als fehlertolerante, generalisierende Mustererkenner. Die genannten Lernverfahren bezeichnet man, wie erwähnt, als *überwachte Lernalgorithmen* oder *supervised learning*. Bekannt sind hier vor allem *Backpropagation, Quickprop* und *Backpercolation*.

Diese stehen im Gegensatz zu den nichtüberwachten Verfahren (*unsupervised learning*), die ohne eine Lerneingabe an der Ausgabeschicht arbeiten. Sie klassifizieren die Eingabemuster aufgrund ihrer Ähnlichkeit. Die *Kohonen Feature Map* und die Basismodelle der *Adaptive Resonance Theory*, ART 1 und ART 2 gehören zu den nichtüberwachten Lernverfahren.

Für eine ausführliche Einführung in die Thematik der neuronalen Netze sei der Leser vor allem auf [Rumelhart86], [McClelland88] und [Kemke88] verwiesen.

1.2 Das Plastizitäts-Stabilitäts-Dilemma

Bei den meisten Lernalgorithmen tritt das Problem auf, daß ein Netz beim Lernen neuer Muster die alten, bereits gelernten wieder vergißt. Ein Ausweg aus dieser Situation ist es, dem Netz alte und neue Muster gleichermaßen zu präsentieren. Nur dann ist bei diesen Verfahren die oben erwähnte Durchschnittsbildung gewährleistet.

Dieses Problem wird *Plastizitäts-Stabilitäts-Dilemma* genannt. Dabei versteht man unter *Plastizität*, daß ein Netz in der Lage sein sollte, neues Wissen zu erwerben, auch wenn bereits eine bestimmte Menge an Trainingsmustern gelernt wurde. *Stabilität* bedeutet, daß das bereits erworbene Wissen stabil ist, also nicht vergessen wird.

Die dieser Diplomarbeit zugrunde liegende *Adaptive Resonance Theory*, im folgenden durch ART abgekürzt, zeigt einen Ausweg aus diesem Dilemma auf. Kapitel 2 wird näher auf diesen Lösungsansatz eingehen.

1.3 Simulation neuronaler Netze mit SNNS

An der Fakultät Informatik der Universität Stuttgart wurde 1990 ein Projekt begonnen, dessen Aufgabe es war, einen Simulator für konnektionistische Modelle zu implementieren. Der *Stuttgarter Neuronale Netze Simulator* (SNNS) liegt inzwischen in der Version 2.1 vor und wurde bis heute vor allem im Rahmen studentischer Arbeiten implementiert und weiterentwickelt. Er ist lauffähig auf Unix-Maschinen unter X-Windows. Für ihn wurden inzwischen zahlreiche Lernverfahren implementiert.

1.4 Aufgabenstellung

Aufgabe dieser Diplomarbeit ist die Implementierung von drei verschiedenen ART-Modellen (ART 1, ART 2 und ARTMAP) in dem oben beschriebenen Simulator für neuronale Netze, SNNS. Die Implementierung erfolgt in der Programmiersprache C. Die ART-Lerverfahren sollen am Ende der Arbeit integraler Bestandteil des SNNS-Simulators sein und über die graphische Benutzeroberfläche bedient werden können.

Weiter sollen die ART-Architekturen in einer einheitlichen Notation beschrieben und verschiedene Methoden der Realisierung gegeneinander abgewogen werden. Die Implementierung, die notwendigen Schritte der Entwicklung sowie die Vor- und Nachteile der behandelten Verfahren sollen ebenfalls dokumentiert werden.

Kapitel 2

Einführung in ART

Die *Adaptive Resonance Theory*, die von Gail A. Carpenter und Stephen Grossberg entwickelt wurde, beschreibt nicht nur Trainingsverfahren, mit denen neuronale Netze selbstorganisierend lernen. Diese Verfahren verlangen auch spezielle Architekturen oder auch *Topologien* der Netze, für die sie jeweils angewandt werden können.

Dies unterscheidet ART grundsätzlich von den meisten Algorithmen für überwachtes Lernen. Diese Verfahren schränken die Topologie der Netze, die mit ihnen trainiert werden sollen nur in relativ geringem Maße ein, während für jedes der ART-Modelle die Struktur der Netzwerke, die damit bearbeitet werden können, genau definiert ist. Auf die Topologien werden wir in dem jeweils für das behandelte Verfahren vorgesehenen Kapitel genauer eingehen.

2.1 Eigenschaften von ART

An dieser Stelle soll nur allgemein untersucht werden, was die ART-Architekturen auszeichnet. Dazu soll hier zunächst noch einmal das bereits im vorangegangenen Kapitel erwähnte Plastizitäts-Stabilitäts-Dilemma betrachtet werden.

Zur Erinnerung noch einmal die Frage: wie kann ein lernendes System so entworfen werden, daß es adaptiv auf relevante Ereignisse reagieren kann (Plastizität), ohne dabei gelerntes Wissen wieder zu vergessen (Stabilität). Wie kann ein solches System stabil sein, ohne gleichzeitig einschränkend zu werden und wie kann es plastisch sein, ohne Chaos zu verusachen?

Betrachtet man das Lernen bei Lebewesen, so stellt man fest, daß diese neue Dinge lerenen, ohne dabei notwendigerweise alte zu vergessen. Der Mensch reagiert adaptiv auf neue, für ihn wichtige Ereignisse und stabil in Bezug auf alte, bereits gelernte. Dabei werden natürlich alte Regeln angepaßt, auch das ist Teil des Lernens neuer Ereignisse. Stabil bedeutet also nicht, daß eine Reaktion nach 10 Jahren genauso zu erfolgen hat, wie heute, vielmehr soll stabil bedeuten, daß eine neue Regel eine alte nicht *überschreibt* sondern *verfeinert.*

Die ART-Modelle sind, wie in den folgenden Kapiteln noch zu sehen sein wird, auf eine Weise konzipiert, die ein solches Verhalten für diese Netze sicherstellt.

Betrachten wir aber noch einen anderen Aspekt. Jeder Mensch reagiert auf äußere Einflüsse. Diese Reaktion wird aber nicht nur durch diese Stimulationen von außen beeinflußt, sondern wir begegnen ihnen mit gewissen Erwartungshaltungen.

Ein Beispiel: lesen wir einen handgeschriebenen Text, so kommt es nicht selten vor, daß ein oder mehrere Buchstaben eines Wortes so schlecht geschrieben sind, daß die Buchstaben für sich allein genommen nicht zu entziffern sind. Der Kontext aber erzeugt eine Erwartungshaltung bei uns, um welches Wort es sich dabei handeln könnte und das sogar auf mehreren Ebenen.

Zum einen wissen wir oft, welcher *syntaktischen* Kategorie das gesuchte Wort angehören muß, um unserem Gespür für die Grammatik unserer Sprache gerecht zu werden. So können wir zum Beispiel mit Sicherheit sagen, daß in dem Satz: „Hans ... Peter im Garten", an der durch Punkte markierten Stelle nur ein transitives Verb eingesetzt werden kann, wie z.B. „sieht".

Zum zweiten können wir aus dem umliegenden Text den *Sinn* des momentan gelesenen Satzes erraten und auf die Bedeutung des Wortes schließen, die es haben muß um in den Kontext zu passen.

Drittens kennen wir die *morphologischen Regeln* und einen großen Teil des Vokabulars unserer Sprache, was, wenn wir umliegende Buchstaben erkannt haben, die Menge der in Frage kommenden Wörter ebenfalls eingrenzt. So kann der fehlende Buchstabe in „_och" keinesfalls ein „u" sein, sehr wohl aber ein „k", ein „r" oder ein „n".

Die ART-Modelle versuchen dieser Tatsache gerecht zu werden. Sie reagieren ebenfalls auf einen Reiz von außen, indem sie eine Erwartung von *oben* nach *unten* publizieren. Die Erfüllung dieser Erwartung wird gemessen. Ist sie hinreichend bestätigt, so steht die Reaktion des Netzwerkes fest, ist dies dagegen nicht der Fall, so wird die Erwartung verworfen und eine neue ausprobiert. Dieser Zyklus wird dann solange wiederholt, bis entweder eine der produzierten Erwartungen bestätigt wird oder aber alle Versuche gescheitert sind, das Netz schließlich aufgibt und meldet, daß ein angelegtes Muster nicht klassifizierbar ist.

2.2 ART Basisarchitektur: Competitive Learning

Um ART noch etwas besser kennen zu lernen, sollte ihre Basisarchitektur zumindest nicht unerwähnt bleiben. Begründet sind sämtliche ART-Modelle auf einem einfacheren selbstklassifizierenden Verfahren, dem *Competitive Learning*. Ein Competitive-Learning-Netzwerk besteht aus zwei Zellschichten, der F_1- und der F_2-Schicht, die vollständig miteinander vernetzt sind (siehe Abbildung 2.1).

Die Units innerhalb des F_2-Layers sind alle untereinander mit negativen Gewichten verbunden. Diese Struktur bewirkt, daß die F_2-Unit, die von der F_1-Schicht am stärksten aktiviert wird, nach mehreren Propagierungszyklen die maximale Aktivierung annimmt. Alle anderen F_2-Units dagegen werden durch sie immer weiter „unterdrückt", also auf die Aktivierung 0 gebracht.

Steht die *Gewinnerzelle* v_j in dieser *winner-takes-all-Schicht* fest, dann werden die Gewichte zwischen den Eingabezellen und v_j so angepaßt, daß v_j beim wiederholten Anlegen des gleichen Eingabemusters noch stärker aktiviert wird. Man sagt, die Gewinnerzelle v_j klassifiziert oder kodiert das entsprechende Eingabemuster I.

Es ist mathematisch nachweisbar, daß sich das Lernverhalten unter gewissen Bedingungen stabilisieren kann. Das Netz aktiviert in diesem Fall nach mehreren Lernzyklen direkt die richtige Klassifizierungszelle. Eine der Bedingungen für eine Stabilisierung ist die, daß nicht zu viele verschiedene Eingabevektoren im Verhältnis zur Zahl der F_2-Units trainiert werden.

Gegenbeispiele zeigen aber, daß nicht notwendigerweise ein solch stabiler Zustand erreicht wird.

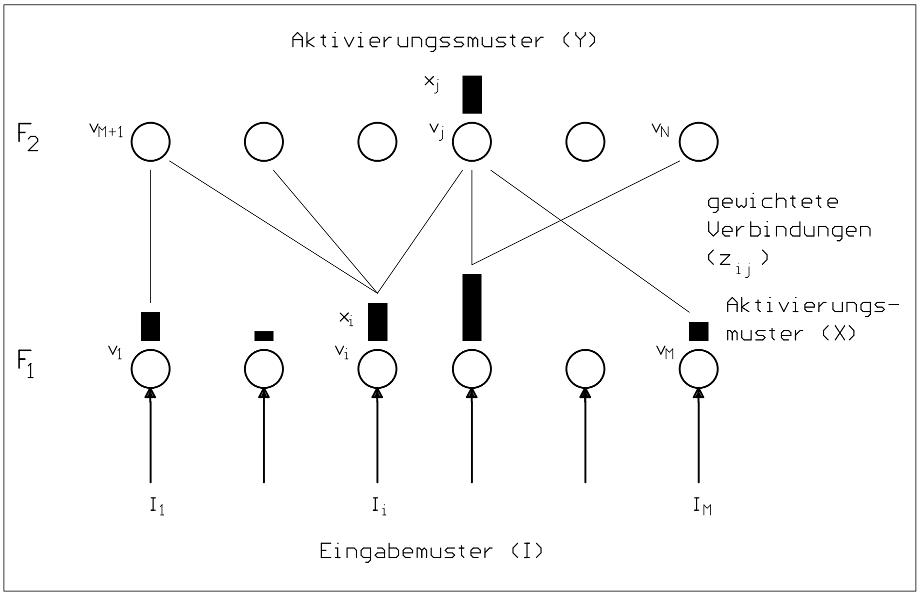


Abbildung 2.1: *Schematische Darstellung eines* Competitive Learning *Netzwerkes. Das Eingabemuster I generiert ein Aktivierungsmuster X im F_1-Layer. Hinreichend aktivierte F_1-Zellen geben ein Signal von unten nach oben an die F_2-Schicht weiter. Das Signal x_i der Unit v_i zur Unit v_j wird mit dem Gewicht z_{ij} multipliziert und alle eingehenden Signale aufsummiert, bevor diese Summe jeweils über die neue Aktivierung des Zielneurons entscheidet und so das Ausgangsaktivierungsmuster Y festlegt. Es gewinnt nur eine Zelle in der F_2-Schicht. Alle anderen haben die Aktivierung 0. Die unterschiedlichen Stärken der Linien stellen die unterschiedlich starken Gewichte der Verbindungen dar, die Höhe der Rechtecke die jeweilige Aktivierung der Unit (aus [Carpenter88]).*

So kann es vorkommen, daß in jedem Lernzyklus das gleiche Eingabemuster wieder einer neuen Kategorie zugeordnet wird und niemals in zwei aufeinanderfolgenden Präsentationen die gleiche Ausgabe in der F_2-Schicht erzeugt wird.

Dieses instabile Verhalten ist auf das Lernen anderer Muster zurückzuführen. Das Training der anderen Eingabevektoren ändert die Gewichte unter Umständen derart, daß das zuerst gelernte Muster im nächsten Zyklus einer völlig anderen Kategorie zugeordnet wird. Mit anderen Worten kann die Plastizität des Netzwerks, also die Fähigkeit neues Wissen zu erwerben, dazu führen, daß bisher Gelerntes vollständig vergessen wird.

2.3 ART Grundstruktur

An diesem Punkt setzt ART an. Aufbauend auf dem Competitive Learning werden in diesen Netzwerken Verbindungen und Schichten hinzugefügt, die es letzten Endes ermöglichen, ein stabiles Lernverhalten bei gleichzeitiger Plastizität des Netzwerkes zu gewährleisten.

In Abbildung 2.2 ist ein ART-Netz schematisch dargestellt. Es besteht aus der F_0- oder Eingabeschicht, an die jeweils ein Eingabemuster angelegt werden kann, der F_1- oder Vergleichsschicht und abschließend der F_2-Schicht, die wie beim Competitive Learning die Klassifiezierungszellen enthält. Daneben gibt es die Aufmerksamkeitskontrolle, bestehend aus $Gain_1$ und $Gain_2$ und die Orientierungskontrolle.

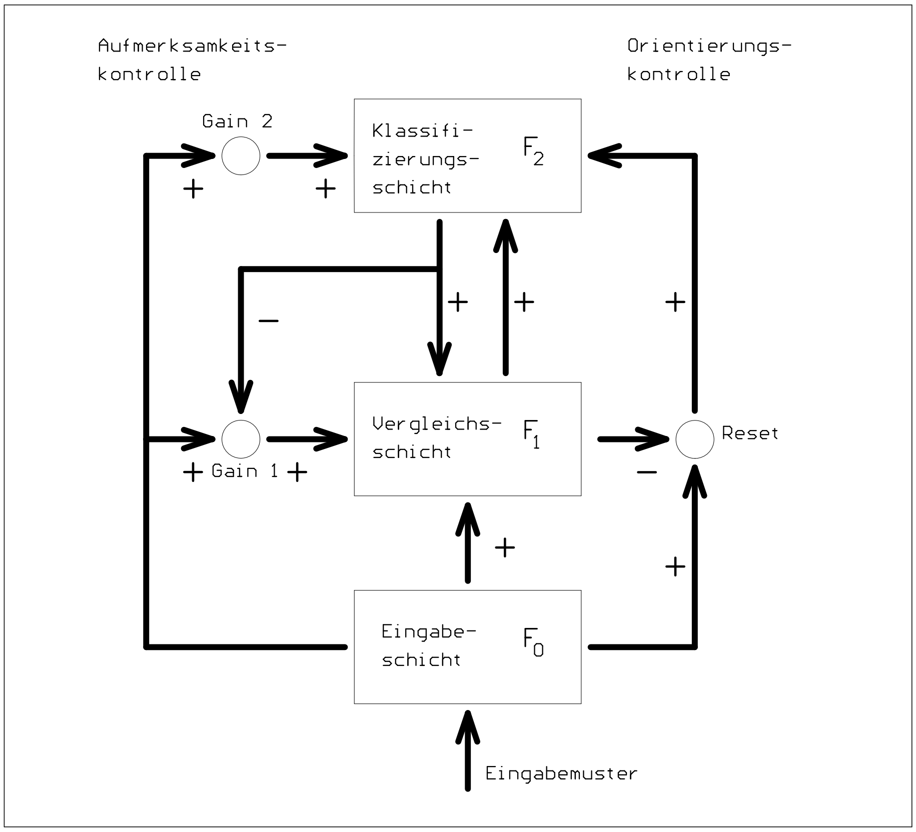


Abbildung 2.2: *Schematische Darstellung eines ART-Netzwerkes.*

In die F_0-Schicht wird zunächst das zu klassifizierende Eingabemuster kopiert. Die $Gain_1$-Unit sorgt dann dafür, daß das gleiche Muster zunächst auch unverändert in die Vergleichsschicht gelangt. Die $Gain_1$-Unit ist genau dann 1, wenn in F_0 ein Muster anliegt und in F_2 keine Units aktiviert sind. Die F_1-Units nehmen eine Aktivierung von 1 an, wenn mindestens zwei ihrer drei Eingänge ($Gain_1$, F_0, F_2) gleich 1 sind. Da die $Gain_1$-Unit momentan aktiv ist und von F_2 her keine Signale anliegen, wird das Muster aus F_0 übernommen.

Die F_1-Schicht ist vollständig von unten nach oben mit der Klassifizierungsschicht vernetzt. Über diese Verbindungen wird das Muster in F_1 dann weiter in den F_2-Layer propagiert. Dort erzeugt es ein reellwertiges Aktivierungsmuster. Diese Schicht von Klassifizierungszellen bildet wie beim Competitive Learning ein winner-takes-all-Netz, so daß genau eine F_2-Unit den Wert 1, alle anderen den Wert 0 annehmen.

Damit steht vorläufig die Klasse fest, von der das Netz „vermutet", daß der angelegte Eingabevektor in sie einsortiert werden sollte, bis jetzt also das gleiche Verfahren, wie Competitive Learning. Vorausgesetzt, es befinden sich M Units in der F_2-Schicht, so ist jetzt genau eine davon aktiv und $(M-1)$ sind inaktiv. Dieses Aktivierungsmuster wird über die Verbindungen, die von oben nach unten wieder zum F_1-Layer führen, zurückpropagiert.

Inzwischen hat die $Gain_1$-Unit die Aktivierung 0 angenommen, da ja jetzt in F_2 eine Unit aktiv ist. Damit werden in der Vergleichsschicht diejenigen Units aktiv, die sowohl von der ihr jeweils entsprechenden F_0-Unit, als auch aus der F_2-Schicht ein 1-Signal erhalten.

Die Gewichte der Verbindungen von oben nach unten sind zu Beginn alle mit 1 initialisiert. Das heißt, daß eine noch nicht verwendete Klassifizierungszelle in der Vergleichsschicht also wieder genau das Eingabemuster erzeugen wird.

Hat eine F_2-Unit dagegen bereits ein Muster klassifiziert, so sind einige der Gewichte die von ihr zu den Comparison-Units führen gleich 0 und erzeugen folglich bei diesen Zellen der F_1-Schicht keine Aktivierung, selbst wenn sie von den Eingabezellen her 1-Signale bekommen. Die aktivierten Comparison-Units bilden also immer eine Untermenge der aktivierten Eingabezellen.

Jetzt muß das Netz noch entscheiden, ob das Eingabemuster tatsächlich in die Klasse gehört, die es bis dahin angenommen hat. Dazu werden die Vektoren in F_0 und F_1 auf ihre Ähnlichkeit hin untersucht.

Stellt sich heraus, daß die Übereinstimmung hinreichend groß ist, so werden zum einen die Gewichte der Verbindungen, die von der F_1- zur F_2-Schicht führen so angepaßt, daß das gleiche Eingabemuster bei der nächsten Präsentation sofort das gewählte Klassifizierungsneuron sehr stark aktiviert. Zum anderen werden all die Gewichte der Verbindungen von der F_2-Gewinnerzelle, die zu aktiven F_1-Units führen auf 1, diejenigen, die zu inaktiven F_1-Units führen auf 0 gesetzt. Das heißt, es wird eine Kopie des durch die Gewinnerzelle erzeugten Vergleichsmusters in den Gewichten der Verbindungen von der F_2- zur F_1-Schicht abgelegt.

Für den Fall, daß die beiden Vektoren in F_0 und F_1 jedoch nicht ähnlich genug waren, wird nun ein Reset-Signal ausgelöst, das die Gewinnerzelle blockiert. Dann beginnt der ganze Vorgang von vorn, das heißt, das Eingabemuster wird wieder in F_1 kopiert ($Gain_1$ gleich 1). Dieses Muster wird erneut nach F_2 propagiert und wiederum ein Gewinner ermittelt. Bei diesem handelt es sich nun mit Sicherheit nicht um die Unit, die vorhin Gewinner war, denn die wird durch den Reset blockiert, bis ein neues Eingabemuster angelegt wird. Wir erhalten also eine *neue* Gewinnerzelle, die nun wiederum ein Muster in F_1 erzeugt, das wieder mit dem Eingabevektor verglichen wird.

Dieser Vorgang wiederholt sich solange, bis eine adäquate Klassifizierungszelle gefunden wurde, die einen F_1-Vektor erzeugt, der dem Eingabemuster ähnlich genug ist oder bis alle Klassen ausprobiert wurden, keine passende gefunden wurde und auch keine neue mehr zur Verfügung steht.

Warum aber ist dieses Verfahren im Gegensatz zu Competitive Learning stabil? Der Grund ist leicht nachzuvollziehen.

Nehmen wir an, ein Eingabemuster wird in eine neue, bislang ungebrauchte Klasse einsortiert: Wir hatten festgestellt, daß eine neue Klasse bei der Rückwärtspropagierung von F_2 nach F_1 eine genaue Kopie des Eingabemusters in F_1 erzeugt, da die Gewichte der Verbindungen von oben nach unten, die von dieser Klassifizierungszelle ausgehen, alle noch 1 sind, denn so wurden sie ja initialisiert.

Beim Lernen wird dann, wie erwähnt, in diesen Gewichten eine Kopie des F_1-Musters abgelegt. In diesem Fall also eine Kopie des Eingabemusters.

Nehmen wir weiter an, das nächste Eingabemuster bringt die gleiche F_2-Unit als Gewinner hervor. Dann werden in der F_1-Schicht die Zellen an denjenigen Stellen aktiviert, in denen sowohl das erste, als auch das zweite Eingabemuster eine 1 besitzen. Von oben nach unten liegt höchstens an all denjenigen F_1-Units eine 1 an, die im ersten Eingabemuster aktiviert waren, von F_0 nach F_1 genau an denjenigen Stellen, an denen das neue eine 1 besitzt.

Somit ist das sich in der Vergleichsschicht ergebende Muster nicht nur eine Untermenge des aktuellen, sondern auch eine Untermenge des ersten Eingabemusters.

Entscheidet die Orientierungskontrolle, daß der neue Eingabevektor tatsächlich in dieser Klasse einzuordnen ist, so werden höchstens einige der Gewichte, die ursprünglich 1 waren, 0, niemals aber eines, das bereits 0 gewesen ist, plötzlich 1.

Damit kann es zwar vorkommen, daß das erste Eingabemuster bei der nächsten Präsentation noch einer neuen Klasse zugeordnet wird. Es ist jedoch mathematisch nachweisbar, daß das System nach wenigen Zyklen stabil ist. Das heißt, zugeordnete Muster werden nicht mehr in neue Klassen einsortiert, sondern bleiben fest in der ihnen bereits zugewiesenen.

2.4 Verschiedene ART-Modelle

Offen geblieben ist bis jetzt die Frage, wozu verschiedene ART-Modelle notwendig sind.

Der Grund ist einfach. Das soeben beschriebene Verfahren geht nämlich von der Annahme aus, daß es sich bei den Eingabemustern um binäre Vektoren handelt. Das für sie entwickelte Modell heißt ART 1 und war, wie die Nummer leicht erkennen läßt, das erste, das Carpenter und Grossberg entwickelten. Wir werden in Kapitel 3 das Modell sowohl von der mathematischen als auch von der praktischen Seite beleuchten. Literatur zu ART 1 findet man vor allem in [Carpenter89], [Carpenter87a], [Carpenter88] und [Carpenter91b].

Um auch für reellwertige Eingabevektoren gerüstet zu sein wurde ART 2 entwickelt. In Kapitel 4 werden wir dieses Modell untersuchen und die zugehörige Implementierung betrachten. (Literatur: [Carpenter87b]).

ART 3, das in dieser Arbeit nicht behandelt wird, simuliert die in Kapitel 1 kurz erwähnten Neurotransmitter in natürlichen neuronalen Netzen und führt damit zu einer weiteren Verfeinerung des Modells. (Literatur: [Carpenter91a]).

Der Wunsch nach einem überwachten Lernverfahren brachte schließlich die Idee von ARTMAP hervor. In Kapitel 5 wird gezeigt, wie die Vorzüge von ART mit diesem Wunsch in Einklang gebracht werden kann. (Literatur: [Carpenter91b], [Carpenter91c]).

Ein letztes, recht ungewöhnliches Modell, das Carpenter, Grossberg und Rosen in [Carpenter91d] veröffentlichten, heißt Fuzzy ART. Es verbindet Ansätze aus der Fuzzy-Theory mit ART. Es sollte hier nicht unerwähnt bleiben, wird im Rahmen dieser Arbeit jedoch nicht näher betrachtet.

Kapitel 3

ART 1

Adaptive Resonance Theory für binäre Eingabemuster

In diesem Kapitel soll neben der Theorie speziell für ART1-Netzwerke, in je einem Abschnitt auf die Besonderheiten der Realisierung und Implementierung des Modells für SNNS eingegangen werden. Ein weiterer Abschnitt wird sich mit einem Beispiel für die Anwendung beschäftigen und exemplarisch die Möglichkeiten aufzeigen, die ART 1 bietet.

3.1 ART 1 Theorie

In Kapitel 2 wurde bereits informell der Klassifikationsablauf in einem ART-Netzwerk betrachtet. Hier sollen nun die mathematischen Bedingungen und die speziell für ART1-Netzwerke geletenden Regeln untersucht werden[1].

Aktivierung im F_1-Layer

In Abbildung 3.1 sind die wichtigsten Komponenten eines ART1-Netzwerkes dargestellt. Der Eingabevektor **I** mit den Komponenten $(I_1, I_2, \ldots, I_N)$ wird in F_0 gespeichert.

Jede F_1-Unit hat drei verschiedene Eingangssignale: erstens, die Eingabe von der ihr entsprechenden F_0-Unit, zweitens von der Gain-Kontrolle und drittens natürlich von den F_2-Units. Dabei sei schon an dieser Stelle bemerkt, daß jede F_1-Zelle von jeder F_2-Unit ein Eingangssignal bekommt. Weil aber immer nur eine einzige F_2-Unit aktiv ist, beschränkt sich die Zahl der Eingangssignale von F_2 immer auf genau eines.

Somit können an jeder F_1-Zelle zu jedem Zeitpunkt bis zu drei 1-Signale anstehen. Für diese Zellen wird nun festgelegt, daß sie genau dann aktiv werden, wenn mindestens zwei der drei Eingangssignale auf 1 stehen. Man nennt diese Aktivierungsregel die *2/3 - Regel*.

Die 2/3 Regel

Die i-te F_1-Unit ist aktiv, wenn ihre Eingangssignale einen festen Schwellwert übersteigen:

[1] Dieser Abschnitt basiert größtenteils auf den Ausführungen in [Carpenter87a] und [Carpenter91b]

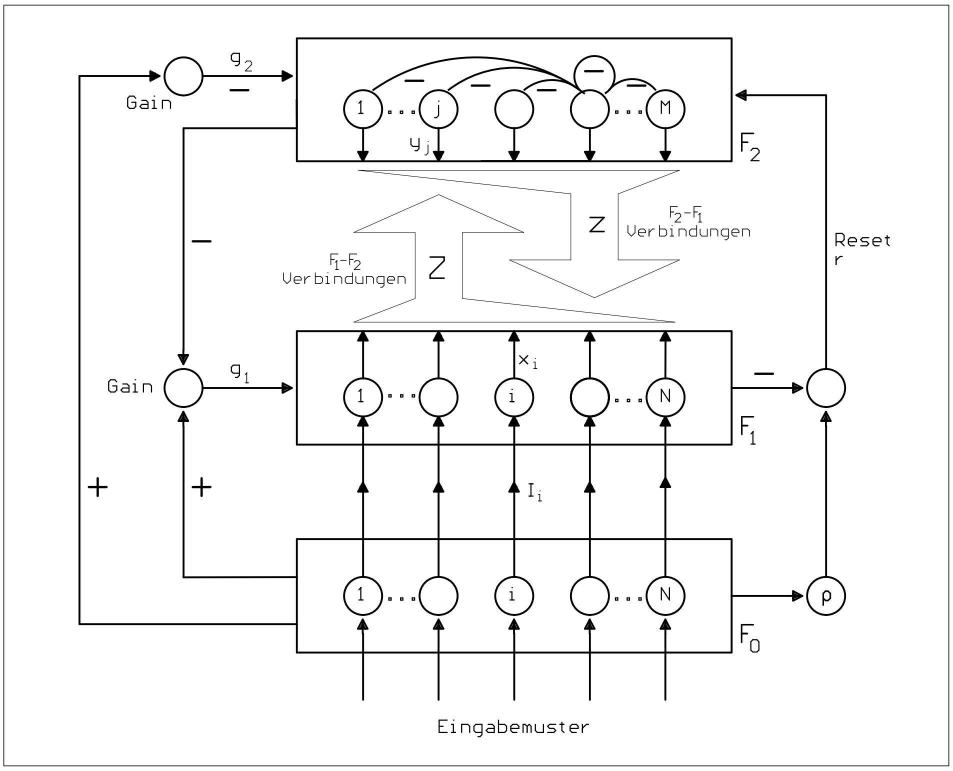


Abbildung 3.1: *Schematische Darstellung eines ART1-Netzwerks. Der binäre Eingabevektor* $\boldsymbol{I}$, *der in* F_0 *zwischengespeichert wird, beeinflußt die Aktivierung der* F_1*-Schicht von unten nach oben. Der Ausgabevektor von* F_1 *wird* $\boldsymbol{x}$ *genannt. Zwischen* F_1 *und* F_2 *befinden sich in beiden Richtungen trainierbare Gewichte, wobei diese beiden Schichten jeweils vollständig miteinander verbunden sind. Jede* F_2*-Unit hat zu jeder anderen* F_2*-Unit ein hemmendes Gewicht. Damit wird die* F_2*-Schicht zum winner-takes-all-Layer. Ein Resetsignal wird ausgelöst, wenn die Ähnlichkeit von* F_0*- und* F_1*-Vektor den Überwachungs- (vigilance) Parameter* ρ *unterschreitet.*

$$x_i = \begin{cases} 1 & \text{wenn } I_i + \mathbf{g}_1 + \sum_{j=1}^{M} y_j z_{ji} > 1 + \overline{z}, \\ 0 & \text{sonst,} \end{cases} \tag{3.1}$$

wobei I_i der $\mathrm{F}_0 \rightarrow \mathrm{F}_1$ Eingang ist, $\mathbf{g}_1$ ist die Eingabe der Gain_1-Unit und $\sum y_j z_{ji}$ ist die Summe der $\mathrm{F}_2 \rightarrow \mathrm{F}_1$ Eingangssignale y_j über die trainierbaren, gewichteten Verbindungen z_{ji}. $\overline{z}$ ist eine Konstante mit

$$0 < \overline{z} < 1. \tag{3.2}$$

Gain-Kontrolle für den $\mathbf{F}_1$-Layer

Die F_1-Gain-Kontrollzelle ist durch folgende Aktivierungsfunktion definiert:

$$\mathbf{g}_1 = \begin{cases} 1 & \text{wenn } \mathrm{F}_0 \text{ aktiv und } \mathrm{F}_2 \text{ inaktiv ist,} \\ 0 & \text{sonst.} \end{cases} \tag{3.3}$$

Alle Units der F_2-Schicht müssen also inaktiv sein, damit bei angelegtem Eingabemuster die Gain_1-Unit aktiviert werden kann. Die F_2-Units wirken hemmend (= *inhibitorisch*) auf die F_1-Gain-Kontrolle, was in Abbildung 3.1 durch das Minus an der Verbindung von F_2 zur Gain_1-Unit angedeutet ist.

Wenn also F_2 inaktiv ist, dann gilt für die Aktivierung der F_1-Units:

$$x_i = \begin{cases} 1 & \text{wenn } I_i = 1, \\ 0 & \text{sonst.} \end{cases} \tag{3.4}$$

Anderenfalls, wenn also genau eine F_2-Zelle J aktiv ist, reduziert sich die Summe $\sum y_i z_{ji}$ in Gleichung 3.1 zum Term z_{Ji}. Für diesen Fall gilt:

$$x_i = \begin{cases} 1 & \text{wenn } I_i = 1 \text{ und } z_{Ji} > \overline{z}, \\ 0 & \text{sonst.} \end{cases} \tag{3.5}$$

Aktivierung im F_2-Layer

Sei T_j die Netzeingabe des F_1-Layers in die j-te F_2-Unit, also

$$T_j = \sum_{i=1}^{N} x_i Z_{ij}, \tag{3.6}$$

wobei Z_{ij} die trainierbaren Gewichte der $F_1 \to F_2$ Verbindungen sind. Gibt es nun ein $T_j > 0$, dann ist die von F_2 gewählte Gewinnerzelle J definiert durch die Gleichung

$$T_J = \max\nolimits_{1 \leq j \leq M} \; \{T_j\}. \tag{3.7}$$

Demnach könnte es möglich sein, daß mehrere F_2-Units $J_1, J_2, \ldots, J_K$, den gleichen, maximalen Netzeingang besitzen:

$$T_{J_1} = T_{J_2} = \ldots = T_{J_K} = \max\nolimits_{1 \leq j \leq M} \; \{T_j\}.$$

Damit wäre kein eindeutiger Gewinner definiert und es wäre in diesem Fall schwierig, das weitere Geschehen zu erklären. Wir lassen diesen Fall aber außer Acht und nehmen an, stets einen eindeutigen Gewinner J zu haben. Wie dieses Problem konkret umgangen wird, soll in Abschnitt 3.2 erklärt werden.

Gain-Kontrolle für den F_2-Layer

Bei der informellen Beschreibung der Funktionsweise von ART-Netzen in Kapitel 2 wurde nicht erläutert, wozu die Gain_2-Unit benötigt wird. Für den Moment soll es genügen, wenn bekannt ist, daß sie lediglich die F_2-Schicht blockiert, wenn kein Eingabemuster in F_0 anliegt. Die Bedeutung dieser Funktion ist in ART1-Netzwerken eher sekundär. Für ARTMAP ist diese Unit jedoch notwendig, wie wir in Kapitel 5 sehen werden.

In der Theorie, wie sie Grossberg und Carpenter veröffentlicht haben, arbeitet die $\mathbf{g}_2$-Unit ähnlich wie die $\mathbf{g}_1$-Unit im F_1-Layer. So können gemäß ihres Entwurfs F_2-Zellen nur dann aktiv werden, wenn die $\mathbf{g}_2$-Unit aktiv ist. Die $\mathbf{g}_2$-Unit ihrerseits wird dann aktiv, wenn mindestens eine Unit der F_0-Schicht eine Aktivierung von 1 aufweist.

Lernregeln

Im folgenden soll jetzt geklärt werden, wie die Anpassung der trainierbaren Gewichte, sowohl zwischen F_1- und F_2-Schicht, als auch derjenigen der umgekehrten Richtung aussehen muß, um der Anforderung nach einem sich selbst stabilisierenden Netz gerecht zu werden.

Nehmen wir dazu an, es sei bereits eine Gewinnerzelle gemäß Gleichung 3.7 ermittelt worden. Weiterhin sei bereits festgestellt worden, daß der Eingabevektor tatsächlich in die gewählte Klasse gehört.

Dann können wir die Lernregeln formulieren, indem wir folgende Notation verwenden: Ist $\mathbf{a}$ ein binärer Vektor mit L Komponenten, also $\mathbf{a} = (a_1, a_2, \ldots, a_L)$, dann definieren wir die *Norm* von $\mathbf{a}$ wie folgt:

$$|\mathbf{a}| \equiv \sum_{i=1}^{L} a_i. \tag{3.8}$$

Mit anderen Worten: $|\mathbf{a}|$ ist hier die Zahl der „Einsen" im Vektor $\mathbf{a}$.

Weiter definieren wir aus dem *Durchschnitt* zweier binärer Vektoren $\mathbf{a}$ und $\mathbf{b}$ einen dritten Vektor $\mathbf{c} \equiv \mathbf{a} \cap \mathbf{b}$ mit

$$c_i = 1 \Leftrightarrow a_i = 1 \text{ und } b_i = 1 \tag{3.9}$$

und schließlich den Begriff *Untermenge* durch

$$\mathbf{a} \subseteq \mathbf{b} \Leftrightarrow \mathbf{a} \cap \mathbf{b} = \mathbf{a}. \tag{3.10}$$

Anpassung der $\mathbf{F}_2 \rightarrow \mathbf{F}_1$ - Gewichte

Das Verfahren der Anpassung dieser Gewichte wurde bereits in Kapitel 2 behandelt. Deshalb soll hier nur noch kurz auf die beschreibenden mathematischen Formeln eingegangen werden. Wie im genannten Kapitel bereits geschildert, lernen die $F_2 \rightarrow F_1$ - Gewichte, indem sie eine Kopie des Vektors $\mathbf{x}$, also des Ausgabevektors der F_1-Schicht, speichern.

Dabei gilt natürlich, daß jeweils nur die Gewichte z_{Ji}, die von der Gewinnerzelle J ausgehen, angepaßt werden. Alle anderen z_{ji} für $j \neq J$ bleiben dagegen unverändert. Formuliert als Differenzengleichung bedeutet dies:

$$\Delta z_{ji}(t) = y_j(x_i - z_{ji}(t-1)). \tag{3.11}$$

Ob sich ein Gewicht ändert, wird also zum einen dadurch bestimmt, ob die entsprechende F_2-Unit aktiv ist (Term y_j). Das stellt sicher, daß nur die Gewichte ausgehend von der Gewinnerzelle angepaßt werden. Zum anderen wird sie durch den Unterschied zwischen bereits gelerntem und aktuellem Vektor bestimmt. Da x_i niemals 1 sein kann, wenn z_{ji} gleich 0 ist, ist die Differenz $(x_i - z_{ji})$ entweder 0 oder -1. Das heißt der Wert von z_{ji} ändert sich entweder gar nicht oder aber er ändert sich von 1 auf 0.

Zu Beginn werden alle z_{ji} auf ihren maximalen Wert gesetzt:

$$z_{ji}(0) = 1 \qquad \forall\, 1 \leq j \leq M, 1 \leq i \leq N. \tag{3.12}$$

Der Vektor $\mathbf{z}_J$ ist also ein binärer Vektor.

Mit den Gleichungen 3.4, ff. kann dann die Aktivierung der F_1-Schicht wie folgt beschrieben werden:

$$\mathbf{x} = \begin{cases} \mathbf{I} & \text{wenn } F_2 \text{ inaktiv ist} \\ \mathbf{I} \cap \mathbf{z}_J & \text{wenn die } J\text{-te } F_2\text{-Unit aktiv ist} \end{cases} \tag{3.13}$$

und die Änderung des Gewichtsvektors der Gewinnerzelle J mit den Gleichungen 3.5 und 3.11 durch

$$\mathbf{z}_J^{\text{NEU}} = \mathbf{I} \cap \mathbf{z}_J^{\text{ALT}}. \tag{3.14}$$

Eine noch nicht verwendete Klasse J impliziert somit wegen $\mathbf{I} \subseteq \mathbf{1}$:

$$\mathbf{z}_J = \mathbf{I}. \tag{3.15}$$

Anpassung der $F_1 \rightarrow F_2$ - Gewichte

Zunächst ein Wort zur Initialisierung dieser Gewichte. Um dafür Sorge zu tragen, daß die F_2-Knoten in der Reihenfolge ihres Index belegt werden, ist folgende Belegung der Gewichte im noch nicht trainierten Netz üblich:

$$Z_{ij}(0) = \alpha_j \tag{3.16}$$

mit

$$\alpha_1 > \alpha_2 > \ldots > \alpha_M. \tag{3.17}$$

Ähnlich wie der $F_2 \rightarrow F_1$ - Vektor $\mathbf{z}_J$ wird auch $\mathbf{Z}_J$ durch den Lernvorgang proportional zum F_1-Ausgangsvektor $\mathbf{x}$ angepaßt, wenn Unit J in der F_2-Schicht aktiv ist. Dabei wird jedoch $\mathbf{Z}_J$ umgekehrt proportional zu $|\mathbf{x}|$ skaliert.

$$Z_{iJ} = \frac{x_i}{\beta + |\mathbf{x}|}, \tag{3.18}$$

mit $\beta > 0$. Als Vektor dargestellt bedeutet dies

$$\mathbf{Z}_J = \frac{\mathbf{I} \cap \mathbf{z}_J^{\mathrm{ALT}}}{\beta + |\mathbf{I} \cap \mathbf{z}_J^{\mathrm{ALT}}|}. \tag{3.19}$$

Um sicherzustellen, daß ein Eingabemuster $\mathbf{I}$, das genau einem zuvor gelernten Vektor $\mathbf{Z}_J$ entspricht, die F_2-Unit J aktiviert und nicht statt dessen eine neue Kategorie hervorbringt, muß bei der Gewichtsinitialisierung folgende Bedingung zusätzlich beachtet werden:

$$0 < \alpha_j = Z_{ij}(0) < \frac{1}{\beta + |\mathbf{I}|} \leq \frac{1}{\beta + \mathrm{N}}. \tag{3.20}$$

Dadurch werden die initialen Gewichte der $\mathrm{F}_1 \rightarrow \mathrm{F}_2$ - Verbindungen so klein, daß eine neue Klasse tatsächlich nur dann gewinnen kann, wenn alle F_2-Units, denen bereits ein Muster zugeordnet wurde, die Aktivierung 0 besitzen.

Der F_2-Netzeingabevektor $\mathbf{T} \equiv (T_1, T_2, \ldots, T_M)$, der von der F_1-Schicht generiert wird, kann dann durch

$$T_j = \sum_{i=1}^{N} x_i Z_{ij} = \begin{cases} |\mathbf{I}|\alpha_j & \text{wenn } j \text{ eine noch nicht} \\ & \text{verwendete Unit ist} \\ |\mathbf{I} \cap \mathbf{z}_j|/(\beta + |\mathbf{z}_j|) & \text{wenn } j \text{ eine verwendete} \\ & \text{Unit ist} \end{cases} \tag{3.21}$$

beschrieben werden. Dabei sollte β nur so groß gewählt werden, daß T_j – innerhalb der bereits verwendeten Klassen – durch die Größe von $|\mathbf{I} \cap \mathbf{z}_j|$ in Relation zu $|\mathbf{z}_j|$ bestimmt wird. Würde β nämlich groß gewählt, so fiele letzterer Term kaum noch ins Gewicht und T_j hinge nur noch von $|\mathbf{I} \cap \mathbf{z}_j|$ ab.

Test einer Hypothese

Mit den Gleichungen 3.7, 3.20 und 3.21 könnte nun eine F_2-Unit zur Klassifikation des Eingabemusters gewählt werden, obwohl die Übereinstimmung zwischen $\mathbf{I}$ und $\mathbf{z}_J$ nur sehr gering ist.

ART 1 ist aber in der Lage zu prüfen, ob diese Übereinstimmung für das System „gut genug" ist und, wenn nicht, gegebenenfalls zu veranlassen, daß eine andere Klasse gesucht wird, die ein besseres Ergebnis liefert.

Dazu wird, wie bereits an früherer Stelle beschrieben, die Erwartung der Gewinnerzelle J in die F_1-Schicht zurückpropagiert ($= \mathbf{I} \cap \mathbf{z}_J$). Um festzustellen, ob $\mathbf{I}$ tatsächlich in die Klasse J gehört, findet hier ein Vergleich des F_1-Musters mit dem Eingabemuster in der F_0-Schicht statt.

Die Zahl der Units in der F_1-Schicht, deren Aktivierung 1 ist, wird mit derjenigen in der F_0-Schicht verglichen. Ist der Quotient größer als ein „Vertrauensparameter" (= Vigilance-Parameter) ρ, dann wird die Hypothese bestätigt, die Gewichte also entsprechend der vorigen Ausführungen angepaßt. Ist der Quotionent dagegen $\leq \rho$, dann wird ein Reset-Signal ausgelöst. Mathematisch bedeutet das

$$\text{Reset} \Leftrightarrow \frac{|\mathbf{I} \cap \mathbf{z}_J|}{|\mathbf{I}|} \leq \rho. \tag{3.22}$$

Mit diesem Reset wird die aktuelle Gewinnerzelle unterdrückt (und zwar nicht nur kurzzeitig, sondern bis zum Anlegen eines anderen Eingabemusters). Damit ist in der F_2-Schicht jetzt keine Unit mehr aktiv und $\mathbf{g}_1$ bekommt dadurch wieder den Wert 1, das Eingabemuster $\mathbf{I}$ wird wieder in den F_1-Layer kopiert und der ganze Zyklus beginnt von vorn.

Der einzige Unterschied besteht darin, daß J jetzt nicht mehr als Gewinner aus der F_2-Schicht hervorgehen kann, da sie ja durch das Reset blockiert wird. Wir werden in Abschnitt 3.2 sehen, wie diese Blockierung im Netz realisiert werden kann.

Bevor diese etwas trockenen, theoretischen Betrachtungen abgeschlossen werden, sei der Leser noch darauf hingewiesen, daß für $\rho \leq 0$ kein Reset stattfindet und so die Orientierungskontrolle abgeschaltet werden kann. Für $\rho = 1$ wird jedes Eingabemuster in eine eigene Klasse eingeordnet, denn die aus dem F_2-Layer zurückpropagierte Erwartung muß zu 100% mit dem Eingabemuster $\mathbf{I}$ übereinstimmen. Für $\rho > 1$ wird auf jeden Fall ein Reset ausgelöst. Ein solcher Wert ist somit nicht sinnvoll.

3.2 Realisierung von ART 1

In diesem Abschnitt soll die Realisierung der eben behandelten Theorie für SNNS behandelt werden. Dabei wird neben der Modellierung, wie sie schließlich Eingang in die Implementierung gefunden hat, auch jeweils erwähnt werden, welche anderen Entwurfsmöglichkeiten bestanden hätten.

Grobentwurf

Eine grundsätzliche Entscheidung steht am Beginn des Entwurfs der ART1-Implementierung: Wird ART 1 so implementiert, daß die Steuerung des Modells zu großen Teilen *algorithmisch* erfolgt, oder gestaltet man das Netz so, daß es diese Aufgaben selbst erfüllen kann.

Diese Entscheidung wird hauptsächlich von einem Aspekt beeinflußt: der Effizienz. Eine algorithmische Steuerung würde im einzelnen bedeuten:

1. Wir merken uns in logischen Variablen, ob die Bedingungen für die Aktivierung von $\mathbf{g}_1$ gegeben sind.
2. $\mathbf{g}_1$ wird nicht tatsächlich als Unit realisiert sondern als logische Variable.
3. Wir ermitteln den Gewinner in der F_2-Schicht nicht wirklich dadurch, daß ein winner-takes-all-Netz solange „läuft", bis derselbe feststeht. Vielmehr durchwandern wir die Units des F_2-Layers linear und ermitteln so mit Zeitkomplexität $O(M)$ die Zelle mit der maximalen Aktivierung.
4. Der Vergleich zwischen dem Aktivierungsmuster in der F_1-Schicht und dem Eingabemuster in F_0 erfolgt, indem wir beide Layer durchwandern und dabei jeweils die Zahl der Units mit Aktivierung 1 ermitteln.
5. Ebenso ist die Blockierung der entsprechenden Recognition-Unit nach einem Reset durch ein Feld logischer Werte realisiert. Bei der Ermittlung des Gewinners wird dann jeweils in diesem Feld überprüft, ob die gerade betrachtete Unit noch als Gewinnerzelle in Frage kommt.

Würden obige Punkte jedoch tatsächlich in dieser Form implementiert, so bliebe vom Charakter eines neuronalen Netzes nicht sehr viel übrig. Da nach Wunsch des Autors eben dieser erhalten bleiben soll, wird hier eine Kompromißlösung gewählt.

Dabei wird die Hauptarbeit dem Netz überlassen. Einzig die Ermittlung des Gewinners in der F_2-Schicht soll, wie eben unter Punkt 2. beschrieben, algorithmisch erfolgen. Dafür gibt es zwei Gründe: erstens ist dieser Punkt tatsächlich der zeitkritischste, denn wenn in einem winner-takes-all-Netz zufällig mehrere Neuronen fast die gleiche Aktivierung besitzen, so sind sehr viele Propagierungszyklen notwendig, bis der Gewinner feststeht.

Der zweite Grund beeinflußt diese Entscheidung jedoch noch stärker. Wie im letzten Abschnitt bereits erwähnt wurde, kann es vorkommen, daß der Gewinner nicht eindeutig zu ermitteln ist. Das ist dann der Fall, wenn mehrere Zellen in dem winner-takes-all-Netz exakt den gleichen Netzeingang haben. Dieser Fall sollte jedoch, wie wir dort gesehen haben, vermieden werden. Mit der algorithmischen Ermittlung des Gewinners kann sichergestellt werden, daß es nur einen solchen gibt. Dabei handelt es sich jeweils um die F_2-Unit mit minimalem Index bei maximaler Aktivierung. Alle F_2-Zellen mit größerem Index, die die gleiche, maximale Aktivierung besitzen, werden ignoriert.

Damit sind die grundlegenden Entwurfsentscheidungen gefällt, und es kann daran gegangen werden, etwas detaillierter zu betrachten, wie die Grobstruktur des ART1-Netzes aus Abblidlung 3.1 im einzelnen realisiert werden kann.

Der Entwurf der Schichten

In Abbildung 3.2 ist der vorläufige Entwurf eines ART1-Netzes dargestellt, dessen Schichten im folgenden erläutert werden.

Der Input-Layer

Zur F_0-Schicht gibt es nicht viel zu sagen. Sie ist eine Schicht bestehend aus N Eingabeneuronen, also solchen, die selbst keine Eingänge haben dürfen. Wir werden die i-te dieser Units in Zukunft mit $\mathbf{inp}_i$ bezeichnen.

Der Comparison-Layer

In der F_1-Schicht ist etwas mehr zu beachten. Es handelt sich dabei um einen Layer von N Zellen, die drei untserschiedliche Arten von Eingängen besitzen.

Zum einen hat die i-te dieser Comparison-Units ($\mathbf{cmp}_i$) einen Eingang von der Unit $\mathbf{inp}_i$, zweitens einen Eingang von der Gain-Unit $\mathbf{g}_1$ und schließlich von allen Zellen in der F_2-Schicht.

Die 2/3-Regel, wie sie oben diskutiert wurde, wird durch eine einfache Schwellwertfunktion realisiert. So wird eine Comparison-Unit $\mathbf{cmp}_i$ genau dann aktiv, wenn die Summe ihrer Eingänge mindestens 2.0 beträgt.

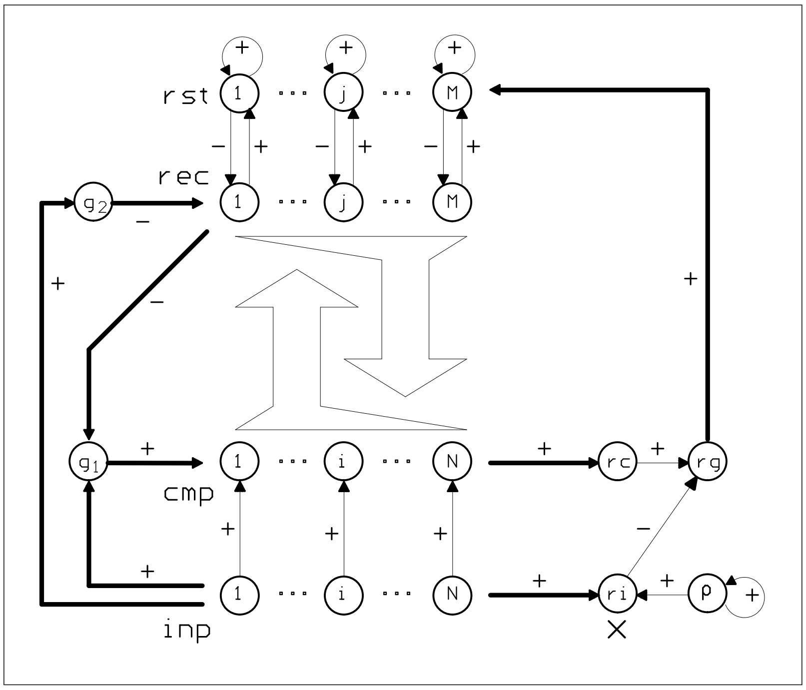


Abbildung 3.2: *Vorentwurf der ART1-Netzwerkstruktur. Dünne Pfeile stellen eine Verbindung von einer Unit zu einer anderen dar. Dicke Pfeile, die von einem Layer zu einer Unit verlaufen bedeuten, daß jede Unit des Layers eine Verbindung zum Zielneuron hat. Analog bedeutet ein dicker Pfeil von einer Unit zu einem Layer, daß die Unit mit jeder Zelle des Layers verbunden ist. Die beiden großen Pfeile in der Mitte stellen die vollständige Verbindung von Recognition- und Comparison-Layer und umgekehrt dar. Das $\times$-Zeichen unter der* **ri**-*Unit deutet an, daß sie eine* Produkt-Zelle *ist, deren Aktivierung sich aus der Multiplikation ihrer Eingnge berechnet. Der erste Eingang ist die Summe der aktiven Input-Units, der zweite ist der Wert von ρ.*

Der Recognition-Layer

Diese M Units ($\mathbf{rec}_1$, ..., $\mathbf{rec}_M$) haben jeweils Eingänge von allen Comparison-Units. Später werden noch mehr hinzukommen. Diese Verbindungen werden jedoch erst an den relevanten Stellen eingeführt und erklärt.

Dieser Entwurf entspricht dem Netz in Abbildung 3.1. Trotzdem sind einige Zusätze nicht zu vermeiden, um einen reibungslosen Betrieb zu gewährleisten.

Da ist zunächst

Der Reset-Layer

Er besteht wiederum aus M Units, die wir als lokale Reset-Units bzw. die j-te dieser Zellen mit $\mathbf{rst}_j$ bezeichnen werden. So ist jede dieser lokalen Reset-Units genau einer Recognition-Unit zugeordnet. Ihre Aufgabe ist es, bei auftretendem Reset-Signal die entsprechende Recognition-Unit zu blockieren und zwar so lange, bis ein neues Eingabemuster angelegt wird.

Anders formuliert: es gibt bei angelegtem Eingabemuster $\mathbf{I}$ genau zwei Aktivierungsbedingungen für die lokale Reset-Unit $\mathbf{rst}_j$:

1. $\mathbf{rec}_j$ ist aktiv **und** es ist ein Reset-Signal ausgelöst worden, dann soll $\mathbf{rst}_j$ aktiviert werden.
2. $\mathbf{rst}_j$ ist bereits aktiv, dann soll $\mathbf{rst}_j$ aktiv bleiben, bis ein neues Eingabemuster angelegt wird.

Konkret bedeutet das, daß jede Zelle $\mathbf{rst}_j$ für $1 \leq j \leq M$ drei Eingänge besitzt, einen von der globalen Reset-Unit **rg**, einen von der entsprechenden Recognition-Unit $\mathbf{rec}_j$ und einen von sich selbst. Letztgenannter sorgt einfach dafür, daß sich die lokale Reset-Unit in jedem Propagierungs-Zyklus wieder selbst aktiviert, wenn sie nur einmal durch Erfüllung der ersten Bedingung aktiv geworden ist.

Da es die Aufgabe einer lokalen Reset-Zelle ist, die ihr jeweils entsprechende Recognition-Unit zu blockieren, wenn sie aktiv ist, erhält jede Unit $\mathbf{rec}_j$ einen zusätzlichen Eingang mit negativem Gewicht von $\mathbf{rst}_j$. Dieser sorgt dafür, daß die Aktivierung von $\mathbf{rec}_j$ so gering wird, daß sie in keinem Fall Gewinnerzelle werden kann, wenn $\mathbf{rst}_j$ aktiv ist.

Die $\mathbf{g}_1$-Unit

Die Gain-Kontrolle wird wieder so realisiert, wie es im vorigen, theoretischen Abschnitt erläutert wurde. D. h., die $\mathbf{g}_1$-Unit hat Eingänge von jeweils allen Input- und Recognition-Units. Sie wird genau dann aktiv, wenn die beiden folgenden Bedingungen erfüllt sind:

1. Die Summe der Eingänge vom Input-Layer ist größer oder gleich 1.0, d. h. es ist mindestens eine Unit $\mathbf{inp}_i$ aktiv.
2. Die Summe der Eingänge vom Recognition-Layer ist kleiner oder gleich 0.0, d. h. keine dieser Zellen weist eine positive Aktivierung auf.

Die $\mathbf{g}_2$-Unit

Die $\mathbf{g}_2$-Kontrolle wird gegenüber dem Vorschlag von Carpenter und Grossberg gerade in umgekehrter Weise realisiert. Das heißt, sie wird nicht dann aktiv, wenn ein Eingabemuster in F_0 anliegt, sondern dann, wenn keines anliegt.

Die Recognition-Units erhalten dann auch keinen positiven Eingang, sondern einen negativen von der $\mathbf{g}_2$-Unit, der sie für den Fall, daß kein Muster in der Eingabeschicht vorhanden ist, hemmt.

Die Reset-Kontrolle

Um die Arbeit der Reset-Kontrolle zu erklären, müssen wir zunächst Ungleichung 3.22 aus Abschnitt 3.1 etwas umformen. Zur Erinnerung: ein Reset-Signal soll dann ausgelöst werden wenn die Zahl der aktivierten Comparison-Units dividiert durch die Zahl der aktiven Input-Units kleiner als der Vigilance-Parameter ρ ist:

$$\text{Reset} \Leftrightarrow \frac{|\mathbf{I} \cap \mathbf{z}_J|}{|\mathbf{I}|} < \rho.$$

Bringen wir den Nenner auf die andere Seite, so entsteht

$$\text{Reset} \Leftrightarrow |\mathbf{I} \cap \mathbf{z}_J| < \rho|\mathbf{I}|. \tag{3.23}$$

Die linke Seite der Ungleichung wird durch eine Unit **rc** realisiert. Sie erhält Eingänge von allen Comparison-Units und ihre Aktivierung berechnet sich aus der Summe all dieser Eingänge. Sie ermittelt also die Zahl der aktiven F_1-Zellen.

Die rechte Seite ist etwas komplexer: wir führen die Unit **ri** ein, die vor allem dafür zuständig ist – analog zur **rc**-Unit – die Zahl der aktiven F_0-Units zu ermitteln. Das Problem, das hier jedoch zusätzlich auftritt, ist, daß der so ermittelte Wert noch mit ρ multipliziert werden muß.

Zunächst war die Idee, den Vigilance-Parameter „algorithmisch" zu realisieren. Im Programmcode wäre eine Variable ρ und eine spezielle Aktivierungsfunktion für die **ri**-Unit verwendet worden. Diese Funktion hätte Zugriff auf die Variable gehabt und wäre so in der Lage gewesen, das gewünschte Produkt zu berechnen.

Im Hinblick auf die Implementierung von ARTMAP (siehe Kapitel 5) jedoch, wurde schließlich doch ein anderer Weg gewählt. Auch der Parameter ρ wird, aus später noch näher zu erläuternden Gründen, netzintern realisiert. Dazu benötigen wir noch eine weitere Unit ρ, deren Aufgabe es ist, einfach ihren Wert ständig an die **ri**-Unit zu liefern.

Diese multipliziert dann die Summe aller Eingänge vom Input-Layer mit dem Wert, der von der Unit ρ geliefert wird. Schließlich formen wir Ungleichung 3.23 noch einmal um, indem wir die rechte Seite durch Subtraktion auf die linke bringen.

$$|\mathbf{I} \cap \mathbf{z}_J| - \rho|\mathbf{I}| < 0. \tag{3.24}$$

Eine dritte Unit **rg** – die globale Reset-Unit – besitzt dann einen positiven Eingang von **rc** und einen negativen Eingang von **ri**. Sie summiert beide auf und Sie berechnet so genau die Differenz der linken Seite in Ungleichung 3.24. Ist sie kleiner als 0, so wird **rg** aktiv und gibt ein Reset-Signal in das Netz.

Synchronisation

Obwohl im bisherigen Entwurf alle „lebensnotwendigen" Elemente für das Netzwerk enthalten sind, ist es leider noch nicht komplett. Aus Gründen der Synchronsation müssen noch einige Units hinzugefügt werden.

Was bedeutet Synchronisation? An einigen Stellen tritt das Problem auf, daß bestimmte Ereignisse im Netz für relevante Units erst nach mehreren Takt- bzw. Propagierungszyklen bekannt werden. Häufig müssen zwei oder mehr dieser Signale aber gleichzeitig bei einer bestimmten Zelle eintreffen, um dieser eine vernünftige Entscheidung zu ermöglichen.

Hierzu ein Beispiel: Betrachten wir noch einmal Abbildung 3.2 und bezeichnen den Zeitpunkt, zu dem die Input-Units durch das Anlegen eines Eingabemusters aktiviert werden mit t_0. Den Zeitpunkt nach dem ersten Propagierungszyklus nennen wir t_1, den nach dem n-ten Zyklus, t_n.

Zum Zeitpunkt t_0 sind einige Input-Units aktiv. Zum Zeitpunkt t_1 hat sich deren Information bis $\mathbf{g}_1$ und **ri** fortgepflanzt. Somit ist $\mathbf{g}_1$ zum Zeitpunkt t_1 aktiv und **ri** bei angenommenem positivem

ρ, ebenfalls. Dagegen ist noch keine Comparison-Unit in einen positiven Aktivierungszustand versetzt worden, da $\mathbf{g}_1$ ja zu t_0 noch inaktiv war.

Im nächsten Schritt (t_2) wird nun zum einen das Eingabemuster in den Comparison-Layer kopiert, da jetzt $\mathbf{g}_1$ aktiv ist – und vorläufig auch bleibt. Weiterhin wird **rg** aktiv, denn **rc** ist noch 0 und **ri** positiv, da wir $\rho > 0$ angenommen hatten.

In t_3 werden dann erstmals die Recognition-Units aktiviert und dort ein Gewinner ermittelt, d.h. $\mathbf{rec}_j$ hat Aktivierung 1, $\mathbf{rec}_m$ für alle $1 \leq m \leq M$ und $m \neq j$, Aktivierung 0. **rg** bleibt weiterhin aktiv, da **rc** erst jetzt, zum Zeitpunkt t_3 aktiv wird, denn die Comparison-Units wurden ja erst zu t_2 aktiviert.

Somit ist jetzt der kritische Fall eingetreten, daß zum Zeitpunkt t_4 an $\mathbf{rst}_j$ sowohl ein Signal von $\mathbf{rec}_j$, als auch ein Reset-Signal ansteht, und so die Aktivierungsbedingung für $\mathbf{rst}_j$ erfüllt ist. Dieser Reset ist jedoch lediglich als initiales Fehlverhalten des Netzwerkes zu werten, da zu Beginn zwar **ri** schon nach einem Zyklus, **rc** jedoch erst im dritten Zyklus aktiv wird.

Im vierten Zyklus sind die Verhältnisse dann klar, denn jetzt gibt **rc** ein positives Signal an **rg** und die wird inaktiv. Wir müssen also nur dafür sorgen, daß die Weitergabe der Information des Recognition-Layers zum Reset-Layer um einen Propagierungszyklus verzögert wird, damit die fälschliche Aktivierung von $\mathbf{rst}_j$ verhindert wird.

Dazu wird zwischen Recognition- und Reset-Schicht eine dritte eingefügt, die Delay-Schicht. Sie ist in Abbildung 3.3 dargestellt und bietet neben der eben beschriebenen, notwendigen Verzögerung von Recognition- zu Reset-Layer einen weiteren Vorteil.

Da jetzt auch keine direkten Verbindungen mehr zwischen Recognition- und Comparison-Layer bestehen, sondern die $F_2 \rightarrow F_1$-Verbindungen vom Delay-Layer zur Comparison-Schicht geführt werden, ist damit gleichzeitig sichergestellt, daß die Informationen der Rückwärtspropagierung erst zu dem Zeitpunkt an den Comparison-Units anliegen, zu dem auch bereits die $\mathbf{g}_1$-Unit inaktiv geworden ist. Diese bekommt ihre Eingänge nämlich weiterhin direkt aus dem Recognition-Layer und hat somit genau einen Zyklus Zeit, inaktiv zu werden, bevor die Informationen der Rückwärtspropagierung bereitstehen.

Bei Betrachtung von Abbildung 3.3 fällt auf, daß noch andere Zellen hinzugekommen sind. Zunächst soll auf die **cl**-Unit und die **nc**-Unit eingegangen werden.

Dazu ist noch einmal hervorzuheben, daß das Netz so konstruiert ist, daß es gewissermaßen ein „Eigenleben" führt: einmal angestoßen, wird es, das ist mathematisch sichergestellt, irgendwann in einen stabilen Zustand gelangen. Der Lernalgorithmus muß jedoch von diesem Ereignis in Kenntnis gesetzt werden.

Bis jetzt sieht die Propagierungsvorschrift wie folgt aus:

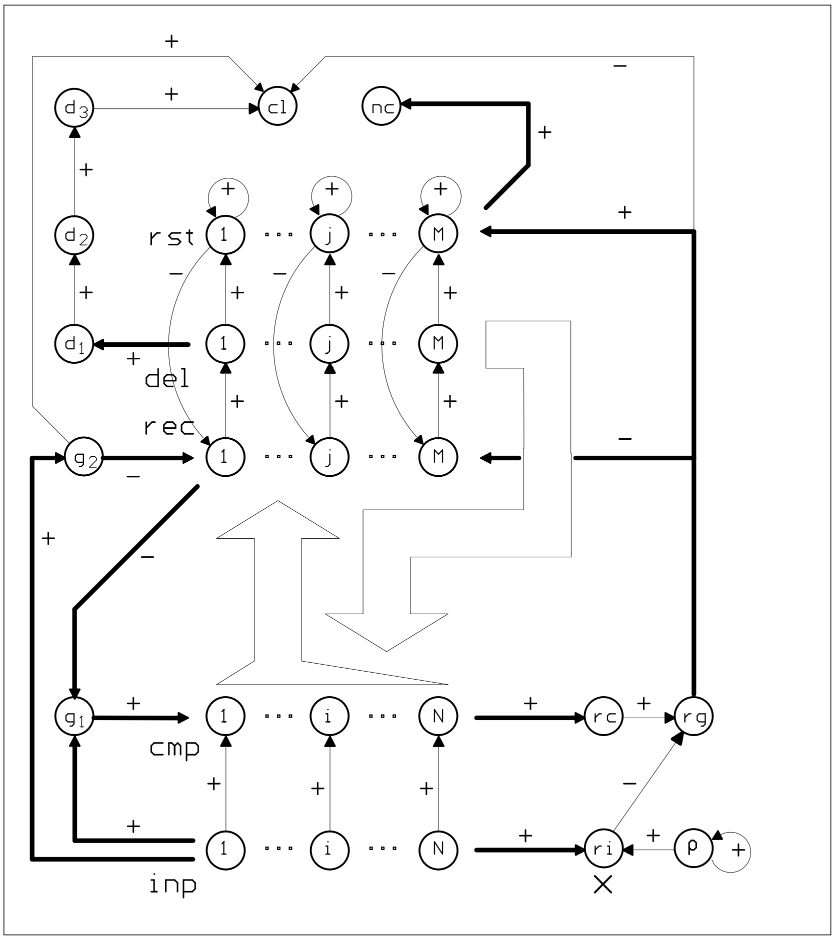


Abbildung 3.3: *Endgültiger Entwurf der ART1-Netzwerkstruktur. Dünne Pfeile stellen eine Verbindung von einer Unit zu einer anderen dar. Dicke Pfeile, die von einem Layer zu einer Unit verlaufen, bedeuten, daß jede Unit des Layers eine Verbindung zum Zielneuron hat. Analog bedeutet ein dicker Pfeil von einer Unit zu einem Layer, daß die Unit mit jeder Zelle des Layers verbunden ist. Die beiden großen Pfeile in der Mitte stellen die vollständige Verbindung von Comparison- und Recognition-Layer bzw. Delay- und Comparison-Layer dar.*

```
do {
     propagate();
     get_winner();
} while (TRUE);
```

Leider ist sie endlos. Drei Abbruchkriterien können jedoch – vorläufig verbal – formulieret werden. Zum einen braucht das Netz nicht weiter zu propagieren, wenn für ein Muster eine zugehörige Klasse gefunden wurde. Zum anderen auch dann nicht, wenn ein Muster nicht klassifizierbar ist, also in keine der etablierten Klassen hineinpaßt und auch keine neue mehr vorhanden ist. Schließlich ist die Fortsetzung der Propagierung auch dann unnötig, wenn kein Eingabemuster in der F_0-Schicht anliegt.

Das Netz soll also mitteilen, wenn eine dieser Bedingungen erfüllt ist.

Dazu werden die beiden Units **cl** (für *classified*) und **nc** (für *not classifiable*) eingeführt. Die **cl**-Unit soll dann aktiv werden, wenn eine Unit $\mathbf{del}_j$ aktiv ist und die Rückwärtspropagierung keinen Reset ausgelöst hat. Die **rg**-Unit bekommt die Information von der Rückwärtspropagierung jedoch erst drei Zyklen, nachdem $\mathbf{del}_j$ aktiv geworden ist.

Daher muß auch das Ereignis ihres Aktivwerdens drei Zyklen lang vor der **cl**-Unit „geheimhalten" werden, denn nur dann steht die Information ob ein Reset ausgelöst wurde oder nicht, gleichzeitig mit derjenigen darüber an, ob auch eine Klasse aktiv ist.

Drei Delay-Units (in Abbildung 3.3 mit $\mathbf{d}_1 \ldots \mathbf{d}_3$ bezeichnet) werden dazu ins Netz eingefügt. Dabei wird $\mathbf{d}_1$ aktiviert, wenn mindestens eine Unit $\mathbf{del}_j$ aktiv ist. $\mathbf{d}_2$ wird durch $\mathbf{d}_1$ und $\mathbf{d}_3$ durch $\mathbf{d}_2$ aktiviert. Schließlich hat **cl** einen positiven Eingang von $\mathbf{d}_3$ und einen negativen von **rg**. Letzterer sorgt im Resetfall dafür, daß die Aktivierung von **cl** 0 ist. Anderenfalls wird **cl** über den Eingang von $\mathbf{d}_3$ positiv beeinflußt und wird aktiv.

Weiterhin soll die **cl**-Unit dann aktiv werden, wenn im F_0-Layer kein Eingabemuster anliegt. Dazu wird die $\mathbf{g}_2$-Unit über eine positive Verbindung mit **cl** verbunden. $\mathbf{g}_2$ wird – wie oben erläutert – dann aktiv, wenn keine Unit $\mathbf{inp}_i$ aktiviert ist. Sie hemmt zum einen den **rec**-Layer, zum anderen aktivert sie die **cl**-Unit, so daß ein unnötiges Propagieren verhindert wird.

Die **nc**-Unit bekommt positive Eingänge von allen lokalen Reset-Units. Wird ihr Netzeingang gleich M, so bedeutet das, daß alle lokalen Reset-Units aktiv sind und somit keine Klasse für das anliegende Muster gefunden werden kann: **nc** wird aktiv. Sie steht für den Zustand *not classifiable*.

Die Propagierungsvorschrift kann damit wie folgt geändert werden:

```
do {
     propagate();
     get_winner();
} while (( ! cl) && ( ! nc));
```

Schließlich bleibt noch die Funktion der Verbindungen von der **rg**-Unit zu den Recognition-Units zu erklären. Diese sind negativ gewichtet und stellen sicher, daß nach dem Auslösen eines Reset-Signals keine Information mehr in den Reset-Layer übernommen werden kann. So wird $\mathbf{rec}_j$ nicht fälschlich nochmals Gewinnerzelle, wenn bereits feststeht, daß sie nicht der Klasse entspricht, in die das angelegte Eingabemuster eingeordnet wird.

Das wäre ohne diese Verbindungen nicht sichergestellt, denn das Signal der lokalen Reset-Unit steht erst einen Zyklus nach dem Auslösen des globalen Reset-Signals zur Verfügung. $\mathbf{del}_j$ würde noch einemal aktiviert und in den folgenden Zyklen auch $\mathbf{d}_1$, $\mathbf{d}_2$ und $\mathbf{d}_3$. $\mathbf{d}_3$ wäre dann noch aktiv, wenn **rg** gerade inaktiv geworden ist. Das heißt **cl** würde fälschlich aktiviert werden.

Als zusätzlicher Nebeneffekt wird aufgrund dieser Verbindungen der untere Teil des Netzwerkes vom oberen für eine Zeit entkoppelt. Das bewirkt, daß in der Zeit, wo oben die Reset-Information vorwärtspropagiert wird und alle Units wieder ihre Normalzustände annehmen, unten bereits wieder die Kopie des Eingabemusters in den Comparison-Layer übernommen und schließlich das Reset-Signal aufgehoben werden kann.

Komplexitätsabschätzung

Diese Betrachtungen, die eher theoretischer Natur sind, würden ihrer Thematik nach eigentlich besser in Abschnitt 3.1 passen. Da die Komplexität jedoch von einigen Entwurfsentscheidungen abhängt – vor allem von der Realisierung der Suche nach der Gewinnerzelle – kann eine Abschätzung erst jetzt erfolgen.

Um die Zeitkomplexität abzuschätzen betrachten wir zunächst einen einzelnen Propagierungsschritt. Dazu muß zunächst die Zahl der Verbindungen ermittelt werden. Sie ist natürlich vor allem bestimmt durch die vollständige Vernetzung der F_1 und der F_2-Schicht und folglich $O(N \cdot M)$.

Für einen Propagierungsschritt sind also $O(N \cdot M)$ Multiplikationen durchzuführen und, da bei jeder Unit die Eingänge auch aufsummiert werden müssen, noch einmal soviele Additionen. Die Additionen ändern jedoch nichts an der Komplexität für einen Propagierungszyklus, denn $2 \cdot O(N \cdot M)$ bleibt $O(N \cdot M)$.

Auch die Suche nach dem Gewinner bringt keine Veränderung der Komplexität mit sich, denn pro Propagierungszyklus sind dazu M Schritte notwendig. Konkret:
$O(M) + O(N \cdot M) = O(N \cdot M)$.

Für einen Suchzyklus[2] benötigt man 10 Propagierungszyklen. Auch das ist nur ein linearer Faktor und ändert nichts an der Komplexität.

Im ungünstigsten Fall – dem, daß ein Muster nicht klassifizierbar ist – müssen M Suchzyklen durchlaufen werden. Erst dann sind alle Reset-Units aktiv und die **nc**-Unit kann anzeigen, daß das Eingabemuster nicht klassifizierbar ist. Folglich ist $O(N \cdot M^2)$ die Zeitkomplexität für ein zu trainierendes Muster.

Das ist jedoch ein die Tatsachen verfälschendes Ergebnis, denn dieser Wert gilt beim Trainieren des Netzes nur für den ersten Trainingszyklus. Danach wird für die meisten Eingabemuster bereits der erste Suchzyklus zum Erfolg führen, also eine Suche gar nicht mehr notwendig sein.

3.3 Implementierung und Integration von ART 1 in SNNS

Nachdem damit ein fundierter Entwurf feststeht, soll in diesem, etwas kürzeren Abschnitt noch auf die Besonderheiten bei der Implementierung und die Einbindung in die Struktur des SNNS (**S**tuttgarter **N**euronale **N**etze **S**imulator) eingegangen werden.

Allgemeine Kernfunktionen

Dem Simulatorkern des SNNS wurden neue Module hinzugefügt, die die ART-spezifischen Funktionen enthalten. Somit konnten Änderungen am bestehenden Kern auf ein Minimum beschränkt werden.

Das ART-Modul (`kr_art.c`) beinhaltet Funktionen, die von mehreren ART-Modellen verwendet werden können:

[2] Ein Suchzyklus dauert so lange, bis vom Anlegen eines Eingabemusters an festgestellt wurde, ob das Muster in die Klasse der gewinnenden Unit eingeordnet wird oder nicht.

- Eine Funktion zur synchronen Propagierung (`krart_prop_synch`).
- Eine Funktion zur algorithmischen Bestimmung der Gewinnerzelle (`krart_get_winner`).
- Eine Funktion um die Aktivierungen der Units auf ihre initialen Werte zurückzusetzen (`krart_reset_activations`).
- Eine Funktion um die Touch-Flags aller Units im Unit-Array zurückzusetzen (`krart_deleteTouchFlags`).
- Eine Funktion um das Unit-Array für das Sortieren zu initialisieren (`krart_init_sorting`)
- Eine Funktion, die die Zahl der Input-Units zurückliefert (`krart_get_NoOfInputUnits`).
- Eine Funktion, die überprüft, ob noch Units existieren, die keinem logischen Layer zugeordnet werden konnten (`krart_check_undeterminedUnits`).

Sie werden im folgenden nicht näher erläutert, da ihre Aufgaben klar sind oder für das Verständnis von untergeordneter Bedeutung.

Kernfunktionen für ART 1

Im Modul `kr_art1.c` sind die Funktionen implementiert, die nur für das ART1-Modell dienlich sind. Es sind dies `kra1_sort` und `kra1_init_i_act`. Sie werden von den weiter unten beschriebenen Funktionen zur Initialisierung, Propagierung oder zum Trainieren des Netzes aufgerufen.

Die Sortier-Routine `kra1_sort` erfüllt mehrere Aufgaben:

- Überprüfen, ob es sich bei einem gegebenem Netz strukturell um eine ART1-Topologie handelt.
- Alle Units daraufhin überprüfen, ob ihnen jeweils die richtige Aktivierungsfunktion zugeordnet wurde.
- Anlegen eines Feldes von Pointern (`topo_ptr_array`), die nacheinander auf die verschiedenen Layer von Units zeigen. (Siehe Abblidung 3.4).
- Initialisieren der festen Gewichte im Netz. Die trainierbaren Gewichte werden hier nicht initialisiert. (Siehe Initialisierungsfunktion).

Die Topologieprüfung ist eine der wichtigsten Funktionen um einen reibungslosen Betrieb mit einem ART1-Netz sicherzustellen, denn, wie an früherer Stelle bereits erwähnt, müssen die Netzwerke genau der definierten Struktur entsprechen, um richtig zu arbeiten.

Diese Funktion überprüft also ein ART1-Netz auf die genannten Punkte und liefert bei Nichterfüllung einer der Bedingungen einen SNNS-Fehlercode zurück.

Die Funktion `kra1_init_i_act` dient dazu, die initialen Aktivierungswerte der Units zu setzen. Diese werden durch die Funktion `krui_resetNet` (siehe [Zell92]), zur aktuellen Aktivierung. Ein

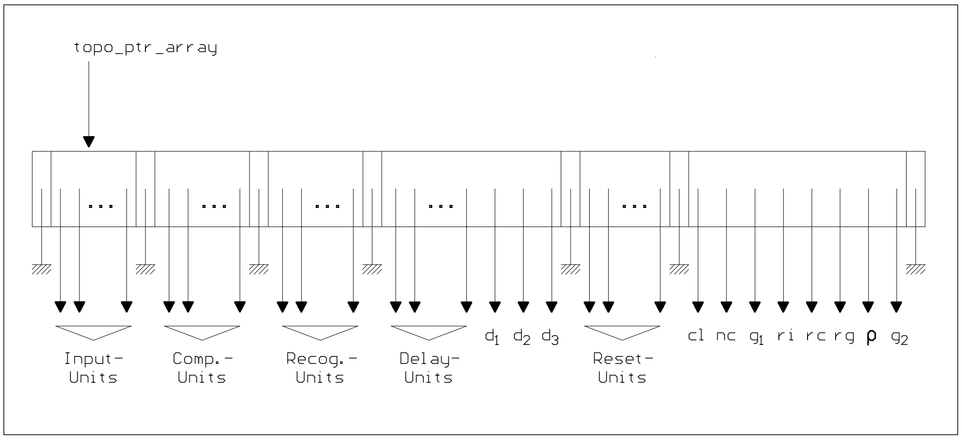


Abbildung 3.4: *Struktur des `topo_ptr_array` für ART1-Netzwerke. Am Anfang des Feldes ist ein NULL-Pointer eingetragen, dann die Zeiger auf die Input-Units, wieder ein NULL-Pointer, die Zeiger auf die Comparison-Units usw.*

solcher Reset der Aktivierungen wird bei jedem Anlegen eines neuen Musters an die Input-Units ausgelöst[3].

Als Parameter wird an `kra1_init_i_act` jeweils der Vigilance-Parameter ρ übergeben, dessen Wert als initiale Aktivierung in die ρ-Unit eingetragen wird.

Die Initialisierungsfunktion

Die Initialsierungsfunktion `ART1_Weights` ist dafür zuständig, die trainierbaren Gewichte in einem ART1-Netzwerk auf definierte Anfangswerte zu setzen.

Die Initialisierung der $F_2 \rightarrow F_1$ - Gewichte wird gemäß Gleichung 3.12 vorgenommen, das heißt sie werden alle auf den Wert 1.0 gesetzt.

Die Gewichte der Verbindungen zwischen F_1- und F_2-Layer ist etwas schwieriger zu erklären. Wir betrachten dazu noch einmal die Gleichungen 3.16, 3.17 und 3.20. Die Gewichte müssen also nach fortschreitendem Index in der F_2-Schicht kleiner werden und gleichzeitig die Bedingung der Ungleichung 3.20 erfüllen.

Zusammengefaßt läßt sich das wie folgt beschreiben:

$$0 < \alpha_M < \alpha_{M-1} < \ldots < \alpha_1 \leq \frac{1}{\beta + N}. \tag{3.25}$$

Um diese Bedingungen zu realisieren muß lediglich der Bruch auf der rechten Seite für fortschreitenden Index j etwas verkleinert und der so berechnete Wert α_j zugewiesen werden. Dazu führen wir eine neue Größe η ein und definieren

[3]Für rekurrente Netze, wie ART 1 ist ein definierter Aktivierungszustand als Ausgangspunkt für die Propagierung von entscheidender Bedeutung. Beginnt eine Propagierung unter falschen Anfangsbedingungen, so ist die Konvergenz des Versuchs nicht mehr gewährleistet.

$$\alpha_j \equiv \frac{1}{\beta + (1 + j\eta)N}. \tag{3.26}$$

Dabei berechnet sich η aus der Zahl der vorhandenen Klassen bzw. der Zahl der Recognition-Units und einem Parameter γ, der neben β an die Initialisierungsfunktion übergeben werden muß:

$$\eta \equiv \frac{\gamma}{M} \qquad \text{mit } \gamma > 0. \tag{3.27}$$

Sowohl für β als auch für γ ist ein Wert von 1.0 für die Initialisierung gebräuchlich. Für sehr große M ($M > 1000$) sollte γ etwas größer gewählt werden, damit η nicht zu klein wird und so fast keine Unterscheidung der Gewichte mehr zuließe.

Jedoch sollte γ nicht zu groß gewählt werden, da sonst für große j der Gewichtswert nahezu 0 wird. Der Wert für β sollte aus gleichem Grund nicht zu groß sein.

Der Parameter β wird bei der Initialisierung im Bias-Feld der Unit-Struktur gespeichert, um später für die Anpassung der Gewichte beim Lernvorgang wieder abrufbar zu sein. Das Bias-Feld wurde für die Speicherung dieses Wertes verwendet, weil es bei der Sicherung eines Netzwerkes mit in die Datei geschrieben wird. Der Wert steht so auch bereit, wenn ein initialisiertes Netz aus einer SNNS-Netzwerkdatei gelesen wird. Der Bias wird in seiner eigentlichen Funktion bei ART1-Netzen nicht benötigt.

ACHTUNG! Es sollte darauf geachtet werden, daß ART1-Netze **immer** mit der eben beschriebenen Funktion (`ART1_Weights`) initialisiert werden. Wird dazu eine andere SNNS-Initialisierungsfunktion verwendet, so ist das Verhalten des Netzes beim Lernen und Propagieren nicht vorhersagbar, da auch die festen Gewichte des Netzes mit anderen, als den für sie vorgesehenen Werten belegt werden können.

Die Lernfunktion

An die Funktion `ART1` ist als Lernparameter nur der Vigilancewert ρ zu übergeben. Sie propagiert ein Eingabemuster so lange, bis entweder feststeht, daß es nicht klassifizierbar ist oder bis eine passende Klasse dafür gefunden wurde.

Im zweiten Fall findet im Anschluß an die erfolgreiche Suche eine Anpassung der Gewichte gemäß den Gleichungen 3.11, ff. aus Abschnitt 3.1 statt.

Dabei wird der Parameter β, der für die Anpassung der $F_1 \rightarrow F_2$ - Gewichte notwendig ist, bereits bei der Initialisierung der trainierbaren Gewichte festgelegt und ist zum Zeitpunkt des Trainings bekannt. Er ist dem Bias-Feld einer beliebigen Unit zu entnehmen.

Die Lernfunktion gibt in dem von SNNS für Rückgabewerte vorgesehenen Feld von reellwertigen Zahlen die Nummern der Muster zurück, die klassifiziert werden konnten und diejenigen, für die eine Klassifikation nicht möglich war.

Das Feld der Rückgabewerte enthält dabei zunächst die Nummern der klassifizierten Muster, dann als Trennmarke den Wert -1.0 und schließlich die Nummern der nicht klassifizierbaren Muster.

Die Größe des Feldes kann erst zur Laufzeit bestimmt werden. Der dafür notwendige Speicherplatz wird zu Beginn der Lernfunktion bestimmt und allokiert. Es gilt:

Zahl der Rückgabewerte = Zahl der übergebenen Muster + 1.

Die Update-Funktionen

Für ART 1 wurden zwei Update-Funktionen implementiert:

1. `ART1_Stable` und
2. `ART1_Synchronous`.

`ART1_Stable` wird mit dem $\boxed{\text{TEST}}$-Knopf der SNNS-Oberfläche XGUI bedient. Sie propagiert ein Muster solange, bis es entweder klassifiziert wurde (**cl**-Unit) oder bis klar ist, daß es nicht klassifizierbar ist (**nc**-Unit). Kurz, es wird solange propagiert, bis ein stabiler Zustand erreicht wurde.

Die zweite Propagierungsfunktion (`ART1_Synchronous`) führt jeweils nur einen Propagierungsschritt aus. Mit ihr kann die Funktionsweise eines ART1-Netzes besonders gut nachvollzogen werden. Dazu wird zunächst der $\boxed{\text{RESET}}$-Knopf der SNNS-Oberfläche gedrückt, damit alle Units auf ihre initialen Werte zurückgesetzt werden.

Danach wird ein Eingabemuster in den F_0-Layer gebracht ($\boxed{\text{TEST}}$ oder $\boxed{<}$ bzw. $\boxed{>}$). Mit der $\boxed{\text{STEP}}$-Taste kann dann schrittweise die Propagierung verfolgt werden.

Beide Update-Funktionen benötigen wiederum einen Wert für ρ als Parameter. Es werden keine Werte zurückgeliefert.

Die Übertragungsfunktionen

Für die ART-Netzwerke wurde ein weiteres Modul eingeführt, das spezielle Aktivierungs-Funktionen für diese Netze beinhaltet (`arttr_f.c`). Diese Funktionen haben Zugriff auf von ihnen benötigte Werte aus den ART-Kern-Modulen (`kr_art.c, kr_art1.c`, siehe oben).

ART 1 benötigt nur eine solche spezielle Aktivierungsfunktion: diejenige für die **nc**-Unit: (`ACT_ART1_NC`). Ihre Aktivierung wird genau dann 1, wenn alle lokalen Reset-Units aktiv sind. Dann steht nämlich fest, daß das angelegte Muster nicht klassifizierbar ist.

Um diese Aktivierungsfunktion realisieren zu können muß ihr der Wert M bekannt sein. Leider gibt es in der Struktur des SNNS keine Möglichkeit Parameter an eine Aktivierungsfunktion übergeben zu können. Daher muß M global zugänglich gemacht werden.

Um jedoch eine weitgehende Kapselung dieses Wertes zu erreichen, wurde die Aktivierungsfunktion nicht in das ursprünglich dafür vorgesehene Modul `trans_f.c` integriert, sondern separiert.

Weitere Funktionen dieser Art werden in den anderen behandelten Modellen hinzukommen.

Die Schnittstellenfunktionen

Für ART 1 sind zwei relevante Schnittstellenfunktionen im Modul `art_ui.c` implementiert. Es sind dies

- `artui_getClassifiedStatus`
- `artui_getClassNo`
- `artui_getN`
- `artui_getM`

Die Schnittstellenfunktionen sind in Anhang A näher beschrieben.

3.4 Ein Versuch mit einem ART1-Netz

Einige Punkte der vorangegangenen Abschnitte sollen mit Hilfe dieses exemplarischen Versuchs verdeutlicht werden.

Das Netz

Das betrachtete Netz soll als Eingabe eine 5×7-Pixelmatrix erhalten. Die angelegten Muster stellen die 26 Buchstaben des Alphabets in der entsprechenden Auflösung dar. Weiterhin erhält das Netz 26 Recognition-Units.

Propagierung im untrainierten Netz

Nachdem das Netz mit der Funktion `ART1_Weights` ($\beta = \gamma = 1.0$) initialisiert worden ist, können wir zunächst betrachten, was geschieht, wenn wir ein Muster durch das Netz propagieren.

Dazu wählen wir die Update-Funktion `ART1_Stable` und propagieren damit einen Buchstaben durch das Netz. In Abbildung 3.5 ist das Ergebnis für das **H** als Eingabemuster dargestellt. Das Resultat wäre im Moment jedoch für alle Buchstaben das gleiche: die erste Klasse geht als Gewinner hervor. (Leicht abzulesen in der Delay-Schicht).

Bei Betrachtung des Recognition-Layers in Abbildung 3.5 fällt auf, daß die Aktivierung der Units in dieser Schicht von kleinem zu großem Index hin abnimmt. Dies ist natürlich darauf zurückzuführen, daß die Gewichte der Verbindungen zu Recognition-Units mit großem Index kleiner sind, als die mit kleinem Index.

Trainieren mit $\rho = 1.0$

Als nächstes soll versucht werden, ob es gelingt, mit $\rho = 1.0$ jeden Buchstaben in eine eigene Klasse einzuordnen. Dazu geben wir als Lernparameter 1.0 an und lassen zwei Lernzyklen für alle Muster ausführen.

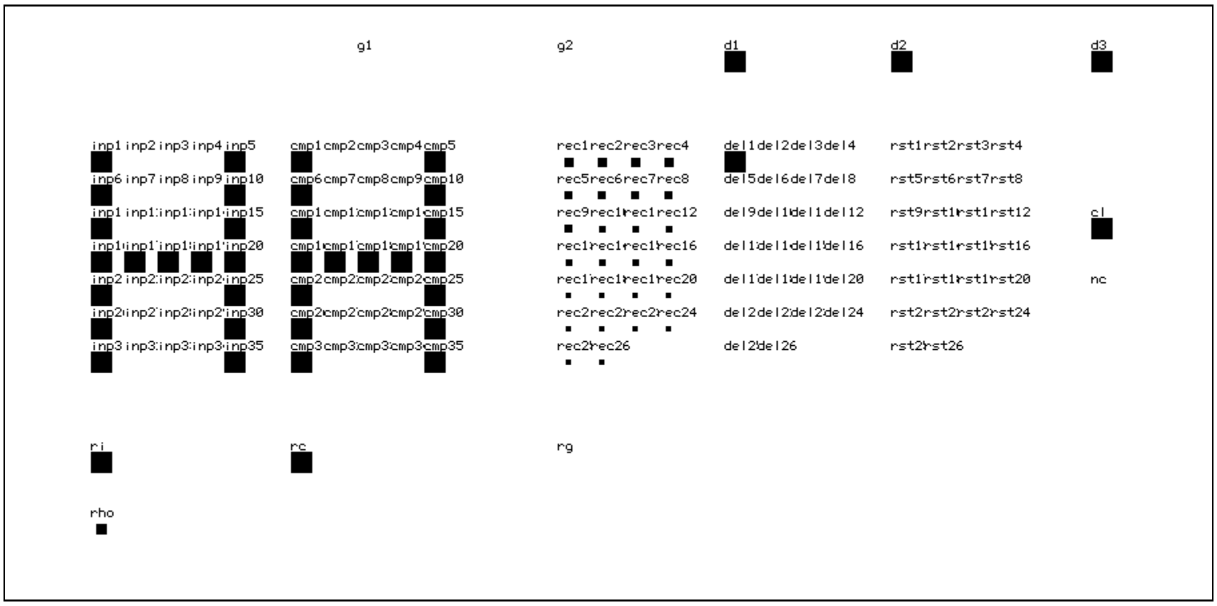


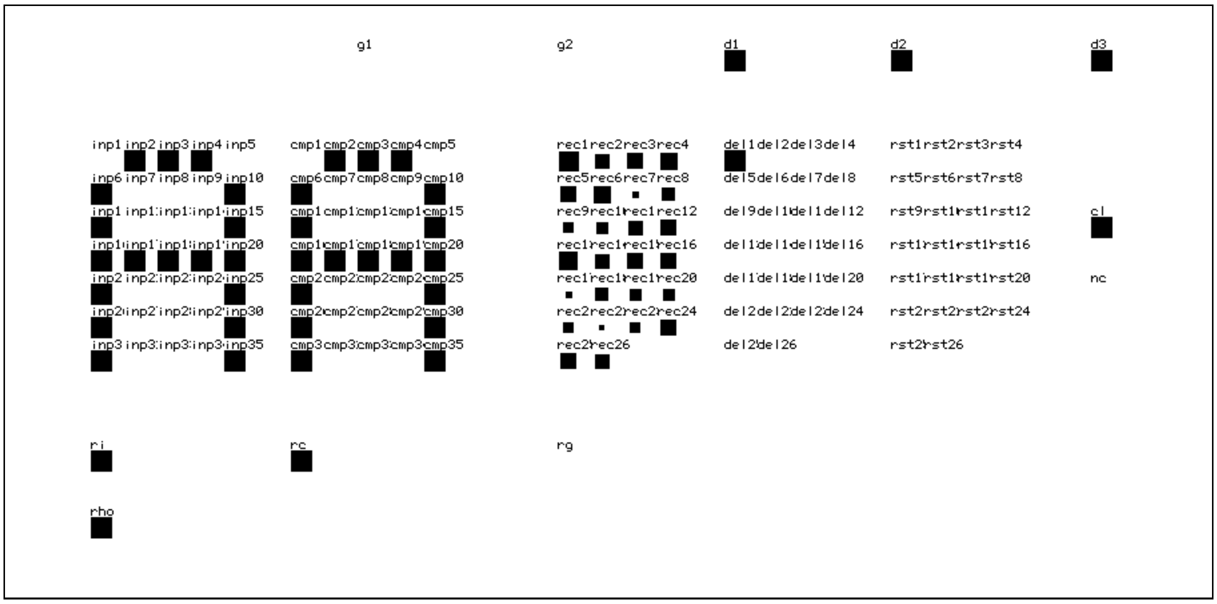
Abbildung 3.5: *Propagierung eines Buchstabens im untrainierten ART1-Netz. Die Initialisierung erfolgte mit den Parametern $\beta = \gamma = 1.0$.*

In Abbildung 3.6 ist exemplarisch für die Buchstaben **A** und **P** gezeigt, daß sie in verschiedene Klassen einsortiert wurden, zu erkennen an den verschiedenen Gewinnerzellen, die jeweils eine andere Delay-Unit aktivieren. Das Muster, das jeweils im Comparison-Layer zu sehen ist, entspricht zu 100% dem Eingabemuster. Das heißt, die Forderung nach einem vollständig zurückpropagierten Muster bei $\rho = 1.0$ ist erfüllt.

Trainieren mit $\rho = 0.5$

Schließlich betrachten wir noch im Vergleich dazu das Verhalten des Netzwerks bei einem Training mit $\rho = 0.5$. Wieder lassen wir mit diesem Vigilance-Parameter zwei Lernzyklen für alle Muster ausführen und betrachten anschließend erneut das Ergebnis für jeweils eine Propagierung der Buchstaben **A** bzw. **P**.

Eine Darstellung der resultierenden Aktivierungen findet sich in Abbildung 3.7. Jetzt sind beide Buchstaben in die gleiche Klasse eingeordnet worden. Der zurückpropagierte Durchschnitt ist jeweils deutlich im Comparison-Layer zu erkennen.



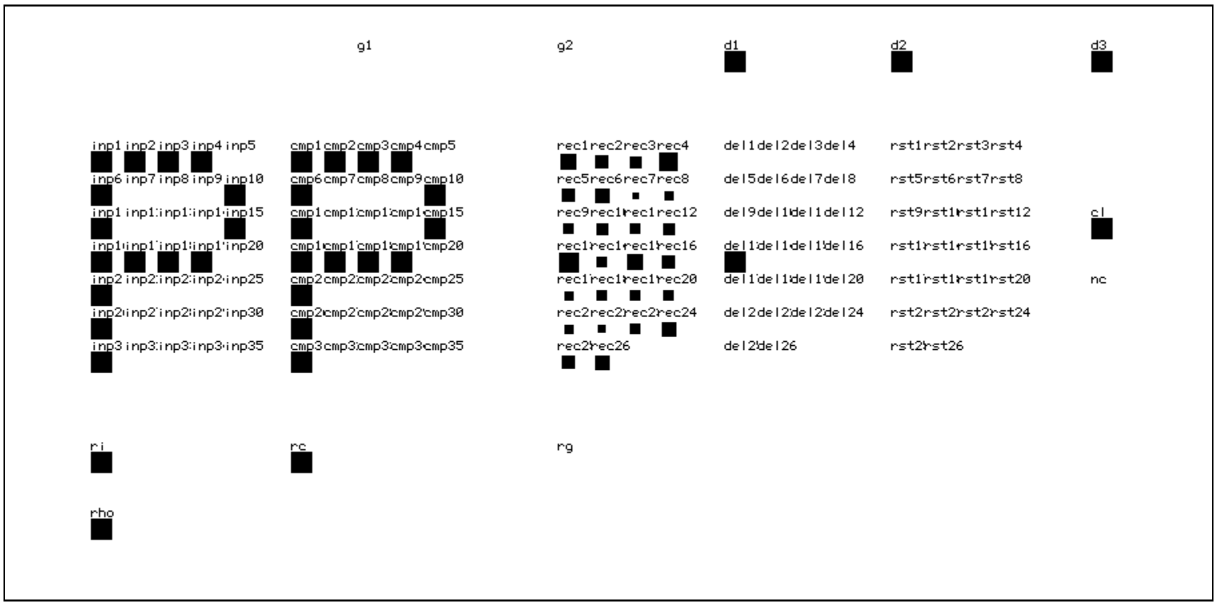
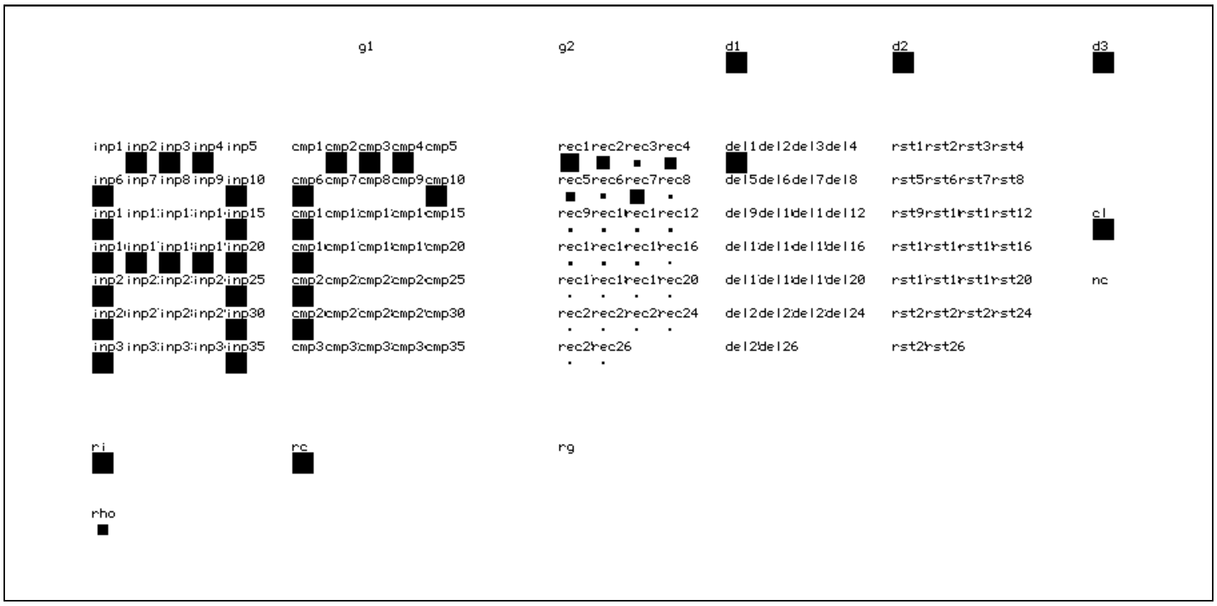


Abbildung 3.6: *Propagierung der Buchstaben* ***A*** *und* ***P*** *nach zwei Lernzyklen für alle Muster mit* $\rho = 1.0$



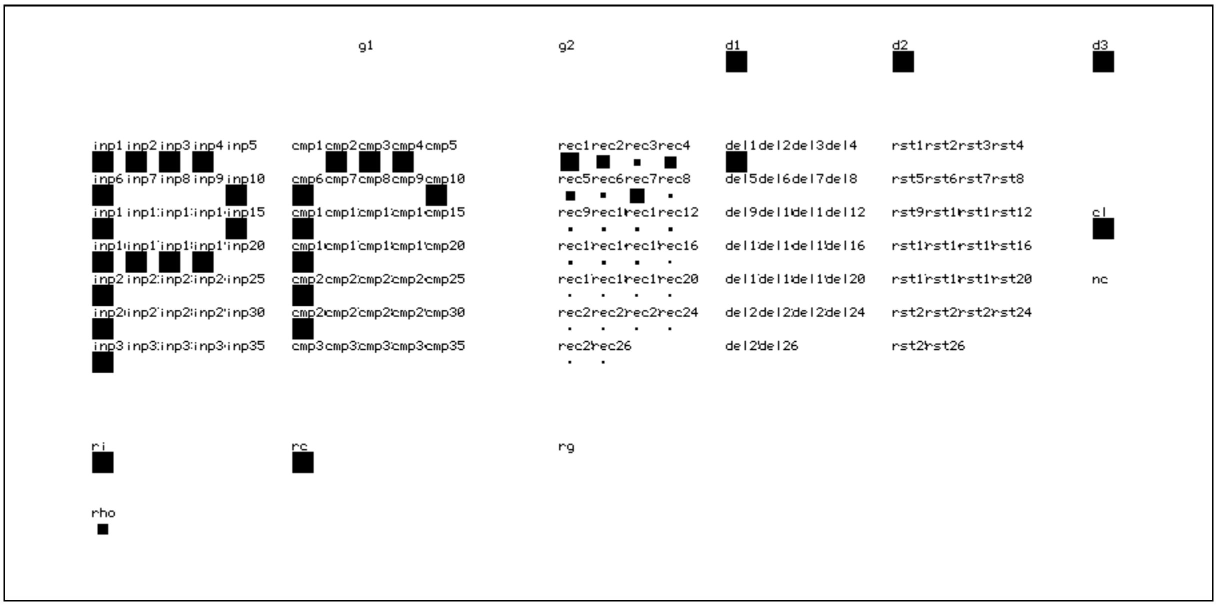


Abbildung 3.7: *Propagierung der Buchstaben **A** und **P** nach zwei Lernzyklen für alle Muster mit* $\rho = 0.5$

Kapitel 4

ART 2

Adaptive Resonance Theory für reellwertige Eingabemuster

Dem Wunsch nach einem Klassifizierungsverfahren für reellwertige Eingabemuster nachkommend, entwickelten Carpenter und Grossberg ART 2. In [Carpenter87b] stellen sie dieses Verfahren vor.

Um ART 2 zu implementieren ist es sehr hilfreich, wenn man auf Menschen trifft, die mit dem Verfahren schon etwas Erfahrung haben und so in der Lage sind, Anforderungen und Funktionsweise klarer zu formulieren, als dies in der Literatur der Fall ist.

Mein besonderer Dank gilt in diesem Zusammenhang Frau Heike Speckmann von der Fakultät Informatik der Universität Tübingen und Herrn Markus Ritt (Student am gleichen Institut), die mir mit ihren Erfahrungen zur Seite standen und so nicht unerheblichen Anteil an der erfolgreichen Realisierung des Verfahrens haben.

Weiter danke ich auch Herrn Peter Weierich von der FORWISS-Gruppe an der Universität Erlangen und Herrn Martin Reczko vom Deutschen Krebsforschungszentrum (DKFZ), die mir ebenfalls Anregungen zur Implementierung gaben.

Die folgenden Ausführungen sollen etwas Licht in die ART2-Theorie bringen und einen Vorschlag zur Realisierung beinhalten. Anhand eines Beispiels werden am Ende dieses Kapitels Stärken und Schwächen des Verfahrens erörtert.

4.1 Informelle Beschreibung von ART 2

Eine Kernaussage soll zu Beginn der Erläuterungen zum ART2-Klassifikationsverfahren stehen:

> ART 2 arbeitet wie ART 1. Es werden jedoch keine binären Vektoren, sondern reellwertige Muster verarbeitet.

So einfach sich das zunächst anhören mag, so komplex wird das Problem, überlegt man sich, was die F_1-Schicht hier leisten soll.

In ART 1 konnte ein Maß für die Ähnlichkeit zwischen angelegtem Eingabemuster und dem, das sich durch die Top-Down-Propagierung in F_1 ergab einfach dadurch definiert werden, daß

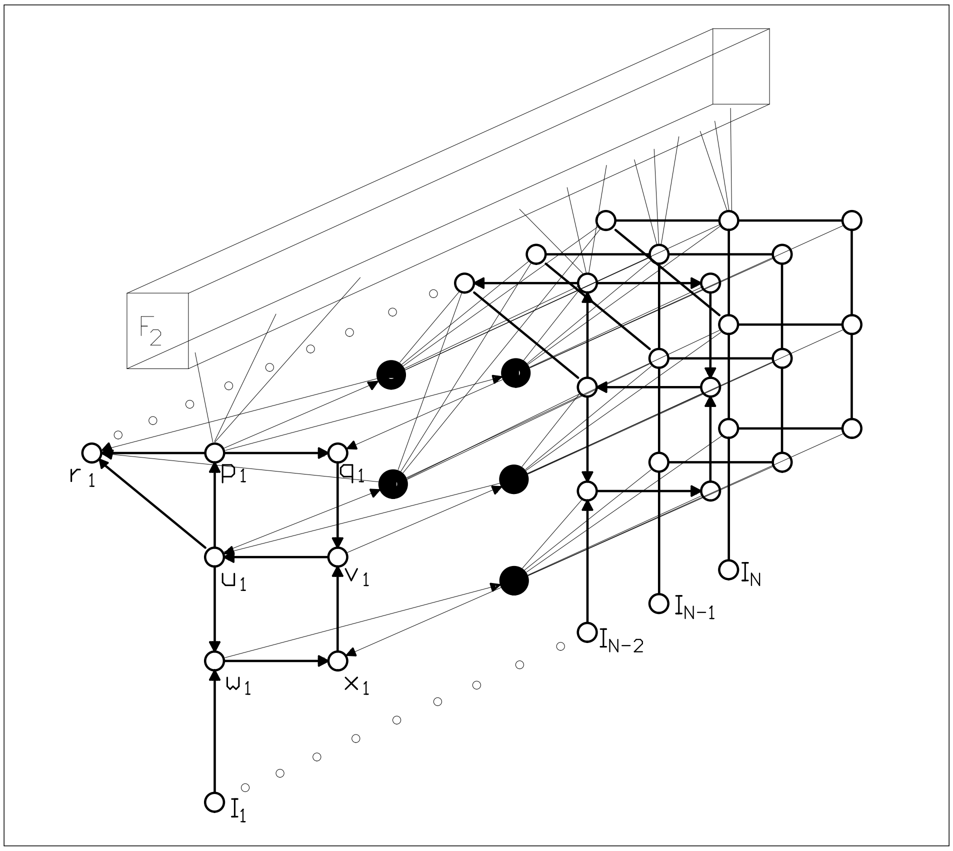


Abbildung 4.1: *Der Aufbau der F_1-Schicht im ART2-Netzwerk. Für jede F_0-Unit gibt es sieben Units im F_1-Layer. Die ausgefüllt dargestellten Neuronen erhalten jeweils Eingänge von allen Zellen eines Typs der F_1-Schicht. Sie liefern jeweils die euklidische Norm ihres Eingangsvektors an alle Units eines anderen Typs der F_1-Schicht. Unten beispielsweise erhält die Norm-Zelle Eingänge von allen Units w_i und liefert die euklidische Norm des Vektors $\boldsymbol{w}$ an alle x_i. So wird Vektor $\boldsymbol{x}$ in die gleiche Richtung weisen, wie $\boldsymbol{w}$, jedoch auf Länge 1 normiert sein.*

die Zahl der aktiven Units in F_1 und F_0 verglichen wurden.

Für reellwertige Vektoren ist das naturgemäß nicht in dieser Weise realisierbar, denn eine 2/3-Regel läßt sich hier schwerlich vorstellen. Bleiben soll aber das Verfahren an sich, nämlich das Propagieren des Eingabemusters in die F_1-Schicht, von dort in die F_2-Ebene, die Ermittlung eines Gewinners und die Rückpropagierung dessen bereits gelernten Wissens nach F_1, sowie eine Art Durchschnittsbildung zwischen Bottom-Up- und Top-Down-Vektor.

Um dies realisieren zu können bedarf es einer komplexeren F_1-Schicht, die es erlaubt, den Bottom-Up- und den Top-Down-Vektor zu überlagern und schließlich einen Ersatz für die 2/3-Regel zu finden. F_1 wird dazu aus drei Ebenen aufgebaut, die eine Kombination von Normierung, Gain-Kontrolle und Lernmechanismus bieten. In Abbildung 4.1 ist der Aufbau der F_1-Schicht dargestellt. In Abschnitt 4.2 wird sie noch detailliert zu behandeln sein.

Die Konstruktion des ART2-Netzes ist so gestaltet, daß die Eigenschaften, die ART 1 bietet erhalten bleiben. So ändert sich gegenüber ART 1 hauptsächlich die F_1-Schicht, was die globalen Vorteile, die ART bietet nicht berührt. Es ist dies vor allem die Gewährleistung der Stabilität bei bleibender Plastizität.

Alle Zellen in der F_1-Schicht folgen einer Membran-Differentialgleichung, die ebenfalls Thema in Abschnitt 4.2 sein wird. Durch sie wird eine Stablisierung der F_1-Zellaktivierungen bereits nach wenigen Propagierungszyklen erreicht.

Ein Eingabemuster wird zunächst in die F_0-Schicht kopiert und dann an die Zellen $\mathbf{w}_i$ unverändert weitergegeben. Die Aktivierungen der Units $\mathbf{x}_i$ stellen den normierten Vektor $\mathbf{w}$ dar. Sie geben ihre Information über eine Funktion f zur Rauschverminderung und Kontrastverstärkung an die $\mathbf{v}_i$-Zellen weiter. Diese Werte werden erneut normiert und nach $\mathbf{u}$ übermittelt. Über die Units $\mathbf{p}_i$, eine weitere Normierungsstufe und $\mathbf{q}_i$ ist das Eingabemuster dann erstmals über alle F_1-Units verteilt.

Nach wenigen Propagierungszyklen ist F_1 stabil. Dann geben die Units $\mathbf{p}_i$ ihre Ausgabe weiter an die F_2-Units, wo eine Gewinnerzelle ermittelt wird. Wie bei ART 1 haben die F_2-Zellen auch in die umgekehrte Richtung Verbindungen zu den $\mathbf{p}_i$. Diese sind mit 0 initialisiert, so daß eine noch unbelegte Klasse in F_2 keinen Einfluß auf den F_1-Layer nimmt.

Die Top-Down-Propagierung bringt in diesem Fall also keine neue Aktivierung der $\mathbf{p}_i$-Zellen. Folglich bleibt F_1 stabil und sofort kann das Anpassen der Gewichte beginnen.

Ist die Gewinnerklasse in F_2 dagegen bereits belegt, so werden die $\mathbf{p}_i$ über die von ihr ausgehenden Top-Down-Verbindungen beeinflußt. In F_1 beginnt erneut die Suche nach einem stabilen Zustand, der wieder nach relativ kurzer Zeit erreicht wird.

Somit ist die Überlagerung von Bottom-Up- und Top-Down-Information realisiert, denn von „unten" steht weiterhin der Eingabevektor an, während von „oben" das Wissen einer Klasse ausgelesen wird. Beide nehmen Einfluß auf die Aktivierungen im F_1-Layer.

Schließlich muß überprüft werden, ob der Vektor, der sich in der Mittelschicht[1] von F_1 ergeben hat (also Vektor $\mathbf{u}$) und Vektor $\mathbf{p}$ ähnlich genug sind. Für die Ähnlichkeit ist vor allem der Winkel zwischen diesen beiden Vektoren ausschlaggebend.

Wie bei ART 1 wird bei einer positiven Entscheidung der anliegende Eingabevektor in die aktuelle Gewinnerklasse eingeordnet, im anderen Fall wird ein Reset ausgelöst und damit ein neuer Suchzyklus gestartet. Die F_2-Gewinnerzelle wird dazu blockiert und kann im folgenden nicht mehr gewinnen.

Durch die oben beschriebene Art der Überlagerung wird sichergestellt, daß keine Gewichtsanpassung stattfindet, wenn der anliegende Eingabevektor eine Obermenge des Top-Down-Vektors ist. Nehmen wir zur Veranschaulichung binäre Vektoren als Eingabemuster an. Der Vektor (1 1 0 0 0) sei in Klasse 1 eingeordnet worden.

Legt man jetzt den Vektor (1 1 1 0 0) an, so werden in der Bottom-Up-Phase auch die zur dritten Eingabezelle gehörenden F_1-Zellen aktiv. Nehmen wir weiter an, die erste Klasse in F_2 gewinnt erneut. Dann bekommt $\mathbf{p}_3$ bei der Rückwärtspropagierung von „oben" jetzt kein zusätzliches, positives Signal.

Da im dritten F_1-Cluster nur das ursprüngliche Bottom-Up-Signal vorhanden ist, die beiden ersten Cluster jedoch wegen der Top-Down-Propagierung wesentlich höhere Werte angenommen haben, wird $\mathbf{u}_3$ aufgrund der Normierung immer weiter gehemmt und endet schließlich auf einem Minimalwert. Eine Beeinflussung des Gewichts von der Gewinnerklasse zu $\mathbf{p}_3$ ist somit ausgeschlossen.

[1] Die Mittelschicht zur Ermittlung der Ähnlichkeit zu wählen ist nur eine Möglichkeit. Alternativ kann auch der Eingabevektor dazu herangezogen werden.

Um diesen Abschnitt zu beschließen soll noch einmal der stark ausgeprägete zweiphasige Charakter von ART 2 hervorgehoben werden. Wichtig ist hier, daß Entscheidungen jeweils erst dann gefällt werden dürfen, wenn sich F_1 stabilisiert hat. Dazu muß ein Suchzyklus streng in zwei Abschnitte gegliedert werden: zum einen der Bottom-Up-Abschnitt, in dem das Eingabemuster nach F_1 propagiert wird. Nach der Stabilisierung und der anschließenden Ermittlung des F_2-Gewinnerknotens dann die Top-Down-Phase. Auch hier darf erst nachdem F_1 einen stabilen Zustand eingenommen hat über einen Reset entschieden werden.

4.2 ART 2 Theorie

Wurde im vorangegangenen Abschnitt eher informell beschrieben, wie der Klassifkationszyklus im ART2-Netz vor sich geht, so sollen jetzt die mathematischen Hintergründe dazu bereitgestellt werden[2].

Aktivierung im F_1-Layer

Die Aktivierung V_i sämtlicher F_1-Units basiert – das wurde bereits erwähnt – auf einer Membran-Differentialgleichung, wie sie in Gleichung 4.1 dargestellt ist. Sie sorgt dafür, daß die Units im F_1-Layer in Schwingung versetzt werden, schließlich – weil gegeneinander rückgekoppelt – in einen Resonanzzustand gelangen und dort verharren, bis das Erregersignal nicht mehr ansteht.

$$\varepsilon \frac{d}{dt} V_i = -AV_i + (1 - BV_i)J_i^+ - (C + DV_i)J_i^- \quad \forall\, 1 \leq i \leq N. \tag{4.1}$$

Dabei ist J_i^+ der erregende, J_i^- der hemmende Netzeingang der jeweiligen Zelle. Ist der Netzeingang 0, so ist auch die Aktivierung $V_i = 0$. Der Parameter ε repräsentiert das Verhältnis zwischen den Stabilisierungszeiten der Aktivierungen in der F_1-Schicht und der Gewichte zwischen F_1- und fz-Schicht. Es gilt daher naturgemäß:

$$0 < \varepsilon \ll 1. \tag{4.2}$$

Für die ART2-Gleichungen gilt außerdem $B \equiv 0$ und $C \equiv 0$. Weiter nehmen wir an $\varepsilon \to 0$, dann reduziert sich Gleichung 4.1 auf:

$$V_i = \frac{J_i^+}{A + DJ_i^-}. \tag{4.3}$$

Gleichung 4.3 gibt in dieser Form die Aktivierungsfunktionen sämtlicher F_1-Units in allgemeiner Weise wieder.

Betrachtet man Abbildung 4.2, so findet man dort den Cluster i der F_0- und F_1-Schicht aus dem Netz herausgegriffen. Die Eingänge der jeweiligen Units in Gleichung 4.3 eingesetzt, ergibt die konkreten Aktivierungsfunktionen der F_1-Units:

[2] siehe dazu auch [Carpenter87b]

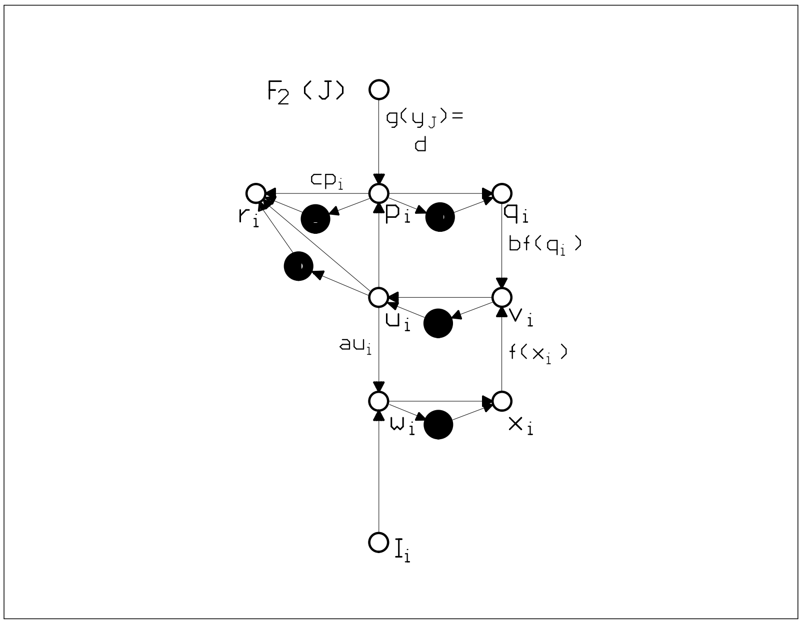


Abbildung 4.2: *Ein Cluster der F_0- und F_1-Schicht.*

$$\begin{aligned}
p_i &= u_i + \sum_j g(y_j) z_{ji}, && (4.4)\\
q_i &= \frac{p_i}{e + ||\mathbf{p}||}, && (4.5)\\
u_i &= \frac{v_i}{e + ||\mathbf{v}||}, && (4.6)\\
v_i &= f(x_i) + b f(q_i), && (4.7)\\
w_i &= I_i + a u_i, && (4.8)\\
x_i &= \frac{w_i}{e + ||\mathbf{w}||}. && (4.9)
\end{aligned}$$

Dabei ist $||\mathbf{V}||$ die L_2-Norm eines Vektors $\mathbf{V}$ und y_j der Aktivierungswert der j-ten F_2-Zelle. a, b und e sind Parameter, auf die am Ende dieses Abschnitts eingegangen wird. Die Signalfunktion f wird normalerweise wie folgt gewählt:

$$f(x) = \begin{cases} \frac{2\Theta x^2}{x^2+\Theta^2} & \text{wenn } 0 \le x \le \Theta, \\ x & \text{sonst.} \end{cases} \tag{4.10}$$

Sie ist stetig differenzierbar. Alternativ kann auch die teilweise lineare Funktion

$$f(x) = \begin{cases} 0 & \text{wenn } 0 \le x < \Theta, \\ x & \text{sonst.} \end{cases} \tag{4.11}$$

verwendet werden. Θ ist ein weiterer Parameter (Beschreibung am Ende dieses Abschnitts). Zweck der Funktion ist es, kleine x-Werte auszublenden. Da sich die Aktivierungen der Zellen

$\mathbf{x}_i$ und $\mathbf{q}_i$ immer zwischen 0 und 1 befinden (Normierung) bleiben auch die Funktionswerte von f zwischen 0 und 1.

Eine weitere Möglichkeit besteht darin, f so zu wählen, daß für hohe Werte von x $(0 \ll x \leq 1)$ eine Sättigung eintritt. Das hat den Effekt, daß Spitzen von Mustern abgeflacht werden und so nicht übermäßig in die Bewertung eingehen.

Aktivierung im F_2-Layer

Die Klassifizierungsschicht ist in ART 2 die gleiche, wie in ART 1. Sie enthält M Klassenzellen, die untereinander ein winner-takes-all-Netz bilden. Die Unit mit dem höchsten Netzeingang gewinnt und gibt als einzige eine positive Ausgabe auf die Top-Down-Verbindungen.

Mit T_j soll wieder die Netzeingabe in die j-te F_2-Zelle bezeichnet werden. Dann gilt:

$$T_j = \sum_i p_i z_{ij}. \tag{4.12}$$

Die Gewinnerzelle J ist definiert durch

$$T_J = \max_{1 \leq j \leq M} \{T_j\}. \tag{4.13}$$

Die Ausgabefunktion der F_2-Zellen wird dann wie folgt definiert:

$$g(y_J) = \begin{cases} d & \text{wenn } J \text{ Gewinnerzelle ist,} \\ 0 & \text{sonst.} \end{cases} \tag{4.14}$$

Dabei ist d ein weiterer Parameter (Beschreibung am Ende dieses Abschnitts).

Mit Gleichung 4.14 reduziert sich die Aktivierungsfunktion der p_i aus Gleichung 4.4 dann auf

$$p_i = \begin{cases} u_i & \text{wenn } F_2 \text{ inaktiv ist,} \\ u_i + dz_{Ji} & \text{wenn die } J\text{-te } F_2\text{-Zelle aktiv ist.} \end{cases} \tag{4.15}$$

Lernregeln

Die Anpassung der Gewichte zwischen F_1- und F_2-Schicht erfolgt nach den folgenden Differentialgleichungen.

1. $F_2 \rightarrow F_1$:

$$\frac{d}{dt} z_{ji} = g(y_j)(p_i - z_{ji}), \tag{4.16}$$

2. $F_1 \rightarrow F_2$:

$$\frac{d}{dt} z_{ij} = g(y_j)(p_i - z_{ij}). \tag{4.17}$$

Da $g(y_j)$ für alle $j \neq J$ und damit auch die Gewichtsänderung gleich 0 ist, betrachtet man nur noch die Gewinnerzelle J. Für sie lassen sich die Gleichungen 4.16 und 4.17 mit Hilfe von 4.15 und 4.14 umschreiben zu

1. $F_2 \rightarrow F_1$:
$$\frac{d}{dt} z_{Ji} = d(1-d)\left(\frac{u_i}{1-d} - z_{Ji}\right), \tag{4.18}$$
2. $F_1 \rightarrow F_2$:
$$\frac{d}{dt} z_{iJ} = d(1-d)\left(\frac{u_i}{1-d} - z_{iJ}\right). \tag{4.19}$$

Für den Parameter d werden damit sinnvolle Grenzen definiert: $0 < d < 1$. Wegen $dz_{ji}/dt \rightarrow 0$ und $dz_{ij}/dt \rightarrow 0$ lassen sich die Gleichungen 4.18 und 4.19 noch einmal vereinfachen zu

1. $F_2 \rightarrow F_1$:
$$z_{Ji} = \frac{u_i}{1-d}, \tag{4.20}$$
2. $F_1 \rightarrow F_2$:
$$z_{iJ} = \frac{u_i}{1-d}. \tag{4.21}$$

Die Reset-Kontrolle

Wie in Abschnitt 4.1 bereits erwähnt, wird zur Entscheidung über einen Reset vor allem der Winkel zwischen Vektor $\mathbf{u}$ und Vektor $\mathbf{p}$ herangezogen.

Unerwähnt blieb bis jetzt die Aktivierungsfunktion der Units $\mathbf{r}_i$ in den Abbildungen 4.1 und 4.2. Ihr Name läßt bereits erkennen, daß es sich dabei um Zellen handelt, die für die Entscheidung über einen Reset relevant sind. Es gilt:

$$r_i = \frac{u_i + cp_i}{e + ||\mathbf{u}|| + ||c\mathbf{p}||}. \tag{4.22}$$

Es gilt dann:

$$\text{Reset} \Leftrightarrow \frac{\rho}{e + ||\mathbf{r}||} > 1. \tag{4.23}$$

Dabei liegt der Ähnlichkeitsparameter ρ – das ist nicht anders als bei ART 1 – zwischen 0 und 1. Um ihn jedoch richtig wählen zu können, sollte der Anwender wirklich verstanden haben, was die Gleichungen 4.22 und 4.23 bedeuten.

Zur Veranschaulichung nehmen wir den zweidimensionalen Fall an, also zwei Eingabeneuronen. Diese Annahme stellt keine Einschränkung dar und dient lediglich dem besseren Verständnis. Weiterhin definieren wir $\mathbf{s}$ als den resultierenden Vektor aus der Addition von $\mathbf{u}$ und $c\mathbf{p}$. (Siehe Abbildung 4.3))

$$\mathbf{s} = \mathbf{u} + c\mathbf{p} \tag{4.24}$$

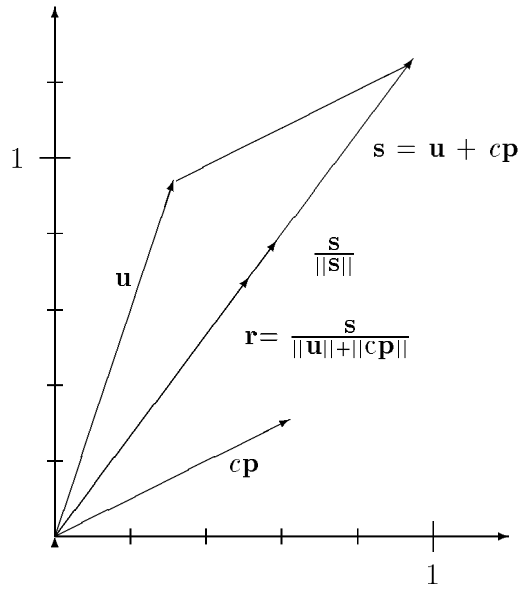


Abbildung 4.3: *Der resultierende Vektor* $\mathbf{s}$ *aus* $\mathbf{u}$ *und* $c\mathbf{p}$*. Dessen normiertes Pendant und der Fehlervektor* $\mathbf{r}$

Implizit wurde bis jetzt bereits vorausgesetzt, daß sämtliche Eingänge positiv sind. Diese Einschränkung muß übrigens tatsächlich gelten, um ein einwandfreies Funktionieren von ART 2 zu gewährleisten. Deswegen ist auch die Signalfunktion f nur für Werte aus $\mathbf{R}^+$ definiert.

Eine wirkliche Einschränkung stellt diese Bedingung jedoch nicht dar, kann doch durch eine einfache Translation erreicht werden, daß sämtliche Punkte in den positiven Koordinatenbereich verschoben werden.

Wegen der Positivität sämtlicher Koordinaten weiß man, daß der Winkel zwischen $\mathbf{u}$ und $\mathbf{p}$ maximal 90^o beträgt. Für diesen Winkel ist bei fester Länge $||c\mathbf{p}||$ der Fehler am größten.

Betrachtet man nun noch einmal Gleichung 4.22, so stellt man fest, daß der eigentliche Fehlervektor $\mathbf{r}$ aus dem oben definierten $\mathbf{s}$ und der Norm von $\mathbf{u}$ und $\mathbf{p}$ berechnet wird (dabei gehen wir im folgenden der Einfachheit halber von $e = 0$ aus):

$$\mathbf{r} = \frac{\mathbf{s}}{||\mathbf{u}|| + ||c\mathbf{p}||} \tag{4.25}$$

Wegen der Dreiecksungleichung gilt $||\mathbf{r}|| \leq 1$. Für einen Winkel von 90^o zwischen $\mathbf{u}$ und $\mathbf{p}$ ist $||\mathbf{r}||$ am kleinsten, der Fehler also am größten. Gehen wir weiter vom ungünstigsten Fall $||c\mathbf{p}|| = ||\mathbf{u}|| = 1$ aus, dann gilt

$$||\mathbf{s}|| = \sqrt{||\mathbf{u}||^2 + ||c\mathbf{p}||^2} = \sqrt{2} \quad \text{und damit} \tag{4.26}$$

$$||\mathbf{r}|| = \frac{1}{2}\sqrt{2}. \tag{4.27}$$

Damit sollte veranschaulicht werden, daß für alle Werte $0 \leq \rho < (1/2)\sqrt{2}$ der gleiche Effekt auftritt: Die Reset-Kontrolle wird komplett abgeschaltet. Eine Klassifikation erfolgt vollständig ohne Rückversicherung. Der Durchschnittsvektor in F_1 wird in jedem Fall akzeptiert. Ein Überprüfung des Durchschnittsvektors erfolgt nur im Bereich $(1/2)\sqrt{2} \leq \rho \leq 1$.

Initialisierung der Gewichte

Auch für ART 2 ist die Wahl der initialen Gewichte der Verbindungen zwischen F_1- und F_2-Schicht von ausschlaggebender Bedeutung.

Die $\mathbf{F}_2 \rightarrow \mathbf{F}_1$- Gewichte

Wird eine noch unbelegte Kategorie in F_2 als Gewinner ermittelt, so soll in der Top-Down-Phase kein Einfluß auf F_1 genommen werden. Schließlich will man den momentan anliegenden Eingabevektor exakt in diese neue Klasse aufgenommen wissen.

So ist es sinnvoll, die $F_1 \rightarrow F_2$-Gewichte mit 0 zu initialisieren.

$$z_{ji}(0) = 0. \tag{4.28}$$

Die $\mathbf{F}_1 \rightarrow \mathbf{F}_2$- Gewichte

Die Wahl der initialen Bottom-Up-Gewichte ist vor allem durch einen Aspekt geprägt: wurde ein Muster trainiert, so darf es bei der nächsten Präsentation keine unbelegte Klasse aktivieren. Vielmehr muß die F_2-Unit deutlich gewinnen, die beim ersten Trainingszyklus gewählt wurde.

Die Norm des anfänglichen Gewichtsvektors muß also kleiner sein, als die, die er nach einem oder mehereren Trainingszyklen hat. Wegen Gleichung 4.21 gilt

$$||\mathbf{z}^J|| \rightarrow \left|\left|\frac{\mathbf{u}}{1-d}\right|\right| = \frac{1}{1-d}. \tag{4.29}$$

Vorausgesetzt, alle Anfangswerte $z_{ij}(0)$ werden gleich gewählt, bedeutet das

$$z_{ij}(0) \leq \frac{1}{(1-d)\sqrt{N}} \qquad \forall\, 1 \leq i \leq N, 1 \leq j \leq M. \tag{4.30}$$

Wird in Gleichung 4.30 Gleichheit gewählt, so wird ART 2 so viel wie möglich noch unbelegte Klassen wählen.

Wahl der Parameter

Die Vielzahl an Parametern in einem ART2-Netz macht es notwendig, hierüber einige Worte zu verlieren. In der Literatur sind nur wenig Angaben über Auswirkungen auf das Verhalten der Netze bei Variierung der Parameter zu finden.

Diese wenigen Hinweise und eigene Erfahrungen sollen in diesem Teilabschnitt ohne Anspruch auf Vollständigkeit zusammengetragen werden.

Zunächst eine Liste der Parameter:

- ρ Ähnlichkeitsparameter. Er gibt an, wie „lang" der Fehlervektor $\mathbf{r}$ mindestens sein muß, um keinen Reset auszulösen.
- a Stärke der Beeinflussung der unteren Schicht in F_1 durch die Mittelschicht.
- b Stärke der Beeinflussung der mittleren Schicht in F_1 durch die obere Schicht.
- c Längenanteil des $\mathbf{p}$-Vektors, der zur Berechnung des Fehlers verwendet wird.
- d Ausgabewert der F_2-Gewinnerzelle.
- e Sorgt bei der Implementierung dafür, daß kein Überlauf bei der Division durch die Norm von 0-Vektoren auftritt.
- Θ $0 \leq x \leq \Theta$ ist der Bereich, in dem die Aktivierungen der Units $\mathbf{x}_i$ und $\mathbf{q}_i$ sehr geringe oder keine Auswirkung auf die Mittelschicht des F_1-Layers haben.

Diese Parameter unterliegen gewissen Bedingungen. Das heißt, es existieren Grenzen, innerhalb derer sie gewählt und Abhägigkeiten, die beachtet werden müssen.

1. **Grenzen für Parameter ρ**
   Wie oben bereits erwähnt, ist für ρ der Bereich
   $$0 \leq \rho \leq 1 \tag{4.31}$$
   definiert. Sinnvoll ist jedoch, wie gezeigt wurde nur der Bereich
   $$\frac{1}{2}\sqrt{2} \leq \rho \leq 1. \tag{4.32}$$

2. **Grenzen für Parameter a**
   Parameter a gibt die Wichtigkeit der von F_2 nach F_1 propagierten Erwartung relativ zum Eingabevektor an. Für a gilt
   $$a > 0 \tag{4.33}$$
   Üblicherweise wird $a \gg 1$ gewählt um eine schnelle Stabilisierung in F_1 zu gewährleisten, insbesondere dann, wenn eine Obermenge des in der Gewinnerklasse gespeicherten Wissens als Eingabevektor anliegt. (Siehe Abschnitt 4.1).

3. **Grenzen für Parameter b**
   Für b gilt weitgehend das für Parameter a Gesagte. Ein großer Wert ist in diesem Fall noch wichtiger, da kleine b zu einem instabilen Verhalten des Netzes führen können ([Carpenter87b]).
   $$b > 0 \tag{4.34}$$
   $$\text{üblicherweise: } b \gg 1. \tag{4.35}$$

4. **Grenzen für Parameter c**
   $$0 < c < 1 \tag{4.36}$$

5. **Grenzen für Parameter d**
   $$0 < d < 1 \tag{4.37}$$

6. **Abhängigkeiten für die Parameter c und d**
   Die beiden Parameter c und d bedingen sich gegenseitig. Um diese Abhängigkeit zu erklären, betrachten wir noch einmal den Fehlervektor $\mathbf{r}$, wie er sich aus Gleichung 4.22 ergibt.

   Setzt man für die $\mathbf{p}_i$ die Werte aus Gleichung 4.15 ein, so ergibt sich unter der Voraussetzung, daß Unit J in F_2 aktiv ist:

$$\mathbf{r} = \frac{(1+c)\mathbf{u} + cd\mathbf{z}_J}{||\mathbf{u}|| + ||c\mathbf{u} + cd\mathbf{z}_J||} \tag{4.38}$$

   Einen möglichst großen Winkel zwischen $\mathbf{u}$ und $c\mathbf{p}$ anstrebend, sollte nun sichergestellt werden, daß der $\mathbf{u}$-Anteil von $\mathbf{p}$ auf die Bestimmung des Winkels möglichst nur geringen Einfluß nimmt. Daher sollte c möglichst klein gewählt werden:

$$0 < c \ll 1. \tag{4.39}$$

   Andererseits war festgestellt worden, daß die Länge des Fehlervektors minimal, der Fehler also maximal wird, wenn $||\mathbf{p}||$ die gleiche Länge wie $\mathbf{u}$ annimmt, also 1. Wir gehen jetzt weiter davon aus, daß c so gewählt wurde, daß der $\mathbf{u}$-Anteil verschwindend klein wird. Andererseits wissen wir, daß während des Lernens $||\mathbf{z}_J|| \rightarrow 1/(1-d)$ strebt (Gleichung 4.20).

   Daher sollte d so gewählt werden, daß $||cd\mathbf{z}_J||$ etwa 1 wird, also

$$0 \ll \frac{cd}{1-d} \leq 1. \tag{4.40}$$

7. **Grenzen für Parameter e**
   Da die Hauptaufgabe von e darin besteht einen Überlauf bei der Division zu verhindern, ist er so zu wählen, daß er diese Aufgabe erfüllt. Andererseits sollte er so wenig wie möglich Einfluß auf die eigentliche Normierung nehmen und ist daher sehr klein zu wählen, also

$$0 < e \ll 1 \tag{4.41}$$

8. **Grenzen für Parameter Θ**
   Dieser Rauschunterdrückungsparameter muß mit Vorsicht behandelt werden. Kleine Änderungen am Wert haben oft große Auswirkung auf die Klassifikation.

   Nach eigenen Erfahrungen kann Θ in den meisten Fällen auf einem Wert von 0 belassen werden. Die durch $\Theta > 0$ erzielten Effekte können oft auch durch Veränderung anderer Parameter erreicht werden.

   Diese Erfahrung beruht jedoch auf nur relativ wenigen Versuchen mit ART 2, von denen ein Beispiel in Abschnitt 4.5 zu finden ist. Dem Anwender bleibt es hier also weitgehend selbst überlassen, geeignete Werte zu finden.

   Grundsätzlich gelten jedoch folgende Grenzen:

$$0 \leq \Theta < 1. \tag{4.42}$$

   Da f nur Werte zwischen 0 und 1 erhält (Vektor $\mathbf{x}$ und $\mathbf{q}$ sind ja normiert), ist ein größerer Bereich für Θ nicht sinnvoll. Für $\Theta = 0$ ist f linear.

Leider ist es nicht möglich, ein „Kochrezept" für die Wahl der Parameter anzugeben. Obige Zusammenstellung soll den Einstieg in die Arbeit mit ART2-Netzen lediglich erleichtern. Sie ist ein Anhaltspunkt für eigene Experimente.

4.3 Realisierung von ART 2

Grobentwurf

Wieder stellt sich an dieser Stelle die Frage, wie die Steuerung von ART 2 erfolgen soll: im Netz oder algorithmisch? So schwer die Entscheidung fallen mag, hier fällt sie weitgehend zu Gunsten der algorithmischen Variante aus.

Wenige Überlegungen führen zu der Überzeugung, daß der Aufwand für ART2-Netze zu groß wäre, alle Steuerungs- und Synchronisationsaufgaben netzintern zu realisieren.

1. **Normierung der Vektoren**
   Betrachtet man noch einmal Abbildung 4.2, so fällt auf, daß die Norm eines Vektors im F_1-Layer erst nach zwei Propagierungszyklen an den Zielzellen zur Vefügung steht. Das direkte Signal der entsprechenden Units liegt aber bereits nach einem Zyklus an – ein Synchronisationsproblem.

   Das ist zwar nichts Neues, bedeutet aber im Fall von ART 2, daß alle direkten Signale, die auch über Normierungszellen zu einem Zielneuron gelangen, um einen Zyklus verzögert werden müßten. Bei fünf Normierungen hieße das konkret, $5N$ Zeitverzögerungszellen einführen zu müssen.

   Die Entscheidung fällt hier zu Gunsten einer Funktion aus, die einmal über die zu normierenden Units läuft, die L_2-Norm berechnet und den Wert an die Aktivierungsfunktion der entsprechenden Zielzelle liefert.

2. **F_1-Stabilitätskontrolle**
   Um netzintern kontrollieren zu können, ob die F_1-Schicht stabil ist, muß für alle F_1-Units überprüft werden, ob sich ihr Wert gegenüber dem vorangegangenen Propagierungszyklus verändert hat. Das würde bedeuten, für jede F_1-Zelle ein Duplikat anzulegen, dessen einzige Aufgabe es ist, die Aktivierung des Zeitpunkts (t-1) zu speichern und aktiv zu werden, falls der neue Aktivierungswert davon abweicht.

   Diese Signale müßten von einer weiteren Zelle aufgenommen werden, die im Stabilitätsfall eine Weiterpropagierung der $\mathbf{p}_i$-Zellen zur F_2-Schicht bzw. eine Entscheidung der Reset-Kontrolle zuließe.

   Bei sieben F_1-Units pro Eingabeneuron würden erneut $7N$ Zellen im F_1-Layer hinzukommen.

3. **Bottom-Up-, Top-Down-Phasenumschaltung**
   Wie an früherer Stelle bereits erwähnt, muß bei ART 2 streng kontrolliert werden, in welcher Phase der Propagierung sich das Netz befindet. Nur in der Bottom-Up-Phase darf von F_1 nach F_2 propagiert werden. Nur im Top-Down-Abschnitt darf die Reset-Kontrolle aktiv werden. Die Steuerung wird in die Hand einer Funktion gelegt.

4. **Reset-Kontrolle**
   Betrachtet man Gleichung 4.23, dann sieht man, daß die Reset-Kontrolle bei ART 2 aus einer Unit bestünde, die zum einen den Parameter ρ, zum anderen die Norm des Vektors $\mathbf{r}$ als Eingang benötigen würde. Obige Entscheidung, die Norm nicht im Netz, sondern durch eine Funktion berechnen zu lassen, würde dafür sorgen, daß die **rg**-Unit in ART 2 – abgesehen von einer eventuellen Unit ρ – keinerlei Eingänge hätte.

   Daher auch für die Reset-Kontrolle die Entscheidung für eine algorithmische Variante.

5. **Die Parameter**
   Bei ART 1 war im Hinblick auf ARTMAP der Parameter ρ als Unit realisiert worden. Auch ART 2 könnte zur Konstruktion von ARTMAP-Netzen herangezogen werden. Daher würde die Realisierung des Parameters ρ als Unit auch hier durchaus einen Sinn ergeben. Wie die Reset-Kontrolle wäre aber auch diese Zelle ohne Verbindung zum übrigen Netz. Deswegen wird in diesem Fall auf die Realisierung in Form einer Unit verzichtet.

   Für die übrigen Parameter a, b, c, d, e und Θ ist kein Grund vorhanden, sie netzintern darzustellen. Daher werden alle Parameter den entsprechenden ART2-Aktivierungsfunktionen über Funktionen zugänglich gemacht.

Zunächst mag es nun scheinen, als wäre überhaupt nichts mehr von einem neuronalen Netz übrig. Im folgenden wird jedoch gezeigt, daß die wesentlichen Teile, wie sie in den Abbildungen 4.1 und 4.2 dargestellt sind, erhalten bleiben, mit Ausnahme der Normierungsneuronen, aus den genannten Gründen.

In Abbildung 4.4 ist dargestellt, wie ein ART2-Netz in der vorliegenden Implementierung aufgebaut ist. Als wichtigste Neuerung gegenüber den Ausführungen in Abschnitt 4.2 ist zu bemerken, daß zur Ermittlung des Fehlervektors **r** hier der Eingabevektor **inp** bzw. **I** verwendet wird.

Das hat den Vorteil, daß die Eingabe in die Units $\mathbf{r}_i$ stabiler ist. Die $\mathbf{inp}_i$-Zellen unterliegen nicht wie die $\mathbf{u}_i$-Zellen Änderungen durch die Propagierung. Die Wahl der Parameter bleibt durch diese Änderung unbetroffen.

Da die Steuerungsaufgaben im Simulationsalgorithmus liegen, muß dieser jetzt erarbeitet werden. Dazu ist es zunächst notwendig, den genauen Ablauf einer Musterklassifikation zu betrachten. Es wird vorausgesetzt, daß das Eingabemuster in der Eingabeschicht, also in den Units $\mathbf{inp}_1 \ldots \mathbf{inp}_N$ zur Verfügung steht.

1. Der Zustand des Netzes wird wie folgt gesetzt:
   (a) Das Netz befindet sich zu Beginn in der Bottom-Up-Phase.
   (b) Das Reset-Flag wird zurückgesetzt.
   (c) Der F_1-Layer wird als instabil bewertet.
2. Es erfolgt ein Propagierungszyklus. Dabei müssen die Normen der Vektoren **inp**, **w**, **v**, **p** und **r** bereit stehen.
3. Der Gewinner in der **rec**-Schicht wird ermittelt. Einen solchen gibt es nur dann, wenn positive Aktivierungswerte in dieser Schicht existieren.
4. War F_1 im vorhergehenden Zyklus bereits als stabil bewertet worden, so konnte in die **rec**-Schicht propagiert und ein Gewinner ermittelt werden. Es erfolgt eine Umschaltung in die Top-Down-Phase, wenn sich das Netz jetzt in der Bottom-Up-Phase befindet.
5. War im vorangegangenen Zyklus ein Reset ausgelöst worden, so ist jetzt die entsprechende lokale Resetzelle $\mathbf{rst}_j$ aktiviert. Es wird in die Bottom-Up-Phase umgeschaltet und das Reset-Flag zurückgesetzt.
6. Es wird überprüft, ob F_1 stabil ist und wenn ja, ein entsprechendes Flag gesetzt.
7. Es wird überprüft, ob Vektor **r** einen Reset impliziert und ein entsprechendes Flag gesetzt, wenn sich das Netz in der Top-Down-Phase befindet.

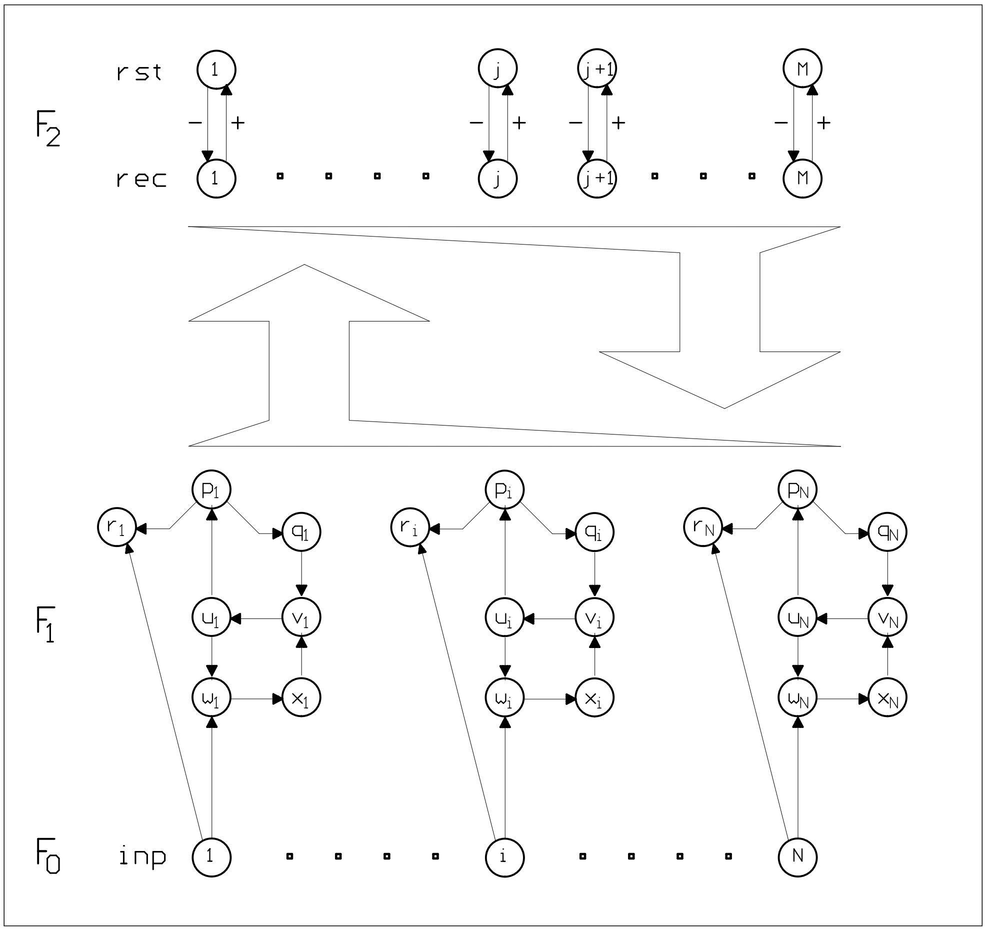


Abbildung 4.4: *Entwurf der ART2-Netzwerkstruktur. Die wichtigsten Komponenten bleiben auch bei diesem Modell in der Hand des Netzes. Steuerungsaufgaben wurden bei ART 2 algorithmisch entworfen.*

8. Wenn das Muster noch nicht klassifiziert wurde und auch noch nicht als unklassifizierbar erkannt wurde, dann wird ab Punkt 2 wiederholt.

9. Wurde das Muster klassifiziert, so kann jetzt ein Anpassen der Gewichte stattfinden.

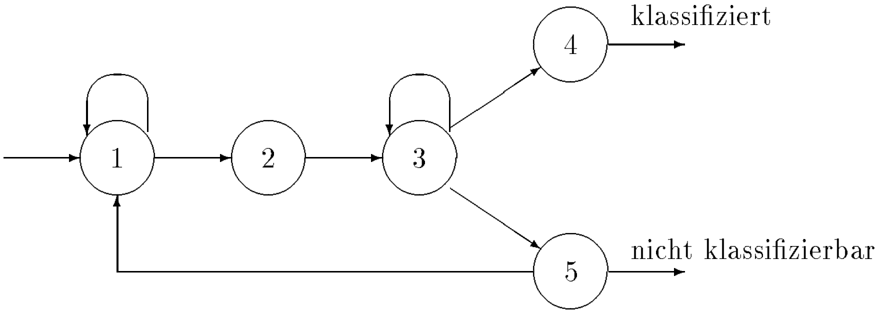


Abbildung 4.5: *Zustandsübergänge eines ART2-Netzwerks.*

Genaugenommen gibt es also fünf Zustände, die ein ART2-Netz annehmen kann:

1. Bottom-Up-Phase, F_1 instabil.
2. Bottom-Up-Phase, F_1 stabil.
3. Top-Down-Phase, F_1 instabil.
4. Top-Down-Phase, F_1 stabil, kein Reset.
5. Top-Down-Phase, F_1 stabil, Reset.

Die Zustandsübergänge können gemäß Abbildung 4.5 erfolgen.

Damit kann der eigentliche Algorithmus formuliert werden.

```
set_initial_state();
do {
     compute_vector_norms();
     save_f1_activation_values(); /* for stability check */
     propagate();
     get_winner();
     adjust_state();
     check_f1_stability();
     check_reset();
} while (( ! classified) && ( ! not_classifiable));
```

Damit ist, soviel bleibt zu hoffen, die Lücke des fehlenden ART2-Algorithmus in der Literatur geschlossen. Das Verfahren wurde so gewählt, daß es unverändert auch für alternative ART2-Architekturen eingesetzt werden kann.

Eine Ausnahme gibt es dennoch. Wird die Architektur so gewählt, wie in Abschnitt 4.2 beschrieben, wo Vektor **u** statt **inp** für die Berechnung des Fehlervektors **r** verwendet wird, dann muß in der Funktion `compute_vector_norms` natürlich die Norm von **u** berechnet werden.

Auf Kosten der Effizienz kann eine Anpassung dieser Funktion jedoch vermieden werden, indem einfach die Norm aller möglicherweise relevanten Vektoren berechnet und zur Verfügung gestellt wird, auch wenn nicht tatsächlich alle benötigt werden.

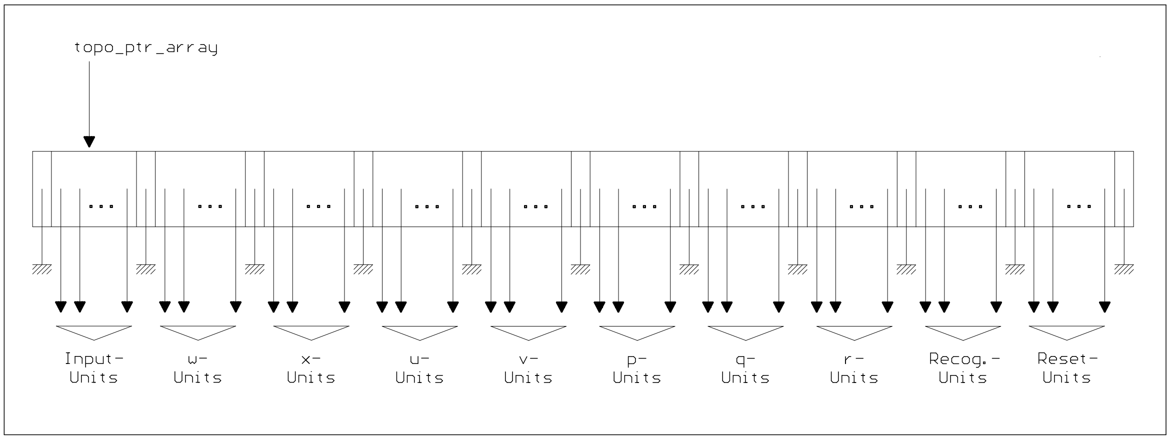


Abbildung 4.6: *Struktur des* `topo_ptr_array` *für ART2-Netzwerke.*

4.4 Implementierung und Integration von ART 2 in SNNS

Einige Besonderheiten der Implementierung werden Thema dieses Abschnitts sein. Es soll auf die Funktionen, die speziell für ART 2 entwickelt wurden, eingegangen werden. Auch die Steuerungsfunktionen werden hierbei nicht unerwähnt bleiben.

Die allgemeinen ART-Kernfunktionen, die im Modul `kr_art.c` implementiert sind, wurden in Abschnitt 3.3 kurz beschrieben. Sie werden auch in ART 2 in unveränderter Form verwendet.

Kernfunktionen für ART 2

Interessieren sollen an dieser Stelle die im Modul `kr_art2.c` implementierten, ART2-spezifischen Kernfunktionen. Diese lassen sich in zwei Gruppen einteilen. Zum einen sind es die üblichen, auch für ART 1 und ARTMAP (siehe Abschnitte 3.3 bzw. 5.4) implementierten Routinen. Zum anderen sind hier natürlich die Funktionen zur Steuerung des Netzwerks untergebracht, die in Abschnitt 4.3 erwähnt wurden.

Topologiecheck und Initialisierung

Zunächst gibt es wieder eine Funktion zur Überprüfung der Netzwerktopologie: `kra2_sort`. Sie erfüllt die gleichen Aufgaben wie die entsprechende ART 1-Routine, die in Abschnitt 3.3 beschrieben wurde.

Unter anderem wird von dieser Funktion wieder ein Feld von Zeigern angelegt, die nacheinander auf die Units in den unterschiedlichen Schichten verweisen (siehe Abbildung 4.6).

Eine Funktion `kra2_init_propagate` übernimmt die gleichen Aufgaben für ART 2, wie `kra1_init_i_act` für ART 1: Sie setzt die initialen Aktivierungswerte der verschiedenen Units. Einen anderen Namen trägt die ART2-Funktion jedoch nicht umsonst. Neben den Aktvierungen muß sie auch die festen Gewichte im Netz initialisieren.

Im Gegensatz zu ART 1 sind einige dieser Gewichte davon abhängig von Lern- bzw. Updateparametern. Die wiederum können sich mit jedem Aufruf einer der entsprechenden Funktionen ändern, womit eine Neuinitialisierung der entsprechenden Gewichte notwendig ist.

Steuerungsfunktionen

Um die Übersichtlichkeit und strukturelle Beschaffenheit von SNNS nicht zu verletzen, wurden sämtliche Steuerungsfunktionen im Modul `kr_art2.c` gekapselt. Auf diese Weise konnten die Auswirkungen durch den verminderten Netzwerkcharakter in ART 2 zumeist auf dieses Modul beschränkt werden.

1. `kra2_set_parameters`
   Da die Parameter für Aktivierungs- und Ausgabefunktionen der Units zugänglich sein müssen, werden sie in globalen Variablen des Moduls `kr_art2.c` abgelegt. Dort als `static` deklariert, sind sie nach außen hin nur über nachfolgend beschriebene Funktionen zugänglich.

   Die Werte der Parameter ρ, a, b, c, d und Θ müssen an `kra2_set_parameters` übergeben werden. Parameter e wurde als fester Wert implementiert. Er ist über das Makro `ART2_PARAM_e` zugänglich.

2. `kra2_get_rho, kra2_get_a, kra2_get_b, kra2_get_c, kra2_get_d, kra2_get_theta`
   Sie liefern als Funktionswert den jeweiligen Parameter zurück.

3. `kra2_init_pattern`
   setzt den Anfangszustand des Netzes (Bottom-Up-Phase, F_1 instabil, kein Reset).

4. `kra2_compute_norms`
   berechnet die Normen der Vektoren **inp**, **w**, **v**, **p** und **r**. Diese werden wieder in statisch globalen Variablen des Moduls `kr_art2.c` abgelegt und sind über folgende Funktion den Aktivierungsfunktionen zugänglich:

5. `kra2_L2_Norm`
   An sie ist die logische Layer-Nummer der Schicht zu übergeben, die den Vektor bildet, dessen Norm gewünscht wird. Der Rückgabewert ist die entsprechende Norm.

6. `kra2_save_for_stability_check`
   speichert die aktuellen Werte aller F_1-Units in `value_a`-Feld der Unit-Struktur.

7. `kra2_check_f1_stability`
   In dieser Funktion werden die durch `kra2_save_for_stability_check` gesicherten Werte mit den aktuellen Aktivierungen verglichen. Sind sämtliche Differenzen kleiner als ein vorgegebener Schwellwert, so wird das Netz als stabil bewertet. Die Funktion setzt das Stabilitiätsflag in diesem Fall auf `TRUE`.

8. `kra2_f1_stable`
   liefert `TRUE` zurück, wenn F_1 stabil ist, sonst `FALSE`.

9. `kra2_checkReset`
   Befindet sich das Netz in der Top-Down-Phase und ist F_1 stabil, dann prüft die Funktion, ob aufgrund des aktuellen Vektors **r** ein Reset ausgelöst wird. Ist das der Fall, so setzt sie ein entsprechendes Flag.

10. `kra2_Reset`
    liefert `TRUE` zurück, wenn das Reset-Flag gesetzt ist, anderenfalls `FALSE`.

11. `kra2_topdn_phase`
    Befindet sich das Netz in der Bottom-Up-Phase, so liefert diese Funktion `FALSE`, in der Top-Down-Phase, `TRUE` zurück.

12. `kra2_classified, kra2_not_classifiable`
    liefern die Abbruchbedingungen für die Propagierung. Ist der Rückgabewert einer der beiden Funktionen `TRUE`, so befindet sich das Netz in einem stabilen Zustand.

Die Initialisierungsfunktion

Für die Initialisierung der trainierbaren Gewichte wurde die Funktion `ART2_Weights` implementiert. Für die $F_2 \rightarrow F_1$-Gewichte erfolgt sie gemäß Gleichung 4.28. Das heißt, alle Top-Down-Gewichte werden auf 0 gesetzt.

Die Gewichte der Bottom-Up-Verbindungen genügen nach der Initialisierung der Bedingung in Gleichung 4.30. Dazu setzt man

$$z_{ij}(0) = \frac{1}{\gamma(1-d)\sqrt{N}} \qquad \forall\, 1 \leq i \leq N, 1 \leq j \leq M, \tag{4.43}$$

mit $\gamma \geq 1$. An `ART2_Weights` sind die Parameter d und γ zu übergeben. Damit Parameter d auch später, für das Training des Netzwerks zur Verfügungs steht, wird sein Wert im `bias`-Feld der Unit-Struktur gespeichert.

So steht d auch dann zur Vefügung, wenn ein bereits trainiertes Netz als SNNS-Netzwerk-Datei gesichert und später wieder geladen wird, da das `bias`-Feld Bestandteil des Unit-Definitionsteils der Datei ist.

ACHTUNG! Es sollte darauf geachtet werden, daß ART2-Netze **immer** mit der eben beschriebenen Funktion (`ART2_Weights`) initialisiert werden. Wird dazu eine andere SNNS-Initialisierungsfunktion verwendet, so ist das Verhalten des Netzes beim Lernen und Propagieren nicht vorhersagbar, da auch die festen Gewichte des Netzes mit anderen, als den für sie vorgesehenen Werten belegt werden können. Das kann dazu führen, daß die Abbruchbedingungen bei der Propagierung nie erfüllt werden können, der Simulator also in eine Endlosschleife gerät.

Die Lernfunktion

Die Lernfunktion `ART2` arbeitet so, wie es im Algorithmus in Abschnitt 4.3 dargestellt wurde. Dabei wird solange propagiert, bis ein stabiler Zustand des Netzes erreicht wurde.

Das ist zum einen dann der Fall, wenn das angelegte Eingabemuster klassifiziert werden konnte. Daraufhin werden die Gewichte gemäß Gleichungen 4.20 und 4.21 angepaßt.

Ein stabiler Zustand ist auch dann erreicht, wenn das Muster nicht klassifizierbar ist. In diesem Fall findet keine Gewichtsanpassung statt.

An `ART2` sind fünf Parameter zu übergeben: ρ, a, b, c und Θ. Parameter d wird bereits durch die Initialisierungsfunktion festgelegt und steht über das `bias`-Feld der Unit-Struktur zur Verfügung (siehe oben). Parameter e wird, wie bereits erwähnt, fest implementiert.

Die Lernfunktion gibt in dem von SNNS für Rückgabewerte vorgesehenen Feld von reellwertigen Zahlen die Nummern der Muster zurück, die klassifiziert werden konnten und diejenigen, für die eine Klassifikation nicht möglich war.

Dabei enthält das Feld – wie bei ART 1 – zunächst die Nummern der klassifizierten Muster, die Zahl -1.0 als Trennmarke und schließlich die Nummern der nicht klassifizierbaren Muster.

Die Update-Funktionen

Wieder wurden zwei verschieden Update-Funktionen implementiert:

1. `ART2_Stable` und
2. `ART2_Synchronous`.

Bedeutung und Bedienung sind der Dokumentation der entsprechendnen ART1-Funktionen zu entnehmen, die in Abschnitt 3.3 zu finden ist.

Die Übertragungsfunktionen

Da viele Steuerungsaufgaben nicht mehr in der Hand des Netzes liegen, war es für ART 2 notwendig, einigen Aktivierungs- und Ausgabefunktionen, Werte zugänglich zu machen, die in Variablen des Moduls `kr_art2.c` liegen. Diese Werte können ausschließlich über Funktionen abgefragt werden, was eine relativ hohe Ebene der Kapselung sicherstellt. Das wird sich vor allem im Falle von Änderungen positiv auswirken.

Ausgabefunktionen

Zwei Ausgabefunktionen wurden für ART 2 definiert. Es sind dies die beiden in Abschnitt 4.2 erwähnten Signalfunktionen. (Gleichungen 4.10 und 4.11). Es bleibt dem Anwender überlassen, welche der beiden Funktionen den Units $\mathbf{x}_i$ und $\mathbf{q}_i$ jeweils zugewiesen wird. In einem Netz sollte jedoch nur eine der beiden Funktionen verwendet werden. Eine Überprüfung durch `kra2_sort` findet jedoch nicht statt, um die Experimentiermöglichkeiten nicht einzuschränken.

Die beiden Funkionen sind `Out_ART2_Noise_ContDiff` (Gleichung 4.10) und `Out_ART2_Noise_PLin` (Gleichung 4.11). ContDiff steht dabei für *continously differentiable*, also stetig differenzierbar, PLin für *piecewise linear*, also abschnittsweise linear.

Aktivierungsfunktionen

1. `Act_ART2_Identity`
   Diese Funktion unterscheidet sich nur durch eine Kleinigkeit von der normalen linearen

Aktivierungsfunktion `Act_Identity`. Sie überprüft, ob ein Reset ausgelöst wurde (`kra2_Reset`) und liefert in diesem Fall den initialen Aktivierungswert zurück.

`Act_ART2_Identity` wird für die Units $\mathbf{w}_i$, $\mathbf{v}_i$ und $\mathbf{p}_i$ verwendet. Im Falle eines Resets werden sie auf ihre Initialaktivierungen zurückgesetzt.

2. `Act_ART2_NormW, Act_ART2_NormV, Act_ART2_NormP, Act_ART2_NormIP`
   Aktivierungsfunktionen der Units $\mathbf{x}_i$, $\mathbf{u}_i$, $\mathbf{q}_i$ bzw. $\mathbf{r}_i$. Die Normen der entsprechenden Vektoren $\mathbf{w}$, $\mathbf{v}$, $\mathbf{p}$ bzw. $\mathbf{inp}$ und $\mathbf{p}$ werden berücksichtigt (`kra2_L2_Norm`).

   Im Resetfall werden auch hier die initialen Aktivierungswerte der Units zurückgegeben.

3. `Act_ART2_Rec`
   Aktivierungsfunktion der **rec**-Units. Sie unterscheidet streng zwischen Top-Down- und Bottom-Up-Phase (`kra2_topdn_phase`).

4. `Act_ART2_Rst`
   Aktivierungsfunktion der **rst**-Units. Sie ermittelt, ob die zugehörige Recognition-Unit aktiv ist. Ist zusätzlich ein Reset ausgelöst worden (`kra2_Reset`), dann gibt sie einen Wert von 1.0 zurück. Das gleiche gilt dann, wenn die Unit bereits aktiviert ist.

   Anderenfalls gibt sie 0.0 zurück.

4.5 Ein Versuch mit einem ART2-Netz

Im Falle von ART 2 kann man sich zahlreiche Klassifikationsaufgaben vorstellen, da nicht mehr nur binäre, sondern reellwertige Vektoren verarbeitet werden.

Hier soll ein Beispiel herausgegriffen werden, das die Möglichkeiten von ART 2 herausstellt, zugleich aber auch auf die Schwachpunkte des Verfahrens hinweist.

Problemdefinition

Im Versuch sollen Punkte, die verrauscht um die vier Ecken eines Tetraeders herum liegen, durch ein ART2-Netz klassifiziert werden, das heißt einer Ecke zugeordnet werden. Als Eingabemuster dienen 40 dreidimensionale Vektoren. Sie repräsentieren die Punkte im Raum, die mit einem gewissen Verrauschungsgrad um die tatsächlichen Ecken herum liegen.

Aufgabe für das ART2-Netz ist es nun, vier Cluster zu erkennen. Das heißt also zum einen, die Punkte in genau vier Klassen einzuteilen, zum anderen, die jeweils richtigen Punkte in eine Kategorie einzusortieren. Dazu wurden die vierzig Punkte in verschiedenen Verrauschungsgraden um die Originalecken gelegt. Die Ergebnisse sind im folgenden aufgezeichnet.

Geringer Verrauschungsgrad

In Abbildung 4.7 ist der Tetraeder dargestellt. Die Punkte, die in der Nähe der Ecken liegen, repräsentieren die Eingabevektoren. Die maximale Entfernung der verrauschten Punkte zu den Originalecken des Tetraeders wurde hier sehr klein gewählt.

In Abbildung 4.8 ist ein Screendump des ART 2-Netzwerks gezeigt, das mit den genannten Eingabevektoren trainiert wurde.

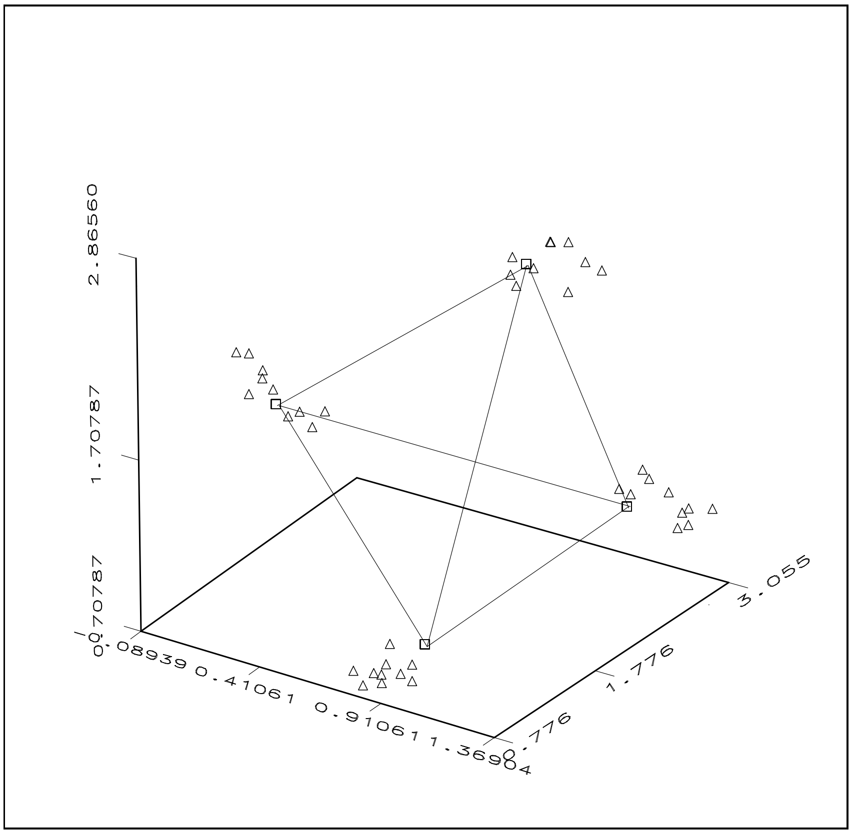


Abbildung 4.7: *Geringer Verrauschungsgrad. Um jede Ecke des Tetraeders liegen zehn Punkte, deren Ortsvektoren die Eingabemuster sind.*

Für das Training wurden folgende Parameter verwendet:

$$\rho = 0.99, a = b = 20, c = 0.1, d = 0.9, \Theta = 0, \gamma = 2$$

Das Netz klassifizierte die vierzig Eingabevektoren vollständig korrekt. Es wurden vier Klassen angelegt und die jeweils zehn zu einer Ecke gehörenden Vektoren in eine der Klassen einsortiert.

Mittlerer Verrauschungsgrad

In Abbildung 4.9 sind die maximalen Entfernungen der verrauschten Punkte um die Ecken des Tetraeders etwas größer gewählt. Zunächst wurden die Parameter wie oben belassen und das Netz über 30 Zyklen trainiert.

Dabei klassifizierte es sieben Eingabemuster falsch: fünf Muster, die in die erste Klasse gehört hätten und zwei, die der vierten Klasse zugeordnet werden müssen, wurden in Klasse drei einsortiert. Auffallend sind vor allem die fünf Klasse-Eins-Muster, die die Hälfte ihrer Kategorie ausmachen.

Auch nach 1000 weiteren Trainingszyklen konnte kein besseres Ergebnis erzielt werden. Das war jedoch auch nicht anders zu erwarten, da ART 2 ja so konzipiert ist, daß das Gelernte

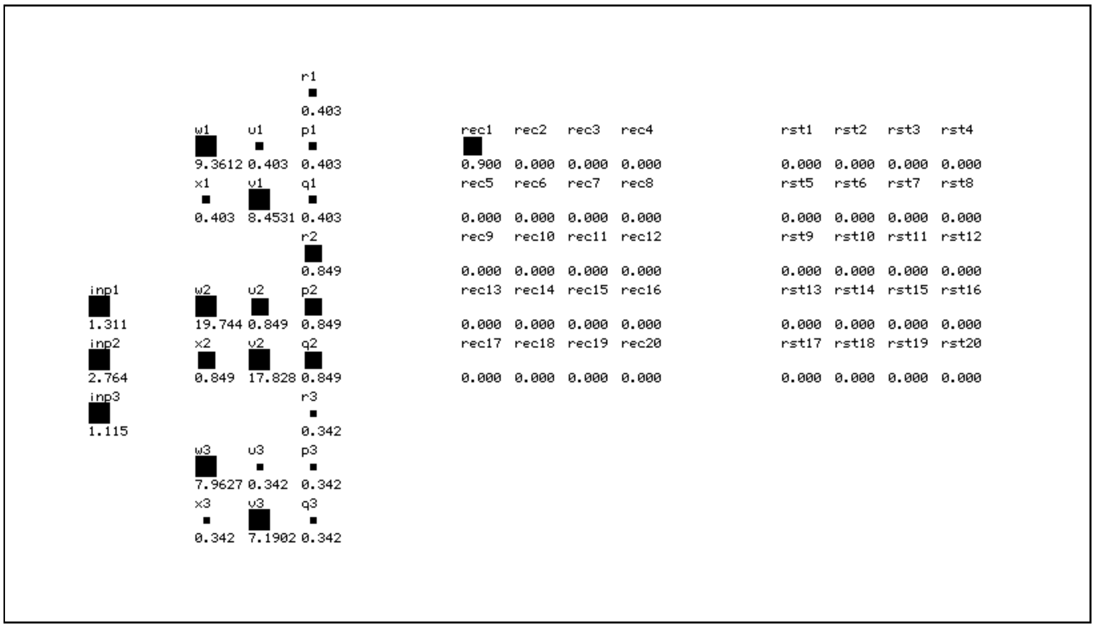


Abbildung 4.8: *Das ART2-Netz, das mit den Mustern trainiert wurde. Links sieht man die Eingabeschicht, darauffolgend die F_1-Schicht mit ihren jeweils sieben Neuronen pro Eingabezelle. Rechts befindet sich die F_2-Schicht bestehend aus **rec**-Layer und **rst**-Layer.*

nach einiger Zeit stabil ist. Folglich bringt weiteres Training nichts, nachdem sich die Gewichte stabilisiert haben. Eine solche Stabilisierung ist nach 30 bis 50 Zyklen weitgehend eingetreten.

Ein neuer Versuch brachte eine Verbesserung des Ergebnisses. Der Ähnlichkeitsparameter wurde dabei auf $\rho = 0.995$ erhöht und das Netz erneut trainiert. Der erhöhte Wert für ρ erzeugte zwar, statt vier, sechs Klassen, jedoch konnten dafür die meisten Muster jetzt korrekt klassifiziert werden.

Die erste und fünfte Klasse enthielten jetzt die Muster der ersten Ecke, Klasse zwei bzw. drei, die der zweiten bzw. dritten und Klasse vier und sechs, die der vierten Ecke. Nur zwei Muster der ersten Ecke wurden wieder fälschlich in die dritte Klasse einsortiert.

Weitere Veränderungen der Parameter ergaben keine wesentlichen Verbesserungen mehr. Es zeigte sich jeweils, daß die größten Probleme bei der Unterscheidung der ersten und dritten Ecke auftreten.

Hoher Verrauschungsgrad

In einem letzten Versuch wurden noch einmal die maximal möglichen Entfernungen der Punkte von ihren Ecken heraufgesetzt. Betrachtet man Abbildung 4.10, so ist es auch für den Menschen kaum möglich, manche Punkte eindeutig einer Ecke zuzuordnen.

Zunächst belassen wir die Parameterwerte wieder so, wie sie im letzten Versuch waren. Damit erhält man fünf Klassen. In der fünften Klasse liegen vier Punkte der vierten Ecke. In Klasse drei sind neben den zehn Punkten der dritten Ecke jetzt sechs Punkte der ersten und ein Punkt der vierten Ecke eingeordnet worden.

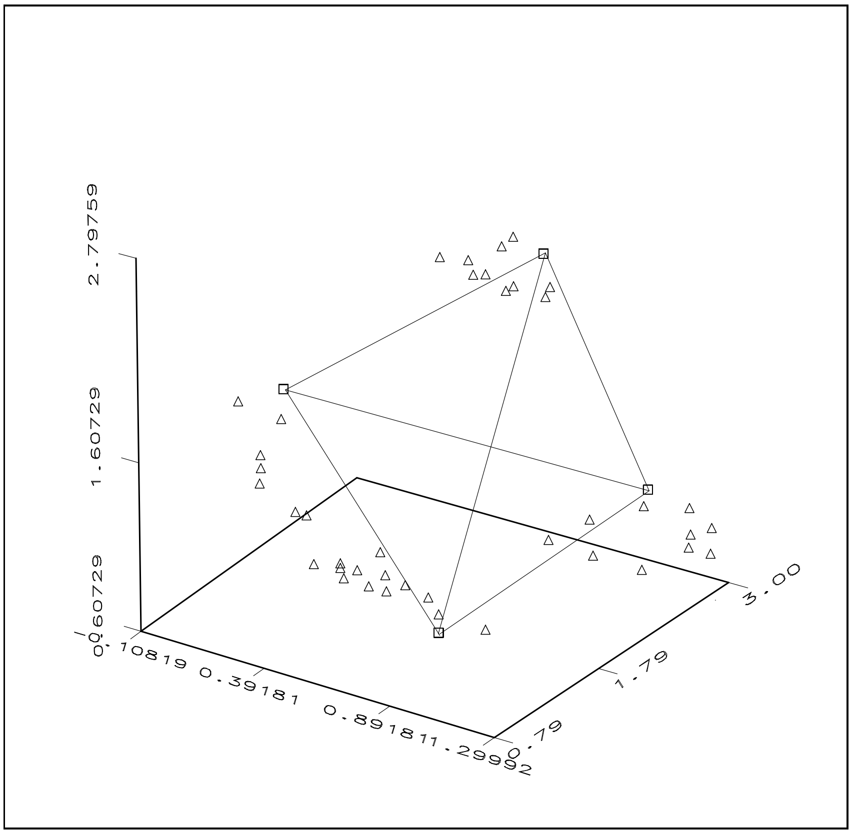


Abbildung 4.9: *Mittlerer Verrauschungsgrad. Die Punkte um die Ecken sind jetzt stärkert verrauscht.*

Noch einmal versuchen wir durch Erhöhen von ρ ein besseres Ergebnis zu bekommen. Die problematischsten Ecken eins und drei können jedoch auch jetzt nicht besser voneinander getrennt werden. Der einzige Effekt ist, daß jetzt sieben, statt fünf Klassen eröffnet werden. Weiterhin befinden sich vier Punkte der ersten Ecke in der ersten Klasse. Die vorhin in Klasse drei vermischten Punkte der ersten und dritten Ecke sind jetzt – noch immer vermischt – verteilt auf die Klassen drei, sechs und sieben.

Was also in dem Versuch mit dem mittleren Verrauschungsgrad bereits festgestellt worden war, findet man hier bestätigt. Die erste und dritte Klasse lassen sich nicht mehr voneinander trennen. Worauf ist das zurückzuführen? Betrachten wir dazu Abbildung 4.11. Sie zeigt die Punkte aus dem Versuch mit dem geringen Verrauschungsgrad als Dreiecke und die Punkte der Ecken eins und drei aus dem letzten Versuch als Quadrate. Die Perspektive wurde hier so gewählt, daß der Betrachter vom Koordinatenursprung aus auf die Punkte der ersten und dritten Ecke schaut, also in Richtung der Ortsvektoren.

Die Punkte der ersten Ecke sind durch die zufällige Verrauschung so ungünstig verteilt, daß sie etwa in der gleichen Richtung liegen, wie die der dritten Ecke. Wegen der Normierung, die bei ART 2 stattfindet, können die Punktgruppen, dann vom Netz nicht mehr getrennt werden.

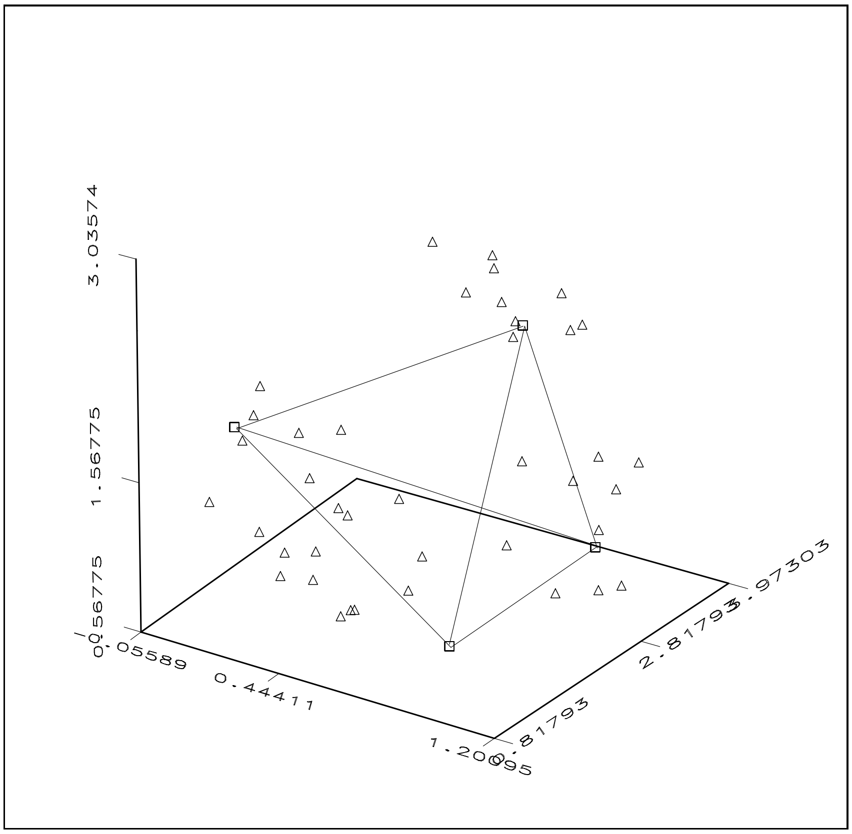


Abbildung 4.10: *Hoher Verrauschungsgrad. Die Punkte um die Ecken des Tetraeders sind jetzt so stark verrauscht, daß es auch dem Menschen schwer fällt, noch vier Cluster voneinander zu unterscheiden. Allerdings fehlt für den Betrachter auch der wirkliche dreidimensionale Eindruck.*

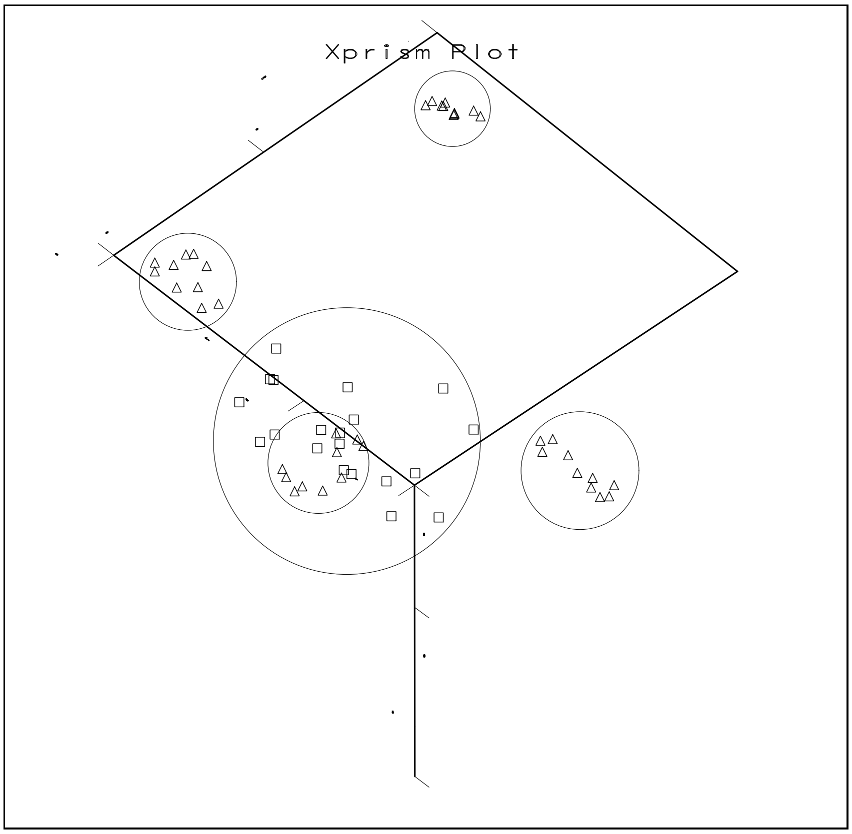


Abbildung 4.11: *Die Dreiecke sind die Punkte des Tetraeders aus dem Versuch mit dem geringen Verrauschungsgrad. Auch in dieser Perspektive lassen sich die vier Cluster deutlich voneinander unterscheiden. Anders ist es bei den als Quadrate dargestellten Punkten der ersten und dritten Ecke aus dem Versuch mit dem hohen Verrauschungsgrad. Es sind keine Cluster voneinander trennbar.*

Kapitel 5

ARTMAP

ART für binäre Eingabemuster mit überwachtem Lernen

In diesem Kapitel wird ein weiteres Modell beschrieben, das zur Klasse der ART-Architekturen gehört: ARTMAP. Wie bereits in Kapitel 2 erwähnt, handelt es sich dabei um ein Modell, das die Vorzüge dieser Architekturen bezüglich des Plastizitäts-Stabilitäts-Dilemmas mit dem Wunsch nach einem überwachten Lernverfahren verbindet.

Zunächst wird ein informeller Abschnitt die Funktionsweise von ARTMAP erläutern. Darauf aufbauend werden wir die zum Modell gehörende Mathematik untersuchen und schließlich wieder Realiesierung und Implementierung beschreiben. Ein letzter Abschnitt soll dieses Kapitel mit einem Beispiel beschließen.

5.1 Informelle Beschreibung von ARTMAP

Überwachtes Lernen

Um einen vernünftigen Einstieg in dieses Kapitel zu bekommen soll zunächst noch einmal erklärt werden, was *überwachtes Lernen* bedeutet.

Sowohl für ART 1, als auch für ART 2 gibt es lediglich eine Eingabe, die vom Netz klassifiziert wird, ohne daß von außen wesentlicher Einfluß darauf genommen werden kann. Für viele Anwendungen ist es jedoch sehr wünschenswert, dem Netz ein Eingabemuster zu geben und ihm gleichzeitig mitzuteilen, was es repräsentiert.

Nehmen wir wieder unser Buchstabenbeispiel, so bedeutet dies: man überläßt es nicht mehr komplett dem Netzwerk, Buchstaben während des Lernvorganges aufgrund ihrer Ähnlichkeit zu sortieren. Man gibt gleichzeitig mit an, daß es sich bei einer Eingabe um einen bestimmten Buchstaben handelt. In Kapitel 1 wurde dieses Beispiel in Abbildung 1.2 verdeutlicht.

Ganz so einfach, wie dort für ein *Feed-Forward-Netz* dargestellt, ist die Arbeitsweise von ARTMAP jedoch nicht. Für den Leser dürfte es aber auch keine Schwierigkeit darstellen zu verstehen, wie ARTMAP arbeitet, denn im wesentlichen besteht ein solches Netz aus zwei selbstklassifizierenden Netzen, also entweder zwei ART 1, zwei ART 2 oder gar zwei ART 3-Netzen.

In der vorliegenden Arbeit wurde ARTMAP in der ersten Form, also aufbauend auf ART 1

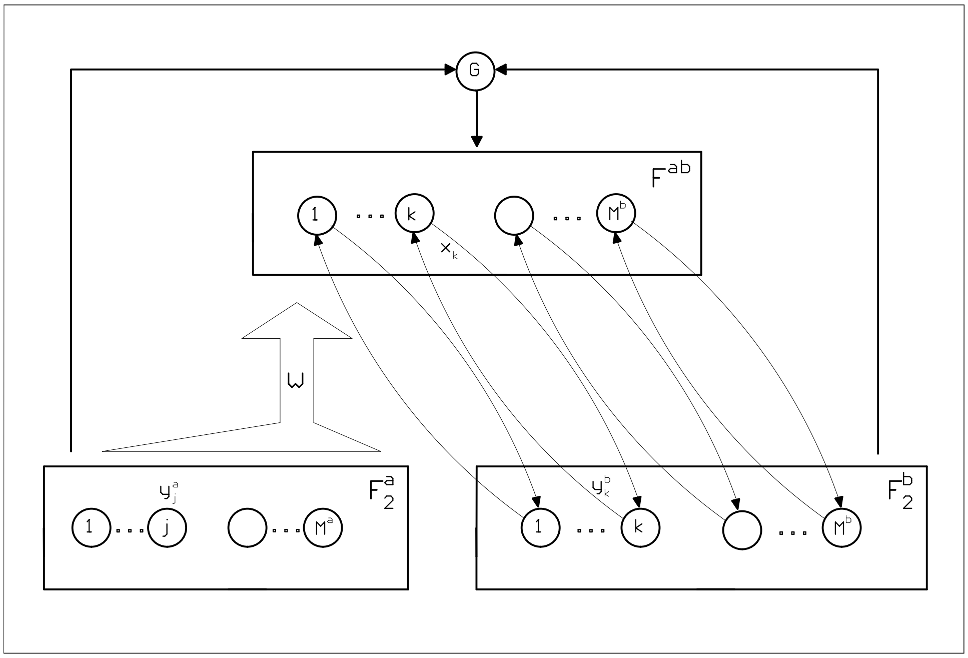


Abbildung 5.1: *Ausschnitt eines ARTMAP-Netzwerkes. Dargestellt sind lediglich F_2^a-, F_2^b-, und F^{ab}-Layer.*

realisiert. Daher wird in der Beschreibung immer von zwei Netzwerken dieses Modells ausgegangen um die Funktionen zu erläutern. Für das Verständnis ist es jedoch von untergeordneter Bedeutung, welches der Modelle letzten Endes verwendet wird.

Eines der beiden ART1-Netze stellt die Eingabeseite dar: an seinem Input-Layer wird das eigentliche Eingabemuster angelegt. Wir nennen dieses Teilnetz ART^a. Am Input-Layer des zweiten kompletten ART1-Netzes (ART^b) erfolgt die Lerneingabe. Hier wird angegeben, wie das Eingabemuster von ART^a zu klassifizieren ist.

Um diese Informationen miteinander zu verknüpfen, bedarf es eines Verbindungsstücks zwischen den beiden ART1-Netzen. Diese Aufgabe erfüllt das *MAP-Field*. Wir werden diesen Layer auch mit F^{ab} bezeichnen. Er verbindet die internen Klasseninformationen von ART^a mit denjenigen von ART^b.

Eine unbenutzte Klasse in ART^a

Zunächst gehen wir von einem Eingabemuster im Input-Layer von ART^a aus. Dieses Teilnetz arbeitet genau so, wie wir es in Kapitel 3 gesehen haben. Das heißt, es propagiert seine Eingabe solange, bis in seiner F$_2$-Schicht eine Klasse feststeht. Um zunächst den einfachsten Fall zu behandeln, soll es sich dabei um eine noch nicht belegte Klasse handeln.

Die F$_2$-Schicht des ART^a-Netzwerkes (= F$_2^a$) ist vollständig mit dem MAP-Field verbunden, wie es in Abbildung 5.1 dargestellt ist. Die Gewichte dieser Verbindungen sind zu Beginn alle mit 1.0 initialisiert. Eine noch nicht belegte Klasse in F$_2^a$ wird also sämtliche F^{ab}-Units sensibilisieren.

Der MAP-Layer arbeitet ganz ähnlich wie die Comparison-Schicht. Bekommt er lediglich eine

Eingabe von ART^a, so ist seine Gain-Unit (**G**) aktiv und nach der 2/3-Regel, die hier erneut zu finden ist, werden alle Units in dieser Schicht aktiviert. Wenn alle Units im MAP-Field aktiv sind, besagt dies nichts anderes, als daß das Netz „keine Ahnung" hat, worum es sich bei der Eingabe handeln könnte.

Nehmen wir jetzt ART^b hinzu. Wir legen hier im Input-Layer ebenfalls ein Eingabemuster an, das – für uns auf irgendeine Weise bedeutungsvoll – die Klasse repräsentiert, in der wir das Eingabemuster von ART^a eingeordnet haben möchten.

Auch hier wird sich nun nach bekannter ART1-Manier im F_2^b-Layer eine ART^b-interne Klasse ausbilden. Die Gewinnerzelle gibt nun ihre Information an die ihr entsprechende Unit im MAP-Field weiter. Damit wurde hier bereits implizit erklärt, daß die Zahl der Units in F^{ab} genau der Zahl der Units in der F_2^b-Schicht entspricht und daß die k-te Unit des F_2^b-Layers eine positive Verbindung zur k-ten Unit des MAP-Fields hat.

Die Gain-Unit des MAP-Fields hat inzwischen den Wert 0 angenommen, da jetzt nicht nur eine Eingabe von F_2^a anliegt, sondern auch von ART^b eine positive Aktivierung vermerkt wird.

Nach der 2/3-Regel bleibt nur noch eine Unit im F^{ab}-Layer aktiv, nämlich genau diejenige, die sowohl von ART^a, als auch von ART^b einen positiven Eingang bekommt. Über die Verbindung, die von der so im MAP-Layer aktivierten Unit zu der ihr entsprechenden Zelle im F_2^b-Layer zurückgeht, wird die Wahl bestätigt.

Das Netz hat jetzt einen stabilen Zustand erreicht und die Gewichte können trainiert werden. Das bedeutet im Fall von ARTMAP, daß die $F_1^a \rightarrow F_2^a$, die $F_2^a \rightarrow F_1^a$-, die $F_1^b \rightarrow F_2^b$, die $F_2^b \rightarrow F_1^b$- und schließlich die $F_2^a \rightarrow F^{ab}$-Verbindungen angepaßt werden müssen.

Für die erstgenannten Verbindungen ist die Methode zur Anpassung der Gewichte Kapitel 3 zu entnehmen. Neu sind hier lediglich die Verbindungen zwischen F_2^a- und F^{ab}-Schicht. Die Methode hier entspricht derjenigen, die für das Training der Gewichte zwischen F_2- und F_1-Schicht im ART1-Netzwerk angewandt wird.

So bleibt lediglich der Wert der Verbindung zwischen ART^a-Gewinnerzelle und der einzigen noch aktiven MAP-Unit auf 1. Alle anderen Verbindungen, die von der Gewinnerzelle zum MAP-Field gehen, werden dagegen auf 0 gesetzt. Dieses Vorgehen bewirkt, daß bei einem erneuten Anlegen des gleichen Eingabemusters im Input-Layer von ART^a, diese MAP-Unit aktiviert wird. Alle anderen Units in der F^{ab}-Schicht verbleiben jetzt jedoch auch ohne eine Entscheidungshilfe von ART^b inaktiv.

Eine bereits benutzte Klasse gewinnt in ART^a

Nehmen wir für die folgenden Ausführungen an, ein angelegtes Eingabemuster in ART^a bringe als Gewinner eine bereits belegte Klasse in der F_2^a-Schicht hervor und die Reset-Kontrolle von ART^a entscheide, daß das angelegte Eingabemuster dem bisher gelernten ähnlich genug ist, um tatsächlich in diese Klasse eingeordnet zu werden.

Die F_2^a-Gewinnerzelle gibt, wie oben, ihre Information an F^{ab} weiter. Hier wird nur noch die für sie trainierte MAP-Unit aktiviert. An dieser Stelle sind – eine Eingabe in ART^b vorausgesetzt – zwei Fälle zu unterscheiden: erstens, die ART^b-Gewinnerzelle bestätigt die Wahl von ART^a, das heißt, es wird die Unit in F_2^b aktiv, die der aktiven F^{ab}-Unit entspricht. Zweitens, die Wahl von ART^a wird durch ART^b verworfen, das heißt eine andere F_2^b-Unit wird aktiv.

Im ersten Fall ist bereits das Ende der Propagierung erreicht. Das Netz hat wieder einen stabilen Zustand erreicht und kann wie oben beschrieben trainiert werden.

Für den zweiten Fall muß zunächst geklärt werden, was das Nichtbestätigen durch ART^b eigentlich bedeutet. Nun, es besagt einfach, daß ART^a eine falsche Klasse als Gewinner hervorgebracht hat, denn die Eingabe in ART^b kontrolliert ja die Arbeit von ART^a.

Also muß in diesem Fall dafür gesorgt werden, daß ART^a – obwohl in sich bereits stabil – seine Wahl verwirft und nach einer neuen Klasse sucht. Um das zu erreichen muß ART^a davon „überzeugt werden", einen Reset auszulösen. Dazu wird der Parameter ρ^a soweit erhöht, daß das Muster im Comparison-Layer dem im Input-Layer gerade eben nicht mehr ähnlich genug ist, um durch den neuen ρ^a-Wert akzeptiert zu werden.

Das Erhöhen von ρ^a wird durch den sogenannten *Inter-ART-Reset* ausgelöst, der dann auftritt, wenn in F_2^b eine Unit aktiv ist, aber keine MAP-Unit aktiviert werden kann, weil von F_2^a eine andere F^{ab}-Unit angesprochen wird.

Die Folge ist die Fortsetzung der Suche nach einer Klasse in ART^a, die sowohl der jetzt verschärften Ähnlichkeitsbedingung genügt, als auch die richtige, von ART^b geforderte MAP-Unit aktiviert.

Test eines Musters

Es verbleibt schließlich noch zu untersuchen, was in einem ARTMAP-Netz abläuft, wenn ein Eingabemuster getestet werden soll.

Dazu legt man einen Eingabevektor am Input-Layer von ART^a an, beläßt aber alle Units in der Input-Schicht von ART^b auf 0. In ART^a wird daraufhin eine passende Klasse gesucht, die dem anzugebenden Ähnlichkeitsparameter ρ^a genügt. Die gefundene Klasse wird dann, falls belegt, die für sie trainierte MAP-Unit aktivieren und so eine für uns relevante Aussage liefern.

Ist die ART^a-Gewinnerklasse jedoch noch unbenutzt, so werden alle Units in F^{ab} aktiv – es stehen ja noch alle von ihr ausgehenden Gewichte zum MAP-Field auf 1. Für den Benutzer ist dies der Hinweis, daß das Netz nicht in der Lage ist, aufgrund seines bisherigen Wissens eine Aussage über die Klassifikation des angelegten Musters zu machen.

Es soll hier deutlich hervorgehoben werden, daß die Ausgabe eines ARTMAP-Netzwerks nicht dort erfolgt, wo die Lerneingabe am Netz angelegt wird. Dies unterscheidet das überwachte Lernen bei ARTMAP erheblich von demjenigen bei „normalen" Feed-Forward-Netzen. Da bei ARTMAP auch die Lerneingabe kodiert wird und – je nach Wahl von ρ^b – mehrere Lerneingabemuster auf die gleiche ART^b-Klasse abgebildet werden können, hat das Verfahren eine andere Qualität, als andere überwachte Lernalgorithmen.

Die eigentliche Ausgabe des ARTMAP-Netzwerks erfolgt im MAP-Field und ist somit eine kodierte Version der eigentlichen Lerneingabe.

5.2 ARTMAP Theorie

Nachfolgend wollen wir die das Modell beschreibende Mathematik betrachten[1]. Dabei werden die theoretischen Grundlagen für ART 1, die in Abschnitt 3.1 behandelt wurden, vorausgesetzt. Wir werden uns zum Teil auf die dort angegebenen Gleichungen beziehen.

Die 2/3-Regel für das MAP-Field

Im MAP-Field des ARTMAP-Netzwerkes finden wir die 2/3-Regel wieder. Jede Unit des F^{ab}-Layers bekommt Eingänge von drei Seiten: von F_2^a, F_2^b und der MAP-Field-Gain-Kontrolle. Damit sieht die Regel für den Ausgabevektor $\mathbf{x}$ von F^{ab} wie folgt aus:

$$x_k = \begin{cases} 1 & \text{wenn } y_k^b + \mathbf{G} + \sum_{j=1}^{M^a} y_j^a w_{jk} > 1 + \overline{w}, \\ 0 & \text{sonst,} \end{cases} \tag{5.1}$$

wobei y_k^b die k-te Komponente des F_2^b-Ausgabevektors ist, $\mathbf{G}$ das binäre MAP-Field-Gain-Signal, und $\sum y_j^a w_{jk}$ ist die Summe der $\mathrm{F}_2^a \rightarrow \mathrm{F}^{ab}$-Signale über die gewichteten Verbindungen w_{jk}. $\overline{w}$ ist eine Konstante mit

$$0 < \overline{w} < 1. \tag{5.2}$$

Gain-Kontrolle für den $\mathbf{F}^{ab}$-Layer

Beim Vergleich der Gleichungen 3.1 und 5.1 bemerkt man eine Analogie zwischen der Layer-Gruppe F_2^b, F^{ab}, F_2^b im ARTMAP-Netz auf der einen Seite und der Gruppe F_0, F_1, F_2 im ART1-Netz auf der anderen.

Dennoch sind natürlich Unterschiede in der Verbindungsstruktur gegeben, die verschiedene Regeln für die Gain-Kontroll-Zellen $\mathbf{G}$ und $\mathbf{g}_1$ erfordern. Die MAP-Field-Gain-Kontrolle folgt der Gleichung

$$\mathbf{G} = \begin{cases} 0 & \text{wenn sowohl } \mathrm{F}_2^a \text{ als auch } \mathrm{F}_2^b \text{ aktiv sind,} \\ 1 & \text{sonst.} \end{cases} \tag{5.3}$$

Initialisierung der $\mathbf{F}_2^a \rightarrow \mathbf{F}^{ab}$ - Gewichte

Eine noch unbelegte ART^a-Klasse muß alle F^{ab}-Units sensibilisieren, damit eine belibige Kategorie mit ihr assoziiert werden kann. Dabei wählt die aktive F_2^b-Unit die entsprechende MAP-Kategorie aus.

Deshalb werden die Gewichte der $\mathrm{F}_2^a \rightarrow \mathrm{F}^{ab}$-Verbindungen analog zu denen der $\mathrm{F}_2 \rightarrow \mathrm{F}_1$-Gewichte im ART1-Netz (Gleichung 3.12) initialisiert:

[1] Die mathematischen Betrachtungen dieses Abschnitts basieren vor allem auf [Carpenter91b].

$$w_{jk}(0) = 1 \qquad \forall\, 1 \le j \le M^a, 1 \le k \le M^b. \tag{5.4}$$

Aktivierung im MAP-Field

Bei der Arbeit eines ARTMAP-Netzes sind vier grundsätzliche Fälle zu unterscheiden: wenn sowohl ART^a als auch ART^b aktiv sind, so kann ein Training der Gewichte zwischen F_2^a und F_{ab} stattfinden. Ist nur ART^a aktiv, so wird eine zuvor gelernte $\mathrm{ART}^a \to \mathrm{ART}^b$ Assoziation aus F^{ab} ausgelesen. Wenn nur ART^b aktiv ist, dann wird die gewählte F_2^b-Kategorie in F^{ab} reprästentiert. Schließlich ist F^{ab} nicht aktiv, wenn weder ART^a noch ART^baktiv ist.

Mit den Gleichungen 5.1, ff. realisiert die 2/3-Regel diese Eigenschaften in allen vier Fällen:

$\mathbf{F}_2^a$ aktiv und $\mathbf{F}_2^b$ aktiv

Ist sowohl ein F_2^a-Gewinnerknoten J als auch ein F_2^b-Gewinnerknoten K aktiv, dann ist gemäß Gleichung 5.3 $\mathbf{G} = 0$. Damit reduziert sich Gleichung 5.1 auf

$$x_k = \begin{cases} 1 & \text{wenn } k = K \text{ und } w_{JK} > \overline{w}, \\ 0 & \text{sonst.} \end{cases} \tag{5.5}$$

Es sind also alle $x_k = 0$ für $k \neq K$ und $x_K = 1$ nur für den Fall, daß zuvor eine Assoziation in der Verbindung von Unit J zu Unit K glernt wurde bzw. Unit J zum ersten Mal belegt wird. Ist mit Unit J eine andere als Kategorie K assoziiert, so sind alle $x_k = 0$.

$\mathbf{F}_2^a$ aktiv und $\mathbf{F}_2^b$ inaktiv

Ist der F_2^a-Knoten J aktiv und F_2^b inaktiv, dann ist $\mathbf{G}$=1 und Gleichung 5.1 sieht in diesem Fall so aus:

$$x_k = \begin{cases} 1 & \text{wenn } w_{JK} > \overline{w} \\ 0 & \text{sont.} \end{cases} \tag{5.6}$$

Mit den Gleichungen 5.4 und 5.6 aktiviert J alle F^{ab}-Units, wenn für Unit J in F_2^a noch nichts gelernt wurde. Dagegen ist für den Fall, daß es für J bereits eine Assoziation gibt nur noch die Unit K in F^{ab} aktiv, zu der die Verbindung von J aus auf einem Wert von 1 verblieben ist.

$\mathbf{F}_2^a$ inaktiv und $\mathbf{F}_2^b$ aktiv

Ist der F_2^b-Knoten K aktiv und F_2^a inaktiv, dann ist $\mathbf{G}$ ebenfalls gleich 1. Gleichung 5.1 reduziert sich dann auf

$$x_k = \begin{cases} 1 & \text{wenn } k = K \\ 0 & \text{sonst.} \end{cases} \tag{5.7}$$

In diesem Fall entspricht der Ausgabevektor $\mathbf{x}$ von F^{ab} genau dem F_2^b-Ausgabevektor $\mathbf{y}^b$.

$\mathbf{F}_2^a$ inaktiv und $\mathbf{F}_2^b$ inaktiv

Ist weder F_2^a noch F_2^b aktiv, dann ist der Netzeingang für jede F^{ab}-Unit beschränkt auf $\mathbf{G} = 1$ und somit nach Gleichung 5.1 alle $x_k = 0$.

Lernregeln für die $\mathbf{F}_2^a \rightarrow \mathbf{F}^{ab}$ - Verbindungen

Die Lernregel für die $F_2^a \rightarrow F^{ab}$ - Verbindungen ist derjenigen für die $F_2 \rightarrow F_1$ - Verbindungen im ART1-Netz sehr ähnlich:

$$\frac{d}{dt} w_{jk} = y_j^a (x_k - w_{jk}). \tag{5.8}$$

Demnach nimmt der Gewichtsvektor $\mathbf{w}_J$ die Werte des Aktivitätsvektors $\mathbf{x}$ an, wenn die J-te F_2^a-Unit aktiv ist. Sonst bleibt $\mathbf{w}_J$ konstant. Wenn für Knoten J noch keine Assoziation gelernt wurde, so haben alle Gewiche w_{Jk} den Wert 1. Kommt in diesem Fall keine Eingabe von ART^b in das MAP-Field, so findet keine Gewichtsanpassung statt.

MAP-Field-Reset und Match-Tracking

Das MAP-Field erlaubt es, sehr ähnliche ART^a-Eingabemuster in unterschiedliche Kategorien einzuordnen, wenn ART^b darauf „besteht". Gleichermaßen bietet es auch die Möglichkeit sehr unterschiedliche Eingabeverktoren von ART^a, die dort in verschiedene interne Klassen eingeteilt werden, auf die gleiche globale Kategorie im MAP-Field abzubilden.

Dazu gibt es neben den Orientierungkontrollen in ART^a und ART^b einen dritten, globalen Resetmechanismus. Er wird nur dann ausgelöst, wenn ART^a aufgrund bereits gelernter Muster eine Kategorie in F^{ab} wählt, die durch ART^b nicht bestätigt wird. Dieses Ereignis tritt dann ein, wenn ein Knoten in F_2^b aktiv ist, jedoch keine Aktivierung einer F^{ab}-Unit zur Folge hat.

Der sogenannte Inter-ART-Reset wird also unter folgender Bedingung ausgelöst:

$$|\mathbf{x}| < \rho |\mathbf{y}^b|. \tag{5.9}$$

Dabei ist ρ der globale MAP-Field-Ähnlichkeitsparameter (siehe dazu auch Abbildung 5.2). Die Folge eines Inter-ART-Reset ist das Erhöhen des ART^a-Ähnlichkeitsparameters, so daß gewissermaßen ein ART^a-Reset erzwungen wird. Zu Beginn der Präsentation eines Eingabemusters ist dabei ρ^a stets auf einem anzugebenden Minimalwert $\overline{\rho^a}$.

Wenn ein Eingabevekor $\mathbf{I}^a$ eine F_2^a-Unit J aktiviert und das ART^a-Teilnetz in einen stabilen Zustand gelangt, dann gilt:

$$|\mathbf{x}^a| = |\mathbf{I}^a \cap \mathbf{z}_J^a| \geq \rho^a |\mathbf{I}^a|, \tag{5.10}$$

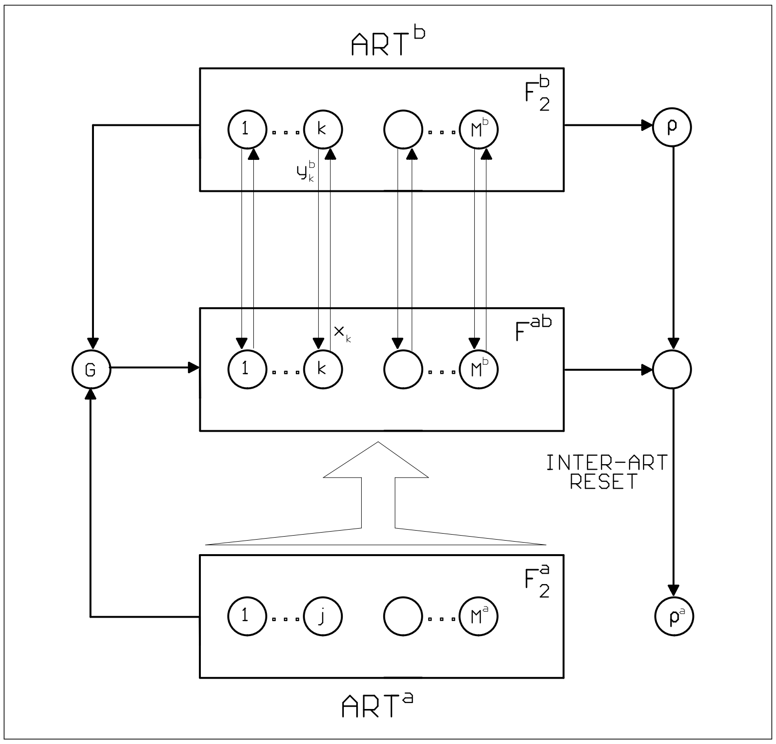


Abbildung 5.2: *Ausschnitt eines ARTMAP-Netzwerkes. Dargestellt sind F_2^a-, F_2^b-, und F^{ab}-Layer. F_2^b und F^{ab} entscheiden über den Inter-ART-Reset.*

Ein Inter-ART-Reset-Signal wird gemäß Gleichung 5.9 an ART^a geschickt, wenn die durch $\mathbf{x}^a$ sensibilisierte F^{ab}-Kategorie nicht mit der Klasse in ART^b übereinstimmt. Dieses Signal R erhöht ρ^a gerade so weit, daß Ungleichung 5.10 fehlschlägt. Dann gilt:

$$\rho^a > \frac{|\mathbf{I}^a \cap \mathbf{z}_J^a|}{|\mathbf{I}^a|}. \tag{5.11}$$

Den Vorgang des Erhöhens von ρ^a nennt man *Match-Tracking.* Die Folge ist ein Reset in ART^a, der die Weitersuche nach einer neuen Klasse bewirkt und den Knoten J in F$_2^a$ blockiert.

Man mag sich nun fragen, wozu diese aufwendige Mathematik dient. Theoretisch würde es doch genügen, mit dem Inter-ART-Reset-Signal in ART^a einen neuen Suchzyklus zu veranlassen, ohne dabei ρ^a zu erhöhen. Das heißt, das Signal würde einfach vorgeben, ein ART^a-internes Reset-Signal zu sein und damit ebenfalls einen neuen Suchzyklus auslösen. Diese Variante bezeichnet man als *MAP-Field-Reset.*

Diese Lösung wäre sicherlich einfacher zu realisieren, führt jedoch nicht zum gewünschten Ergebnis. Betrachten wir die in Tabelle 5.1 aufgeführten Eingabemuster für ART^a und die jeweils zugehörigen Vektoren für ART^b. Nehmen wir ferner an, wir hätten kein Match-Tracking, sondern lediglich ein MAP-Field-Reset, wie im vorigen Absatz beschrieben.

Wir wählen $\overline{\rho^a} \leq 0.6$ und $\rho^b > 0$. Dem Netz wird zunächst das Vektorpaar $(\mathbf{I}_1^a, \mathbf{I}_1^b)$ präsentiert, was eine Aktivierung der ART^a- und ART^b-Klassen $J = 1$ bzw. $K = 1$ zur Folge hat. Damit ist die Assoziation für das erste Vektorpaar gelernt.

Als nächstes lassen wir das Netz das zweite Vektorpaar $(\mathbf{I}_2^a, \mathbf{I}_2^b)$ verarbeiten. $\mathbf{I}_2^a$ wird dabei die

ART^a-Eingabevektoren	ART^b-Eingabevektoren
$\mathbf{I}_1^a$ (1 1 1 0 0 0)	$\mathbf{I}_1^b$ (1 0 1 0)
$\mathbf{I}_2^a$ (1 1 1 1 0 0)	$\mathbf{I}_2^b$ (0 1 0 1)
$\mathbf{I}_3^a$ (1 1 1 1 1 0)	$\mathbf{I}_3^b$ (1 0 1 0)

Tabelle 5.1: *Die ART^a-Eingabevektoren sind so gewählt, daß jeweils der nächste eine Obermenge des vorherigen ist. Rechts in der Tabelle die jeweils zu assoziierenden ART^b-Eingabevektoren. Beispiel aus [CG91b].*

Klasse $J = 1$ aktivieren, ohne, daß ein ART^a-Reset ausgelöst wird, denn

$$\frac{|\mathbf{I}_2^a \cap \mathbf{z}_1^a|}{|\mathbf{I}_2^a|} = \frac{3}{4} \geq \rho^a = \overline{\rho^a}. \tag{5.12}$$

$J = 1$ sensibilisiert $K = 1$ im MAP-Field, wie es bei der Präsentation des ersten Musters gelernt wurde. Eingabevektor $\mathbf{I}_2^b$ bringt jedoch einen anderen Gewinner in F_2^b hervor, nämlich $K = 2$. Dadurch wird in der MAP-Field-Kontrolle ein Reset ausgelöst, und in ART^a beginnt ein neuer Suchzyklus.

Die erfolgreiche Suche führt zu $J = 2$, einer noch unbelegten Klasse, die sofort das gewünschte Ergebnis liefert und – auch ohne Match-Tracking – dafür sorgt, daß $J = 2$ mit $K = 2$ assoziiert wird.

Schließlich präsentieren wir dem Netz noch das letzte Vektorpaar $(\mathbf{I}_3^a, \mathbf{I}_3^b)$. In ART^a wird zunächst Klasse $J = 2$ aktiv. In ART^b haben wir dagegen wieder $K = 1$. Mit $J = 2$ wird jedoch $K = 2$ assoziiert, was wiederum zu einem MAP-Field-Reset führt. Die weitere Suche in ART^a führt auf $J = 1$, das immer noch keinen ART^a-Reset auslöst, da

$$\frac{|\mathbf{I}_3^a \cap \mathbf{z}_1^a|}{|\mathbf{I}_3^a|} = \frac{3}{5} \geq \rho^a = \overline{\rho^a}. \tag{5.13}$$

Weil mit $J = 1$ die richtige Kategorie $K = 1$ verbunden ist, bricht an dieser Stelle die Suche ab und es wird nichts gelernt. Das heißt aber, daß bei einer Präsentation von $\mathbf{I}_3^a$ ohne den zugehörigen ART^b-Vektor $\mathbf{I}_3^b$ die falsche Kategorie aktiviert würde, der gleiche Fehler also immer wieder auftreten würde.

Verwendet man jedoch beim Anlegen des dritten Musterpaares Match-Tracking, so sieht der Ablauf aus, wie folgt: Zunächst wird in ART^a wieder die Klasse $J = 2$ aktiviert. Sie paßt nicht zu $K = 1$ in ART^b und verursacht einen Inter-ART-Reset. Mit Match-Tracking wird dadurch ρ^a nun gemäß Gleichung 5.11 erhöht:

$$\rho^a = \frac{|\mathbf{I}_3^a \cap \mathbf{z}_2^a|}{|\mathbf{I}_3^a|} + \varepsilon = \frac{4}{5} + \varepsilon = 0.8 + \varepsilon > \overline{\rho^a}. \tag{5.14}$$

Dies zieht einen ART^a-Reset nach sich und als nächstes wird $J = 1$ in F_2^a aktiviert. Jetzt ist jedoch die Übereinstimmung zwischen dem für $J = 1$ ursprünglich gelernten Muster und Eingabevektor $\mathbf{I}_3^a$ nicht mehr genügend groß:

$$\frac{|\mathbf{I}_3^a \cap \mathbf{z}_1^a|}{|\mathbf{I}_3^a|} = \frac{3}{5} = 0.6 < 0.8 + \varepsilon = \rho^a \tag{5.15}$$

Der interne ART^a-Reset bringt im nächsten Suchzyklus $J = 3$ als Gewinner hervor und assoziiert diese Klasse dann ebenfalls mit $K = 1$. Danach wird mit jedem ART^a-Eingabevektor direkt die richtige Kategorie verbunden.

Probleme mit Match-Tracking

Die Idee des Match-Tracking birgt jedoch auch Gefahren. In einigen Fällen könnte das Verfahren dazu führen, daß durch einen Inter-ART-Reset ρ^a größer wird als 1, und damit in ART^a überhaupt keine Klasse gefunden werden kann. Welche Fälle sind das?

Die Probleme treten dann auf, wenn ein neues Eingabemuster Untermenge eines breits gelernten ist. Betrachten wir zur Erklärung wieder ein Beispiel, das uns bereits aus Abschnitt 3.4 bekannt ist.

Wir nehmen ein ARTMAP-Netz, dessen ART^a-Eingabe-Schicht 35 Units umfaßt. Sie können die bekannten Buchstabenmuster aus o.g. Abschnitt aufnehmen. Die ART^b-Eingabeschicht enthält 26 Units, von der jede für einen Buchstaben steht: die erste für 'A', die zweite für 'B' usw.

Wir trainieren nun den Buchstaben 'B' im initialisierten Netz mit den Parametern $\overline{\rho^a} = 0.5$, $\rho^b = 1.0$, $\rho = 1.0$. Dabei wird die erste ART^a-Klasse mit diesem Muster belegt. Sie wird alle F^{ab}-Units sensibilisieren, da das Netz gemäß Voraussetzung initialisiert war, bevor das Eingabmuster 'B' angelegt wurde.

In der Eingabeschicht von ART^b ist die zweite Unit aktiv (sie steht für 'B'), alle anderen sind inaktiv. Auch hier wird die erste ART^b-Klasse als Gewinner hervorgehen und so wird das Muster 'B' auf die erste MAP-Field-Kategorie abgebildet. (Siehe Abbildung 5.3)

Trainieren wir nun als nächstes den Buchstaben 'P', so erhalten wir als Zwischenzustand ein Bild, wie es in Abbildung 5.4 dargestellt ist.

Das Eingabemuster in ART^a aktiviert direkt die gleiche F$_2^a$-Unit, die zuvor für den Buchstaben 'B' gelernt wurde. Das passiert, weil der Eingabevektor für 'P' eine Untermenge dessen für 'B' ist. Aus dem gleichen Grund stimmt das Durchschnittsmuster in F$_1^a$ auch zu 100% mit dem Eingabevektor in F$_0^a$ überein.

Die aktive F$_2^a$ Unit $J = 1$ sensibilisiert nun die zuvor gelernte Kategorie $K = 1$ in F^{ab}. In ART^b liegt jetzt natürlich der Eingabevektor für 'P' an. Er bringt, weil nicht ähnlich genug, eine neue Klasse $K = 2$ in F$_2^b$ als Gewinner hervor. ART^a und ART^b widersprechen sich, und in F^{ab} bleiben alle Units inaktiv.

Der dadurch ausgelöste Match-Tracking-Zyklus erhöht dann ρ^a soweit, daß in ART^a ein Reset ausgelöst werden kann. Gemäß Ungleichung 5.11 bedeutet das:

$$\rho^a_{\text{neu}} = \frac{|\mathbf{I}^a \cap \mathbf{z}_J^a|}{|\mathbf{I}^a|} + \varepsilon. \tag{5.16}$$

Da die Muster in F$_1^a$ und F$_0^a$ aber übereinstimmen ist

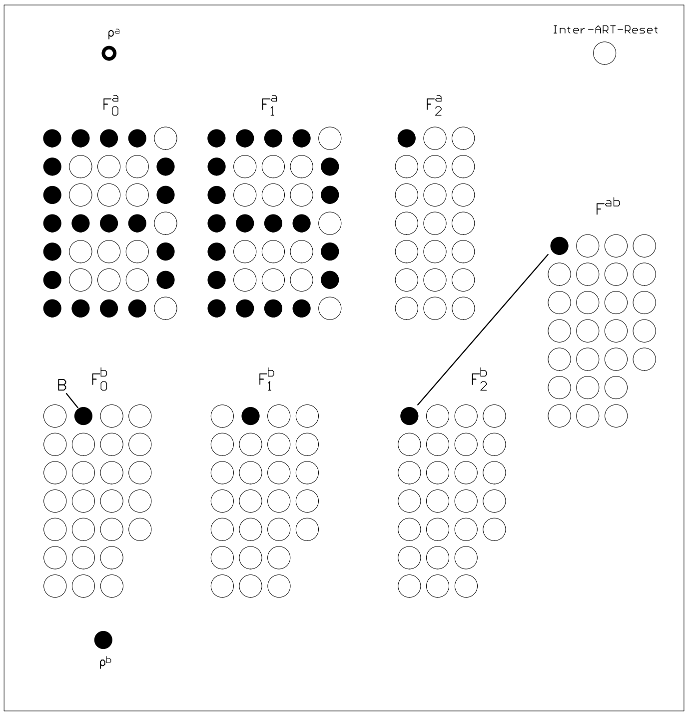


Abbildung 5.3: *Propagierung des Buchstaben 'B' im ARTMAP-Netz bei* $\overline{\rho^a}$ *= 0.5,* ρ^b *=1.0 und* ρ *=1.0. Die aktive Klasse in* F_2^a *sensibilisiert alle* F^{ab}*-Units. Durch* ART^b *wird nur die erste ausgewählt und aktiviert. Nach dem Anpassen der Gewichte wird die erste Klasse des* ART^a*-Teilnetzes mit der ersten MAP-Field-Kategorie assoziiert.*

$$\frac{|\mathbf{I}^a \cap \mathbf{z}_J^a|}{|\mathbf{I}^a|} = 1 \tag{5.17}$$

und folglich

$$\rho^a_{\mathbf{neu}} = 1 + \varepsilon. \tag{5.18}$$

Zwar ist damit der gewünschte Effekt eingetreten: in ART^a wird ein Reset ausgelöst. Jedoch kann in F$_2^a$ natürlich keine Klasse gefunden werden, die die Forderung von $\rho^a > 1$ erfüllen kann.

Complement Coding

Der Ausweg aus diesem Dilemma ist nicht schwer zu finden. Mit einem einfachen Trick kann das Problem umgangen werden. Dazu ist es allerdings erforderlich, die Zahl der Eingabneuronen in ART^a zu verdoppeln. Natürlich wächst die Zahl der Units in F$_1^a$ entsprechend mit.

Statt nur das tatsächliche ART^a-Eingabemuster zu verwenden, wird an die neuen Units nun zusätzlich sein Komplement angelegt. Das heißt, die eine Hälfte des neuen Eingabevektors sieht aus, wie bisher. In der zweiten Hälfte werden jeweils genau die Units aktiviert, die in der ersten inaktiv sind.

Ein Muster

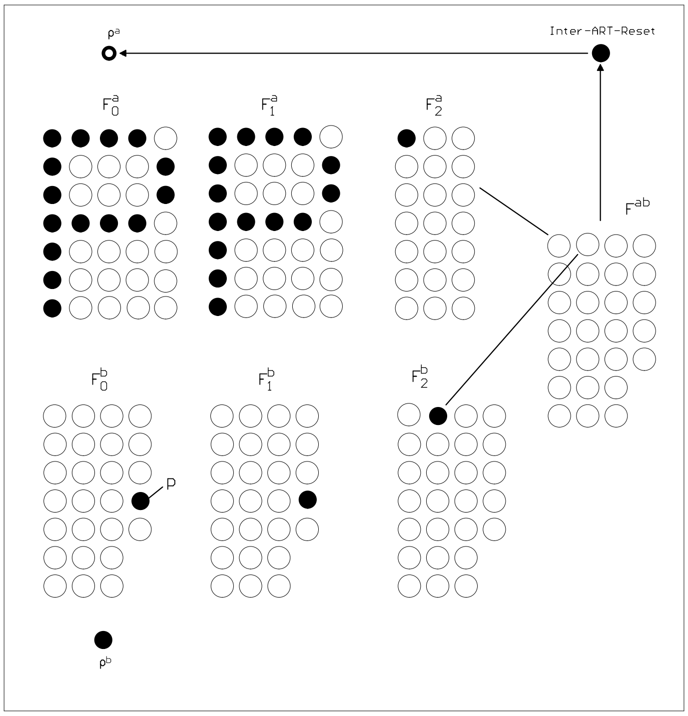


Abbildung 5.4: *Propagierung des Buchstaben 'P' im ARTMAP-Netz bei* $\overline{\rho^a}$ *= 0.5,* ρ^b *=1.0 und* ρ *=1.0. 'P' erzeugt die gleiche Gewinnerklasse wie 'B', da das Eingabemuster eine Untermenge des anderen ist. In* ART^b *wird aber eine andere Kategorie gefordert. In* F^{ab} *ist keine Unit aktiv. Die Folge ist ein Match-Tracking-Zyklus.*

$$\mathbf{I}^a = (1\ 1\ 0\ 0\ 1)$$

sähe in der neuen Version dann so aus:

$$\mathbf{I}^a_{\text{neu}} = (\mathbf{I}^a, (\mathbf{I}^a)^c) = (1\ 1\ 0\ 0\ 1, 0\ 0\ 1\ 1\ 0).$$

Mit diesem Verfahren erreicht man gleich zwei Ziele. Zum einen ist das Untermengenproblem gelöst, was mit folgendem Beispiel veranschaulicht werden soll:

$\mathbf{I}^a$	$\mathbf{I}^a_{\text{neu}}$
(1 1 1 0 0 0)	(1 1 1 0 0 0, 0 0 0 1 1 1)
(1 1 1 1 0 0)	(1 1 1 1 0 0, 0 0 0 0 1 1)

In der linken Spalte ist deutlich erkennbar, daß das erste Muster Untermenge des zweiten ist. Rechts ist dieses Problem durch *Complement Coding* umgangen.

Der zweite Vorteil ist, daß immer genau die Hälfte aller Eingabeneuronen aktiviert ist. Jede Klasse in F_2^a wird also stets durch die gleiche Anzahl von aktiven Units in F_1^a angesprochen. Dadurch sind sich die Beträge der Verbindungsgewichte zwischen F_1^a- und F_2^a-Layer sehr ähnlich.

Warum ist das aber ein Vorteil? Stellen wir uns folgendes Beispiel vor. Wir haben ein ART1-Netz mit sechs Eingabeneuronen, das folgende Muster lernen soll:

Nr.	Muster
1	(1 0 0 0 0 0)
2	(0 0 1 1 1 1)
3	(1 0 0 1 1 0)

Wir wählen $\rho = 0.5$, $\beta = 1.0$. Muster $\mathbf{I}_1$ wird in der ersten Klasse von F_2 klassifiziert. Dabei sind die Gewichte nach dem Lernen gemäß Gleichung 3.18 wie folgt verteilt:

$$\begin{aligned} Z_{11} &= \frac{1}{\beta + |\mathbf{x}|} = 0.5, \\ Z_{i1} &= \frac{0}{\beta + |\mathbf{x}|} = 0.0, \quad \forall\, i \neq 1. \end{aligned}$$

Trainieren wir das zweite Muster $\mathbf{I}_2$, das in die zweite Klasse eingeordnet wird, dann gilt:

$$\begin{aligned} Z_{12} = Z_{22} &= \frac{0}{\beta + |\mathbf{x}|} = 0, \\ Z_{32} = Z_{42} = Z_{52} = Z_{62} &= \frac{1}{\beta + |\mathbf{x}|} = 0.2. \end{aligned}$$

Betrachtet man jetzt Muster $\mathbf{I}_3$, so erwartet man natürlich, daß es in die zweite Klasse einsortiert wird, denn es ist Muster $\mathbf{I}_2$ mit $\rho = 0.5$ ähnlich genug. Statt dessen wird aber zunächst Klasse 1 gewinnen. Die erste Komponente von $\mathbf{I}_3$ bewirkt nämlich, daß die erste F_2-Unit mit 0.5 aktiviert wird. Die zweite Unit in F_2 bekommt dagegen nur $2 \cdot 0.2 = 0.4$ als Netzeingang von den anderen beiden aktiven Eingabezellen.

Natürlich ist das aus der ersten Klasse zurückpropagierte Muster $\mathbf{I}_3$ nicht ähnlich genug, denn

$$\frac{|\mathbf{I}_3 \cap \mathbf{z}_1|}{|\mathbf{I}_3|} = 0.\overline{3} < 0.5 = \rho.$$

Erst der zweite Suchzyklus bringt die zweite Klasse als Gewinner hervor, in die $\mathbf{I}_3$ dann auch eingeordnet wird.

Mit Complement Coding wäre uns der erste Suchzyklus erspart geblieben. Da die Zahl der aktiven Units im ersten, wie im zweiten Muster hier gleich gewesen wäre, hätte kein Gewicht eine so exponierte Stellung eingenommen, wie oben Z_{11}. Bereits beim Vorwärtspropagieren wäre dann tatsächlich die Ähnlichkeit der Eingabevektoren $\mathbf{I}_3$ und $\mathbf{I}_2$ für die Wahl der Klasse ausschlaggebend gewesen.

Ein Nachteil des Verfahrens liegt jedoch insbesondere bei der Simulation mit seriellen Rechnern auf der Hand: Die Zahl der notwendigen Multiplikationen verdoppelt sich durch den vergrößerten F_1-Layer.

Für Anwendungen, bei denen ohnehin sichergestellt ist, daß kein Eingabemuster Untermenge eines anderen sein kann, und die Zahl der aktivierten Units für jedes Eingabemuster in etwa gleich ist, benötigt man Complement Coding natürlich nicht.

5.3 Realisierung von ARTMAP

In diesem Abschnitt wird die Topologie der ART1-Netze aus Kapitel 3 vorausgesetzt. Hier sollen nur die neu hinzugekommenen Units und Verbindungen des MAP-Fields ausführlich diskutiert und deren Realisierung beschrieben werden.

Die Philosophie

Die in Abschnitt 3.2 getroffenen Entscheidungen prägen natürlich auch den Entwurf des ARTMAP-Netzes. Schließlich sollen die beiden ART1-Netze, die die Basis für ARTMAP bilden, strukturell so bleiben wie sie dort definiert wurden.

Damit ist auch für ARTMAP festgelegt, daß die gesamte Mathematik nicht algorithmisch, sondern im Netz realisiert wird. Wir werden neben der Ermittlung der jeweiligen Gewinnerzelle in F_2^a und F_2^b keine weiteren algorithmischen Lösungen benötigen.

Das MAP-Field

Die F^{ab}-Schicht wird, wie es in Abschnitt 5.2 beschrieben ist, zwischen ART^a und ART^b „eingehängt". Sie besitzt M^b Units, also genauso viele, wie der F_2^b-Layer. Wir werden diese Units mit $(\mathbf{map}_1, \ldots, \mathbf{map}_{M^b})$ bezeichnen.

Jede Unit $\mathbf{map}_k$ $(1 \leq k \leq M^b)$ erhält eingehende Verbindungen von drei verschiedenen Quellen:

1. von allen Units $\mathbf{del}_i^a$, $\forall\, 1 \leq i \leq M^a$,
2. von der Unit $\mathbf{del}_k^b$ und
3. von der MAP-Field-Gain-Unit $\mathbf{G}$ (siehe unten).

Diese Struktur war bereits in den Abbildungen 5.1 und 5.2 dargestellt worden. Gleichermaßen führt von jeder Unit $\mathbf{map}_k$ eine Verbindung zurück zu $\mathbf{del}_k^b$.

Die Aktivierung der MAP-Field-Units berechnet sich nach der 2/3-Regel. Bekommt $\mathbf{map}_k$ von zwei ihrer drei Quellen positive Eingangssignale, so schaltet sie auf 1, anderenfalls ist sie 0.

Abbildung 5.5 zeigt die Struktur des MAP-Fields einschließlich der im folgenden beschriebenen Kontrollzellen.

Die Gain-Kontrolle

Die Gain-Kontrolle für das MAP-Field wird wieder durch eine Unit realisiert. Sie unterscheidet sich jedoch von der $\mathbf{g}_1$-Unit im ART1-Netz durch eine etwas andere Aktivierungsfunktion. Hier darf sie gemäß Gleichung 5.3 nur dann einen Wert von 0 annehmen, wenn sowohl ART^a, als auch ART^b aktiv sind.

Dazu werden alle Units $\mathbf{rec}_i^a$ $(1 \leq i \leq M^a)$ und alle $\mathbf{rec}_k^b$ $(1 \leq k \leq M^b)$ mit der $\mathbf{G}$-Unit verbunden. Bekommt sie von jeder Seite einen positiven Eingang, so schaltet die Gain-Kontrolle ab. In allen anderen Fällen ist ihre Aktivierung 1.

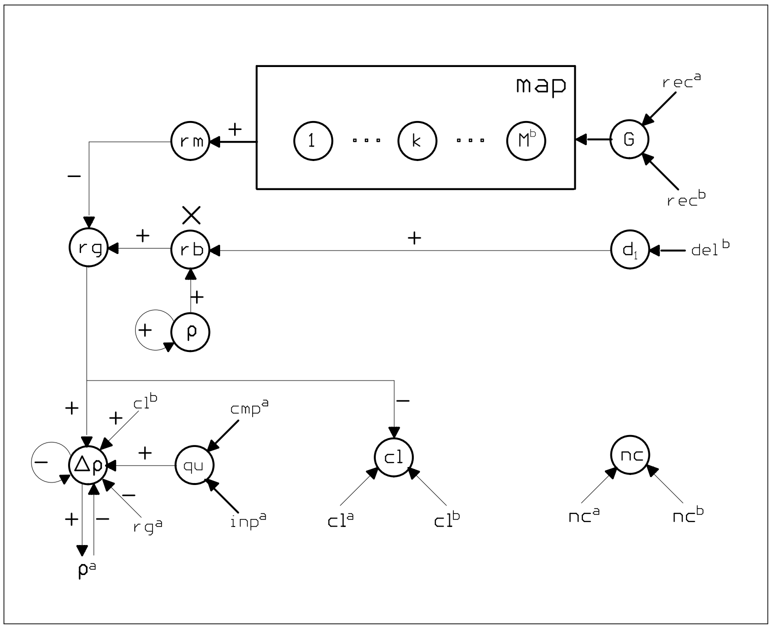


Abbildung 5.5: *Das MAP-Field mit seinen Kontroll- und Steuerzellen.*

Die **G**-Unit sensiblisiert alle MAP-Field-Units, wenn nur von ART^a oder nur von ART^b Eingangssignale kommen. Damit kann ein ART^a-Muster getestet werden. Außerdem kann für ein ART^b-Muster überprüft werden, durch welche MAP-Field-Kategorie es repräsentiert wird.

Die Verbindungen von **G** kommen, wie eben beschrieben, von den Recognition-Units in ART^a bzw. ART^b. Der Grund, warum sie nicht von den $\mathbf{del}^a$- bzw. $\mathbf{del}^b$-Units zu **G** geführt werden ist synchrnonisationstechnischer Natur. Auf diese Weise bekommt nämlich die Gain-Unit genau einen Zyklus vor den MAP-Field-Units ihre Information aus ART^a und ART^b. Dadurch liegt ihr Ausgangssignal zeitgleich mit den entsprechenden Informationen aus den Delay-Schichten am MAP-Field an. Ein weiteres Mal bewährt sich hier die Zweiteilung der F_2-Schicht.

Die Reset-Kontrolle

Die Entscheidung darüber, ob ein Inter-ART-Reset ausgelöst wird oder nicht, wird durch ein ähnliches Konstrukt realisiert, wie es bei ART1-Netzen anzutreffen ist. Die globale Reset-Kontrolle besteht aus den fünf Units: **rm**, ρ, **rb**, **rg** und $\mathbf{d}_1$.

Mit diesen fünf Zellen wird Ungleichung 5.9 realisiert. Zur Erinnerung:

$$|\mathbf{x}| < \rho|\mathbf{y}^b|.$$

Wir stellen sie für unsere Zwecke um und erhalten

$$|\mathbf{x}| - \rho|\mathbf{y}^b| < 0. \tag{5.19}$$

Die rm-Unit

Um den ersten Teil der linken Seite – $|\mathbf{x}|$ – zu berechnen, führen wir Unit **rm** ein. Sie summiert einfach die Zahl der aktiven **map**-Units auf. Dazu bekommt sie positiv gewichtete Eingänge von allen Zellen im MAP-Field.

Die ρ-Unit

Die ρ-Unit hat keine andere Aufgabe, als den Wert für ρ zu repräsentieren. Diesen Wert liefert sie ständig an **rb**. Über eine Verbindung zu sich selbst reaktiviert sie sich mit jedem Propagierungszyklus.

Die rb-Unit

Unit **rb** summiert die Zahl der aktiven $\mathbf{del}^b$-Zellen auf, berechnet also $\mathbf{y}^b$. Weiter multipliziert sie den so erhaltenen Wert mit ρ. **rb** realisiert damit den zweiten Teil der linken Seite von Ungleichung 5.19.

Die rg-Unit

Schließlich müssen die beiden Teile noch zusammengeführt werden. Unit **rg** erhält zu diesem Zweck einen positiven Eingang von **rm** und einen negativen von **rb**. So berechnet sie die Differenz der linken Seite in Ungleichung 5.19.

rg nimmt eine Aktivierung von 1 an, falls die Differenz kleiner als 0 ist, bleibt dagegen auf 0, wenn diese Bedingung nicht erfüllt ist.

Die $\mathbf{d}_1$-Unit

Bisher unerwähnt blieb die $\mathbf{d}_1$-Unit. Sie ist, wie der Name bereits erkennen läßt, wieder eine Verzögerungseinheit. Die Signale, die von $\mathbf{del}^b$ über das MAP-Field zu **rm** gelangen, benötigen für diesen Weg zwei Propagierungszyklen.

Die **rb**-Unit würde ihre Information, bekäme sie sie direkt von den $\mathbf{del}^b$-Units, dann einen Zyklus zu früh erhalten. Eine Fehlentscheidung wäre die Folge. Deshalb legen wir zwischen den $\mathbf{del}^b$-Layer und **rb** die Unit $\mathbf{d}_1$. Sie ermittelt über positive Eingangssignale von allen $\mathbf{del}_k^b$ die Zahl der aktiven Units in dieser Schicht.

Über ein positives Verbindungsgewicht gibt sie ihre Aktivierung weiter an **rg**, der die gleiche Information dann genau einen Zyklus später zur Verfügung steht.

Der Match-Tracking-Mechanismus

In Abschnitt 5.2 wurde ausführlich darüber gesprochen, daß aufgrund des Inter-ART-Reset-Signals ein Match-Tracking-Zyklus ausgelöst werden soll.

Folgende Bedingungen für die Inkrementierung von ρ^a müssen erfüllt sein:

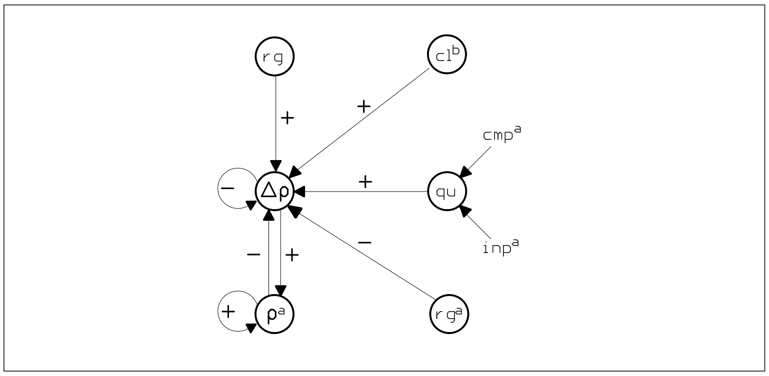


Abbildung 5.6: *Realisierung des Match-Tracking-Mechanismus. Unit* ***qu*** *berechnet den Quotienten* $(|\mathbf{I}^a \cap \mathbf{z}_J^a|)/(|\mathbf{I}^a|)$*. Dazu erhält sie Eingänge von* $\mathbf{cmp}^a$ *und* $\mathbf{inp}^a$*. Unit* $\Delta\rho$ *berechnet* $\Delta\rho$ *gemäß obiger Gleichung, wenn von* ***rg*** *und* $\mathbf{cl}^b$ *jeweils ein positives und von* $\mathbf{rg}^a$ *kein Signal ansteht.*

- Es muß ein Inter-ART-Reset ausgelöst worden sein, das heißt, **rg** ist aktiv.
- In ART^b muß bereits eine Klasse feststehen. Wäre das nicht der Fall, so könnte im $\mathbf{del}^b$-Layer gerade eine Unit aktiv sein, die zwar nicht zu einer Aktivierung im MAP-Field führt, aber noch gar nicht die endgültige Klasse in $\mathbf{del}^b$ ist und somit auch nicht zu einer Erhöhung von ρ^a führen darf: $\mathbf{cl}^b$ aktiv
- Es darf kein interner ART^a-Reset anstehen, der eine Erhöhung unnötig macht: $\mathbf{rg}^a$ inaktiv.

Ist die dritte Bedingungen erfüllt, dann ist bekannt

$$\rho^a \leq \frac{|\mathbf{I}^a \cap \mathbf{z}_J^a|}{|\mathbf{I}^a|}. \tag{5.20}$$

Ziel ist gemäß Gleichung 5.11:

$$\rho^a_{\text{neu}} > \frac{|\mathbf{I}^a \cap \mathbf{z}_J^a|}{|\mathbf{I}^a|}.$$

Anders formuliert:

$$\rho^a_{\text{neu}} = \rho^a + \Delta\rho = \frac{|\mathbf{I}^a \cap \mathbf{z}_J^a|}{|\mathbf{I}^a|} + \varepsilon \tag{5.21}$$

und damit

$$\Delta\rho = \frac{|\mathbf{I}^a \cap \mathbf{z}_J^a|}{|\mathbf{I}^a|} - \rho^a + \varepsilon. \tag{5.22}$$

Gleichung 5.22 soll mit Hilfe zweier Units unter Beachtung der genannten Bedingungen realisiert werden. Betrachten wir dazu Abbildung 5.6.

Unit **qu** bekommt Eingänge von den $\mathbf{cmp}^a$- und den $\mathbf{inp}^a$-Units. Sie berechnet den Quotienten $(|\mathbf{I}^a \cap \mathbf{z}_J^a|)/(|\mathbf{I}^a|)$. Diesen Wert liefert sie an $\Delta\rho$ weiter.

Unit $\Delta\rho$ soll gemäß Gleichung 5.22 arbeiten. Wie Abbildung 5.6 entnommen werden kann, erhält sie dazu einen positiven Eingang von **qu** und einen negativen von ρ^a. Der Wert für ε ist innerhalb der Aktivierungsfunktion für $\Delta\rho$ festgelgt.

Die anderen Eingänge von **rg** , $\mathbf{rg}^a$ und $\mathbf{cl}^b$ sind notwendig, damit $\Delta\rho$ entscheiden kann, ob die drei oben genannten Bedingungen erfüllt sind. Ist das der Fall, so soll Unit $\Delta\rho$ den Wert für $\Delta\rho$ an Unit ρ^a weitergeben.

Die Aktivierungsfunktion sieht dann wie folgt aus:

$$\mathrm{act}_{\Delta\rho} = \begin{cases} \mathbf{qu} - \rho^a + \varepsilon + \mathbf{rg} - 1 + \mathbf{cl}^b - 1 - \mathbf{rg}^a & \text{wenn } \mathrm{net}_{\Delta\rho} > 2, \\ 0 & \text{sonst,} \end{cases} \tag{5.23}$$

mit

$$\mathrm{net}_{\Delta\rho} = \mathbf{qu} - \rho^a + \mathbf{rg} + \mathbf{cl}^b - \mathbf{rg}^a.$$

Diese Funktion erfüllt genau die genannten Forderungen. Die Bedingungen für eine Aktivierung von $\Delta\rho$ sind durch die Eingänge von **rg** , $\mathbf{rg}^a$ und $\mathbf{cl}^b$ realisiert. Um die Unit $\Delta\rho$ möglichst einfach zu gestalten, werden diese Eingänge einfach zu denjenigen hinzuaddiert, die eigentlich für die Berechnung von $\Delta\rho$ verantwortlich sind: **qu**, ρ^a.

Für den Fall, daß $\mathbf{rg}^a$ aktiv ist, hemmt sie $\Delta\rho$ soweit, daß $\mathrm{net}_{\Delta\rho}$ nicht größer als 2 sein kann. Wenn **rg** aktiv ist, $\mathbf{rg}^a$ aber nicht, dann muß natürlich um $\Delta\rho$ zu berechnen der im Netzeingang durch **rg** addierte Wert wieder subtrahiert werden. Deshalb -1 hinter **rg** in Gleichung 5.23. Außerdem muß dann auch $\mathbf{cl}^b$ noch aktiv sein und deren Wert ebenfalls zur Berechnung von $\Delta\rho$ wieder subtrahiert werden. Daher auch -1 hinter $\mathbf{cl}^b$ in Gleichung 5.23.

Schließlich bleibt noch die Verbindung von $\Delta\rho$ zu sich selbst zu erklären: wurde festgestellt, daß ρ^a erhöht werden muß, so nimmt $\Delta\rho$ einen positiven Wert an. Im nächsten Zyklus wird dieser zu ρ^a hinzuaddiert, wie Abbildung 5.6 entnommen werden kann.

Die Erhöhung von ρ^a wirkt sich jedoch erst einige Propagierungszyklen später auf die Aktivierung im Comparison Layer aus, so daß Unit $\Delta\rho$ weiterhin einen positiven Wert behielte. Sie bekommt nämlich immer noch die gleichen Eingangswerte. Die Folge wäre, daß das bereits erhöhte ρ^a nochmals erhöht würde.

Um dies zu verhindern hemmt sich Unit $\Delta\rho$ einfach selbst. Einmal aktiv, gibt sie im nächsten Zyklus nicht nur ihren Wert über eine positiv gewichtete Verbindung an Unit ρ^a weiter. Darüberhinaus sorgt sie über eine negative Verbindung zusätzlich dafür, daß sie selbst wieder „abgeschaltet" wird.

Einen weiteren Propagierungszyklus später hat dann ρ^a bereits den erhöhten Wert angenommen. Dieser Wert geht jetzt anstatt des alten Wertes in die Berechnung von $\mathrm{net}_{\Delta\rho}$ ein. Dadurch ist $\mathrm{net}_{\Delta\rho} < 2$ und Unit $\Delta\rho$ bleibt inaktiv, bis ein tatsächlich neuer Match-Tracking-Impuls von **rg** kommt.

Gleichung 5.23 ändert sich also wie folgt:

$$\mathrm{act}_{\Delta\rho} = \begin{cases} \mathbf{qu} - \rho^a + \varepsilon + \mathbf{rg} - 1 + \mathbf{cl}^b - 1 - \mathbf{rg}^a - \Delta\rho & \text{wenn } \mathrm{net}_{\Delta\rho} > 2, \\ 0 & \text{sonst,} \end{cases} \tag{5.24}$$

mit

$$\mathrm{net}_{\Delta\rho} = \mathbf{qu} - \rho^a + \mathbf{rg} + \mathbf{cl}^b - 1 - \mathbf{rg}^a - \Delta\rho,$$

wobei es sich bei $\Delta\rho$ auf der rechten Seite jeweils um den alten Wert handelt, den die Unit einen Propagierungszyklus zuvor angenommen hatte.

Abbruchkriterien

Auch für ARTMAP muß ein Kriterium geschaffen werden, das die Entscheidung darüber ermöglicht, ob weiter propagiert werden muß, das heißt ob das Netz einen stabilen Zustand erreicht hat. Wie bereits bei ART 1 stehen hier wieder zwei Abbruchkriterien bereit.

Die cl-Unit

Ist ein Eingabemuster in ART^a und in ART^b klassifiziert worden dann ist der Zustand des Netzwerks stabil, sofern im MAP-Field kein Reset ausgelöst wurde. Dann nämlich ist eine Assoziation zwischen beiden Teilen der Eingabe gefunden worden und das Propagieren kann abgebrochen werden.

Die **cl**-Unit wird aktiv, wenn dieses Kriterium erfüllt ist. Dazu erhält sie drei Eingänge: je einen positiv gewichteten von $\mathbf{cl}^a$ bzw. $\mathbf{cl}^b$ und einen negativ gewichteten von **rg**. Ist der Netzeingang von $\mathbf{cl} \geq 2$, dann nimmt sie einen positiven Aktivierungszustand an.

Die nc-Unit

Kann der ART^a-Teil des Eingabemusters von ART^a oder der ART^b-Teil von ART^b nicht mehr klassifiziert werden, weil keine passende Klasse gefunden wird, dann ist das komplette Muster im ARTMAP-Netz nicht klassifizierbar.

Die **nc**-Unit erhält daher je einen positiven Eingang von $\mathbf{nc}^a$ und $\mathbf{nc}^b$. Ist ihr Netzeingang ≥ 1, dann wird sie aktiv und teilt so nach außen mit, daß das Eingabemuster nicht klassifizierbar ist.

5.4 Implementierung und Integration von ARTMAP in SNNS

Wie für die beiden zuvor beschriebenen ART-Modellen soll an dieser Stelle auch für ARTMAP kurz auf die Besonderheiten bei der Implementierung eingegangen werden. Die allgemeinen Kernfunktionen, die für alle ART-Modelle verwendet werden können, sind in Abschnitt 3.3 kurz erwähnt worden.

Kernfunktionen für ARTMAP

Im Modul `kr_amap.c` sind die speziellen Kernfunktionen für ARTMAP implementiert: `kram_sort` und `kram_init_i_act`. Sie werden von den weiter unten beschriebenen Funktionen zum Propagieren, Initialisieren und Trainieren dieser Netze benötigt.

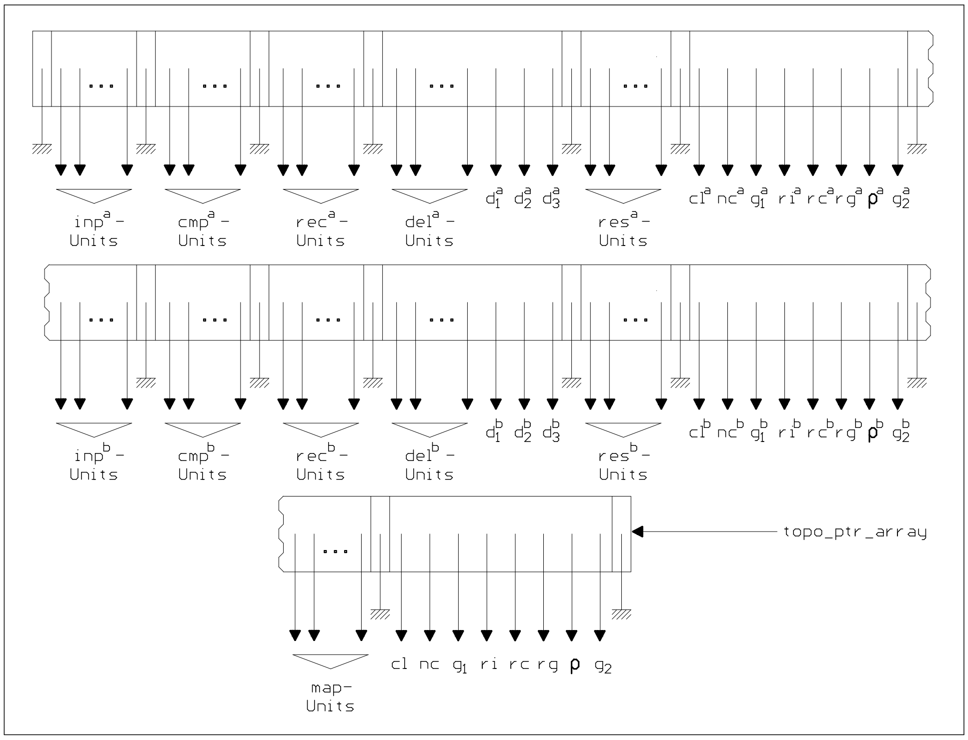


Abbildung 5.7: *Struktur des* `topo_ptr_array` *für ARTMAP-Netzwerke. Am Anfang des Feldes ist ein NULL-Pointer eingetragen, dann die Zeiger auf die Units von ART^a, in der Reihenfolge wie im* `topo_ptr_array` *für ART1-Netze. Danach sind die Zeiger auf die Units von ART^b in der gleichen Ordnung eingetragen. Schließlich die Zeiger auf das MAP-Field und dessen Kontroll-Units.*

`kram_sort` ist die Topologie-Check- und Sortierroutine für ARTMAP. Zum einen überprüft sie, ob ein gegebenes Netz der geforderten ARTMAP-Topologie entspricht, wie sie im letzten Abschnitt definiert wurde. Zum anderen legt sie ein Pointer-Feld an, in dem nacheinander Zeiger auf die Units in den verschiedenen Layern eingetragen werden. Siehe dazu Abbildung 5.7.

Entspricht ein Netz nicht der geforderten Topologie, so liefert `kram_sort` einen SNNS-Fehlercode zurück.

`kram_init_i_act` trägt die initialen Aktivierungen der ARTMAP-Units ein. Sie werden durch die Funktion `krui_resetNet` zur aktuellen Aktivierung. Dabei werden auch die Parameter $\overline{\rho^a}$, ρ^b und ρ in die ihnen jeweils entsprechende Unitstruktur übertragen. Diese Werte werden als Parameter an `kram_init_i_act` übergeben.

Die Initialisierungsfunktion

Während die festen Gewichte eines ARTMAP-Netzes während des Topologie-Checks (siehe oben) gesetzt werden, müssen die trainierbaren Verbindungen jederzeit initialisierbar sein. Dazu wurde die Initialisierungsfunktion `ARTMAP_Weights` im SNNS-Modul `init_f.c` implementiert.

Die Gewichte der $F_2^a \rightarrow F_1^a$ - und der $F_2^b \rightarrow F_1^b$ - Verbindungen werden dabei gemäß Gleichung 3.12 gesetzt. Die der $F_1^a \rightarrow F_2^a$ - und der $F_1^b \rightarrow F_2^b$ - Verbindungen jeweils gemäß den Gleichungen 3.25, ff. Dazu werden an `ARTMAP_Weights` die Parameter β^a, γ^a, β^b und γ^b in der angegebenen Reihenfolge übergeben. Zu deren Wahl siehe 3.3.

Die Parameter β^a und β^b werden jeweils in den Bias-Feldern der Units des ART^a bzw. des ART^b-Teilnetzes gespeichert, um später für die Anpassung der Gewichte wieder verfügbar zu sein.

ACHTUNG! Es sollte darauf geachtet werden, daß ARTMAP-Netze **immer** mit der eben beschriebenen Funktion (`ARTMAP_Weights`) initialisiert werden. Wird dazu eine andere SNNS-Initialisierungsfunktion verwendet, so ist das Verhalten des Netzes beim Lernen und Propagieren nicht vorhersagbar, da auch die festen Gewichte des Netzes mit anderen, als den für sie vorgesehenen Werten belegt werden können.

Die Lernfunktion

An die Funktion `ARTMAP` sind als Lernparameter die Werte für $\overline{\rho^a}$, ρ^b und ρ in der angegebenen Reihenfolge zu übergeben. Sie propagiert ein Eingabemuster – bestehend aus der Eingabe für ART^a und für ART^b– durch das Netz. Steht fest, daß das Muster klassifizierbar ist (**cl**-Unit), dann werden die trainierbaren Gewichte in ART^a, ART^b und die zwischen ART^a und MAP-Field angepaßt.

Wird dagegen die **nc**-Unit aktiv, dann ist die Eingabe nicht mehr klassifizierbar. Das passiert dann, wenn entweder in ART^a oder in ART^b keine neue F_2-Unit mehr vorhanden ist, und das momentan anliegende Eingabemuster in keine der bereits belegten Klassen eingeordnet werden kann.

`ARTMAP` gibt im von SNNS dafür vorgesehenen Feld keine Werte zurück. Die Funktion ist im SNNS-Modul `learn_f.c` implementiert.

Die Update-Funktionen

Auch für ARTMAP wurden zwei Update-Funktionen implementiert:

1. `ARTMAP_Stable` und
2. `ARTMAP_Synchronous`.

`ARTMAP_Stable` propagiert ein Muster solange, bis das Netz einen stabilen Zustand erreicht hat, das heißt, bis entweder **cl** oder **nc** aktiv ist. `ARTMAP_Synchronous` führt analog zu `ART1_Synchronous` nur jeweils einen Propagierungsschritt aus. An die Funktionen sind jeweils die Werte für $\overline{\rho^a}$, ρ^b und ρ zu übergeben.

Zur Bedienung mit der SNNS-Oberfläche XGUI, siehe Abschnitt 3.3

Die Übertragungsfunktionen

Hier soll wieder auf die Aktivierungsfunktionen eingegangen werden, die im Modul `arttr_f.c` implementiert wurden, nämlich die, die einen nur global zugänglichen Wert aus dem ARTMAP-Kernmodul `kr_amap.c` benötigen.

Das sind zum einen die Aktivierungsfunktionen für die $\mathbf{nc}^a$- und die $\mathbf{nc}^b$-Unit. Ihr Netzeingang muß einen Wert von M^a bzw. M^b übersteigen, damit sie aktiv werden. Diese Werte sind aus Gründen, die bereits in Abschnitt 3.3 genannt wurden, global definiert.

Zum anderen ist die Aktivierungsfunktion für die Unit $\Delta\rho$ in diesem Modul untergebracht. Auch wenn sie auf keinen globalen Wert zugreift, so ist sie doch so speziell, daß eine allgemeine Verwendbarkeit ausgeschlossen werden kann. Daher erschien es sinnvoll, auch diese Funktion nicht im SNNS-Modul `trans_f.c` zu implementieren.

Die Schnittstellenfunktionen

Für ARTMAP sind die beiden Schnittstellenfunktionen

- `artui_getClassifiedStatus`
- `artui_getClassNo`
- `artui_getNa`
- `artui_getMa`
- `artui_getNb`
- `artui_getMb`

relevant. Sie werden in Anhang A erläutert.

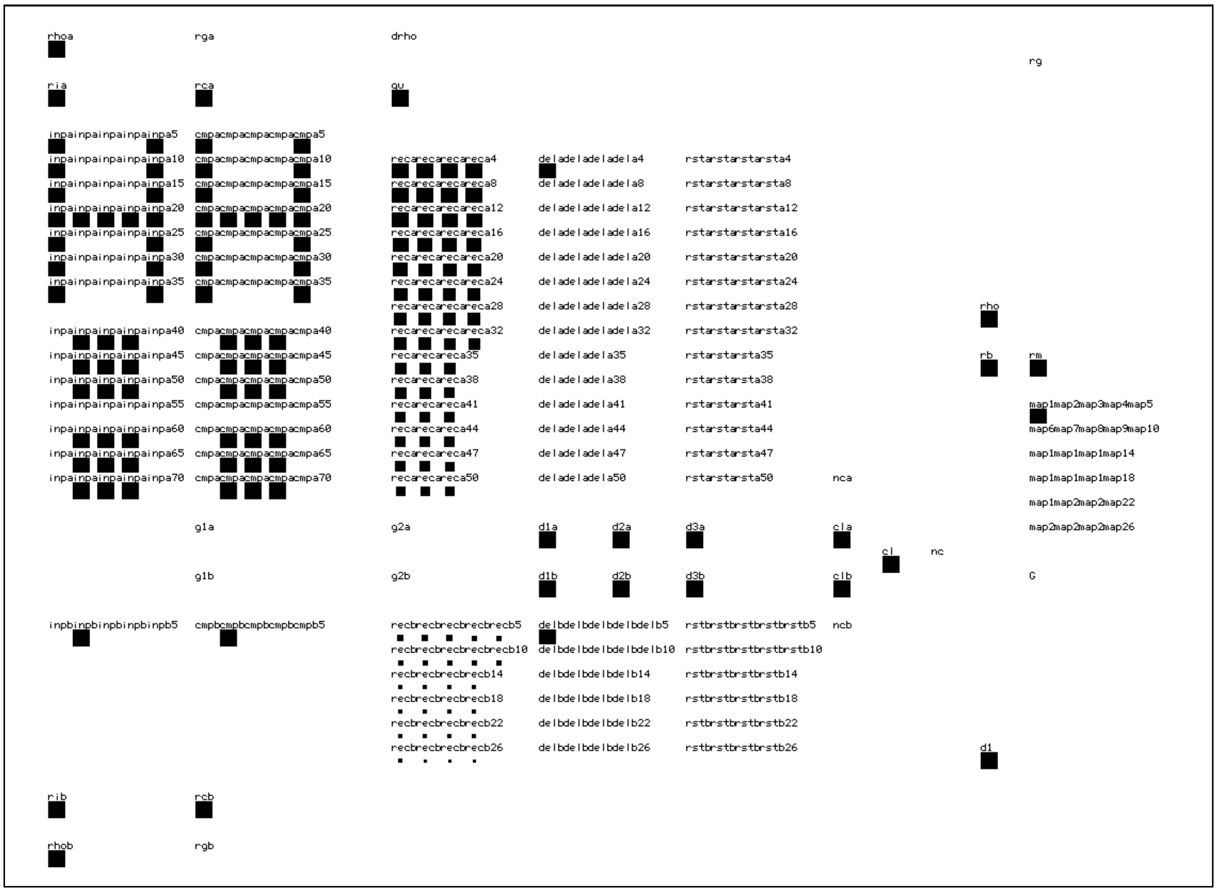


Abbildung 5.8: *Propagierung eines Buchstabens im untrainierten ARTMAP-Netz. Die Initialisierung erfolgte mit den Parametern* $\beta^a = \beta^b = \gamma^a = \gamma^b = 1.0$.

5.5 Ein Versuch mit einem ARTMAP-Netz

Ähnlich wie in Abschnitt 3.4 soll auch für ARTMAP ein kleines Beispiel das Kapitel beschließen. Wieder wird dazu ein 5×7-Buchstabensatz verwendet.

Das Netz

In Abbildung 5.8 findet sich die Darstellung eines ARTMAP-Netzes, wie es in der SNNS-Grafikoberfläche repräsentiert wird. ART^a und ART^b liegen parallel übereinander. Dabei ist jeweils links die Eingabeschicht. Am rechten Rand der Abbildung ist das MAP-Field.

ART^a besitzt zwei Eingabefelder mit je 5×7 Neuronen. Um die Buchstaben zu repräsentieren wurde Complement-Coding aus den oben genannten Gründen verwendet. ART^b erhält als Eingabe die Nummer des jeweiligen Buchstaben im Alphabet binär kodiert. In Abbildung 5.8 ist die vierte ART^b-Eingabezelle aktiv, folglich also eine Acht repräsentiert.

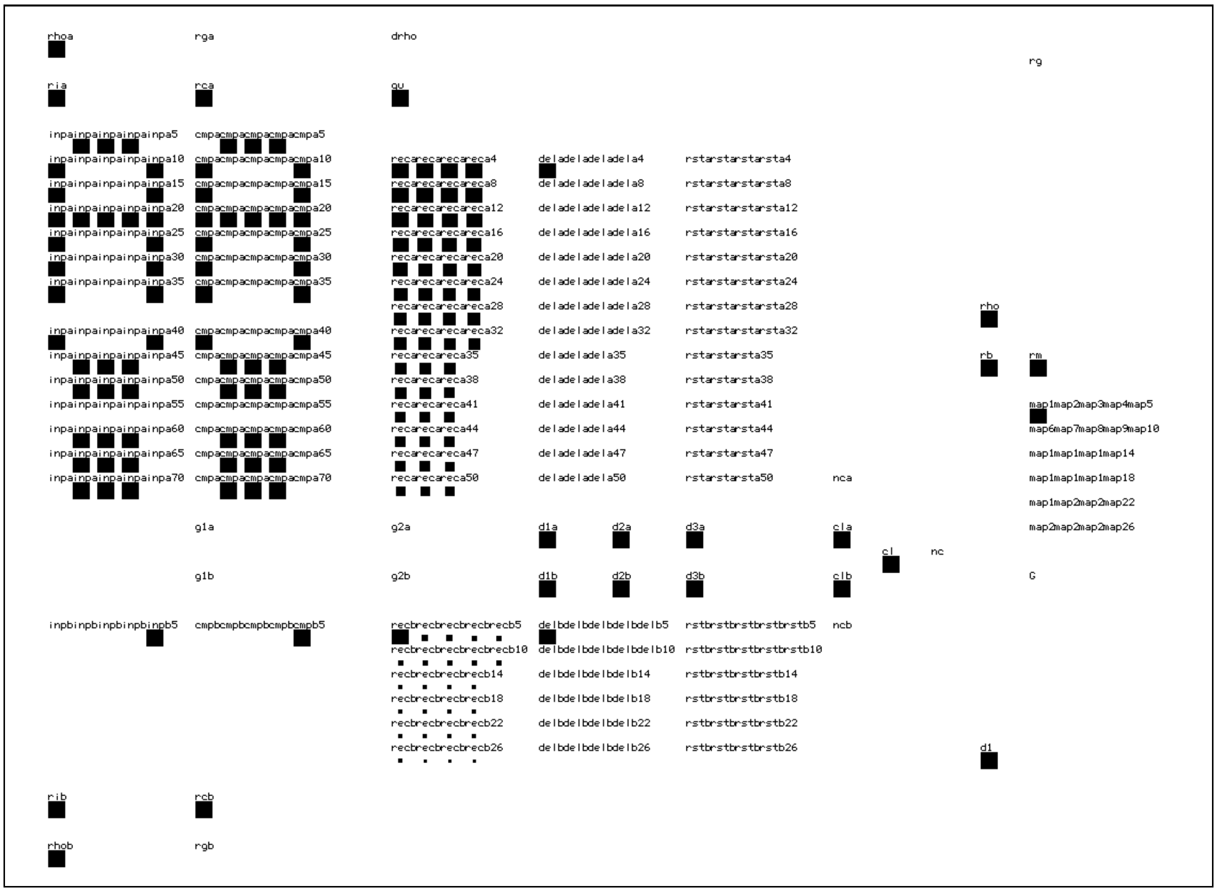


Abbildung 5.9: *Propagierung des Buchstabens **A** nach einem Trainingszyklus für dieses Muster. Die Initialisierung erfolgte mit den Parametern* $\beta^a = \beta^b = \gamma^a = \gamma^b = 1.0$. *Die Lernparameter waren* $\overline{\rho^a} = 0.8$, $\rho^b = 1.0$ *und* $\rho = 1.0$.

Propagierung im untrainierten Netz

Das Netz wird mit $\beta^a = \beta^b = \gamma^a = \gamma^b = 1.0$ initialisiert. Das Eingabemuster **H** wird angelegt und mit der Funktion `ARTMAP_Stable` durch das Netz propagiert.

Sowohl in ART^a als auch in ART^b gewinnt die jeweils erste Unit im Recognition-Layer. Damit wird auch die erste MAP-Field-Unit aktiviert und liefert das endgültige Ergebnis dieser Propagierung. Die erste Klasse würde nach einem Trainingszyklus für den Buchstaben **H** stehen. Neue Schrifttypen würden dann ein **H** immer auf diese Klasse abbilden, selbst wenn in ART^a eine zweite **H**-Kategorie entstehen würde.

Trainieren des Buchstaben A

In Abbildung 5.9 ist die Propagierung des Buchstaben **A** dargestellt, nachdem er einen Zyklus trainiert wurde. Die F^{ab}-Unit $\mathbf{map}_1$ repräsentiert danach die Klasse A. Die Lernparameter waren $\overline{\rho^a} = 0.8$, $\rho^b = 1.0$ und $\rho = 1.0$.

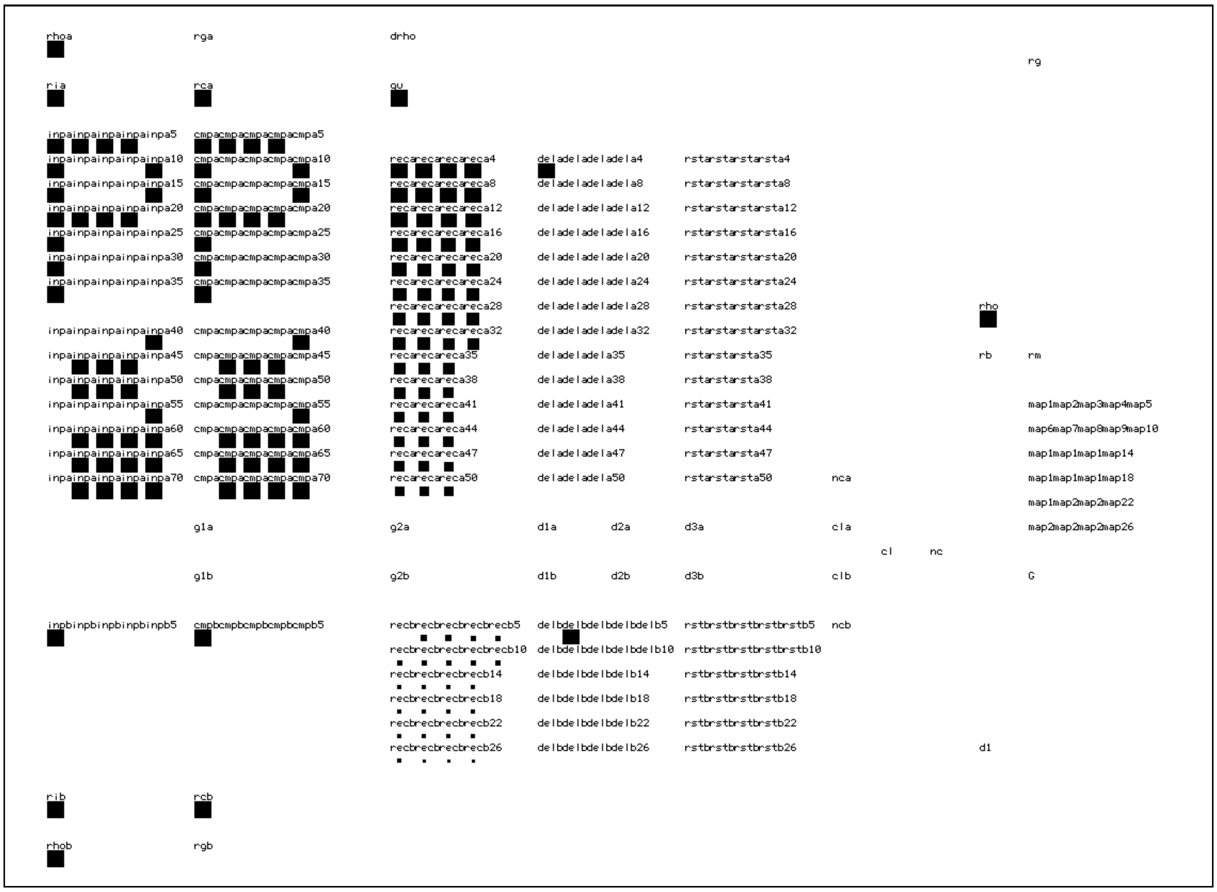


Abbildung 5.10: *Propagierung des Buchstabens **P** nach einem Trainingszyklus für den Buchstaben **A**. In beiden ART1-Teilnetzen haben sich die Gewinner ausgeprägt.*

Propagierung des Buchstaben P

Nach diesem Trainingszyklus soll jetzt der Buchstabe **P** schrittweise durch das Netz propagiert werden um die Reaktion des Netzes nachzuvollziehen. Dazu werden die Parameter gewählt, wie oben.

In Abbildung 5.10 haben sich bereits in ART^a und ART^b die Gewinnerklassen ausgebildet. In ART^a gewinnt erneut die erste Klasse, in ART^b die zweite.

Einige Propagierungszyklen später ist an den Units $\mathbf{cl}^a$ und $\mathbf{cl}^b$ erkennbar, daß beide ART1-Teilnetze die Muster intern bereits klassifiziert haben (Abbildung 5.11). Dabei hat ART^a entschieden, daß das Muster für **P** dem für **A** ähnlich genug ist, um in die gleich interne Klasse zu gehören.

Im gleichen Bild ist aber auch erkennbar, daß ART^b die Entscheidung von ART^a nicht akzeptiert. Es ist keine MAP-Field-Unit aktiv, denn ART^a sensibilisiert $\mathbf{map}_1$, während ART^b versucht $\mathbf{map}_2$ zu aktivieren. Ein Inter-ART-Reset ist die Folge, **rg** ist aktiv (oben rechts).

Auch die Unit $\Delta\rho$ (im Bild mit „drho" bezeichnet) ist bereits aktiviert und wird im nächsten Propagierungszyklus ρ^aerhöhen.

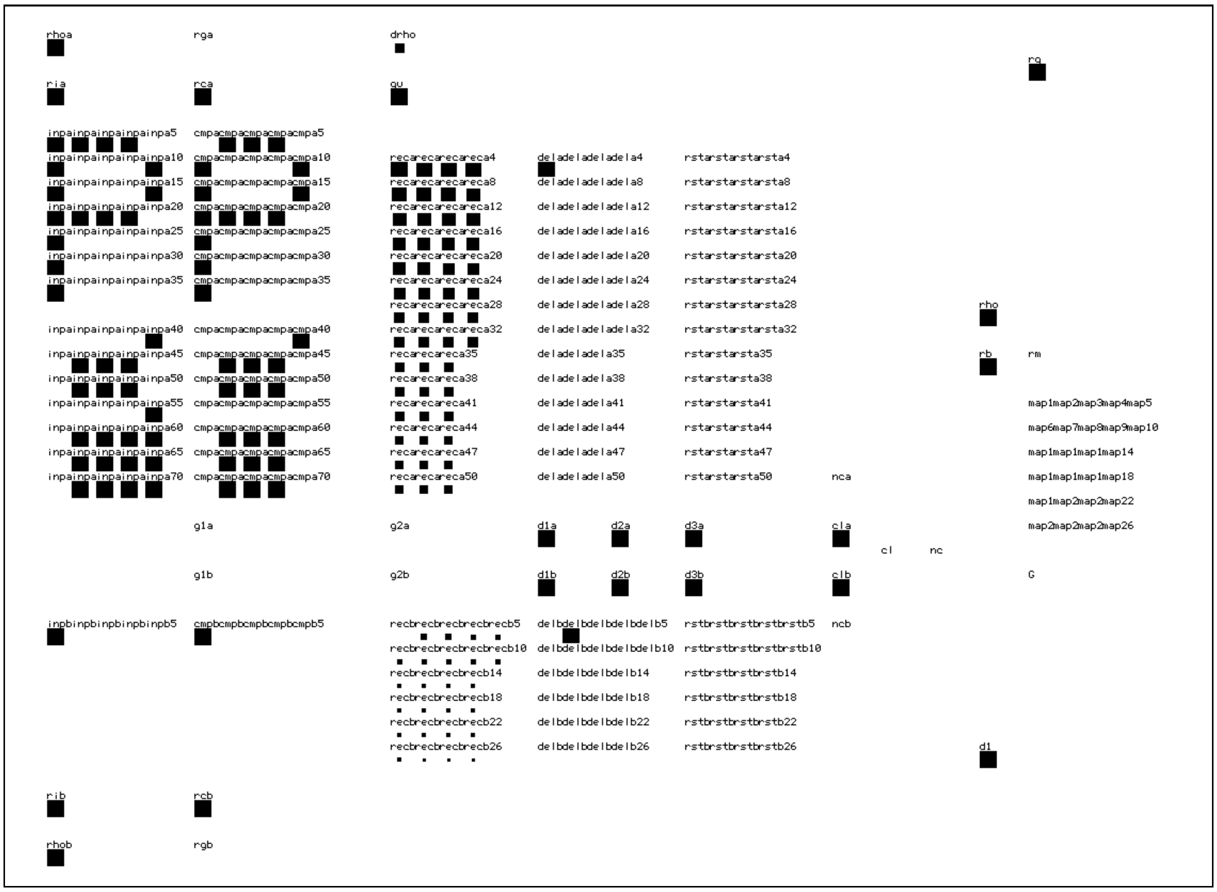


Abbildung 5.11: *Nach weiteren Propagierungszyklen erfolgt ein Inter-ART-Reset und ein Match-Tracking-Zyklus.*

Dadurch wird in ART^a ein Reset erzwungen und im folgenden ein Suchzyklus ausgelöst. In Abbildung 5.12 ist die neue ART^a-Gewinnerzelle ermittelt, die noch unbelegt war und somit eine Aktivierung der zweiten MAP-Field-Unit zuläßt.

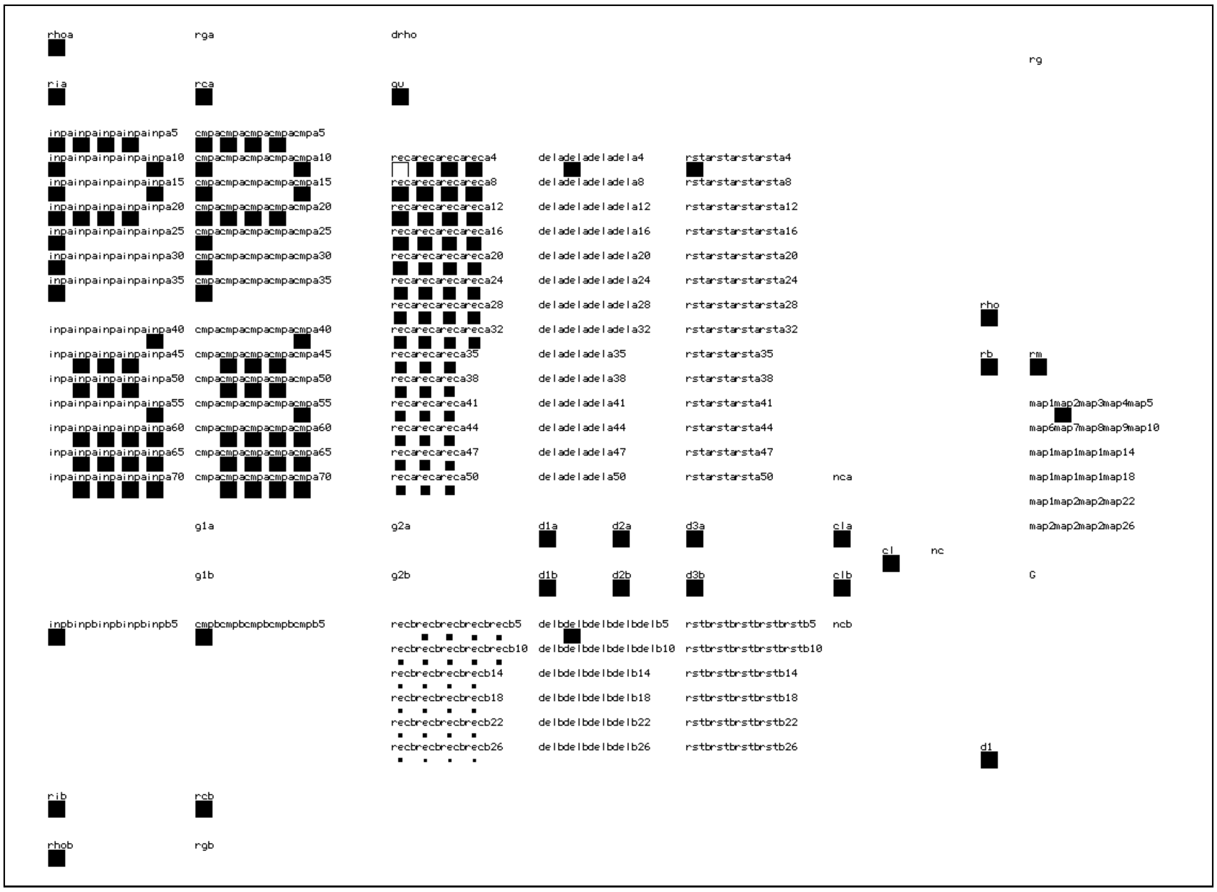


Abbildung 5.12: *Ein Suchzyklus in ART^a sorgt für einen neuen Gewinner, der auf die von ART^b geforderte Klasse abgebildet werden kann.*

Kapitel 6

Bewertung und Ausblick

In diesem abschließenden Kapitel sollen die drei behandelten Verfahren noch einmal zur Sprache kommen. Die technischen Aspekte wurden in den vorangegangenen Kapiteln detailliert behandelt. Hier sollen nun die Vorzüge und Schwächen der Verfahren zusammengefaßt werden.

6.1 Beurteilung von ART 1

Aufgrund der Erfahrungen, die im Laufe dieser Arbeit gesammelt werden konnten, läßt sich über ART 1 ein positives Bild zeichnen. Die Versuche, die damit durchgeführt wurden, bestätigten die Erwartung eines stabilen Lernverhaltens. Eine Stabilisierung findet dabei stets innerhalb von wenigen (2 bis 5) Trainingszyklen statt, so daß das Verfahren als sehr schnell bezeichnet werden kann. Die Bedienung ist sehr einfach, da nur ein Parameter eingestellt werden muß. Dessen Einfluß auf das Lernverhalten ist leicht zu verstehen, was dazu beiträgt, daß schnell gute Trainingserfolge erzielt werden können.

Es handelt sich um einen interessanten Ansatz, der trotz seiner künstlich wirkenden Steuerungen ein natürliches Fundament besitzt. So ist bei Carpenter und Grossberg stets von Kurz- und Langzeitgedächtnis die Rede. Das Kurzzeitgedächtnis sind die Aktivierungen der Units, die sich sehr schnell ändern können, das Langzeitgedächtnis ist durch die Gewichte der Verbindungen repräsentiert, die nach einer gewissen Zeit sehr stabil sind. Auch das bereits in Kapitel 2 erwähnte Propagieren einer Erwartungshaltung aufgrund eines äußeren Reizes, sowie dessen anschließende Überprüfung, beruht auf einem natürlichen Konzept, das oben bereits diskutiert wurde.

Problematisch scheint es dagegen zu sein, konkrete Klassifikationsaufgaben für binäre Muster zu finden. Die meisten Anwendungen erfordern die Möglichkeit von reellwertigen Eingabevektoren, so daß die meisten interessanten Probleme auf ART 2 angewiesen sind.

6.2 Beurteilung von ART 2

Auch ART 2 läßt abschließend einen positiven Eindruck zurück. Die Konzeption, die so gewählt ist, daß reellwertige Eingabemuster – global gesehen – ebenso klassifiziert werden können, wie binäre in ART 1, hält die Versprechen der Entwickler. In verschiedenen kleinen Versuchen konnten gegenüber anderen Klassifikationsverfahren, mindestens ebenso gute Ergebnisse erzielt werden.

Das Erlernen des Umgangs mit den Parametern von ART 2 bedarf jedoch einiger Zeit. Es sind zahlreiche eigene Experimente notwendig, um genügend Erfahrung zu sammeln, die es erlauben, die Auswirkung der Parameter auf das Verhalten des Netzes richtig einschätzen zu können. Außer für den Ähnlichkeitsparameter ρ kann für keinen der Parameter eine intuitive Erklärung gegeben werden, die über eine geeignete Wahl Auskunft gibt.

Ein weiteres Problem von ART 2 ist die potentielle Unfähigkeit, Cluster zu trennen, die unter Umständen weit auseinanderliegen. Dieses Problem ergibt sich dadurch, daß ART 2 normierte Vektoren lernt. Alle Ortsvektoren, die in die gleiche Richtung weisen, zeigen schließlich auf den gleichen Punkt. Auch sehr weit auseinanderliegende und somit deutlich zu unterscheidende Cluster sind so von ART 2 unter Umständen nicht mehr trennbar. Wichtig ist nur die Richtung des Vektors, nicht seine Länge.

Die Stabilisierung der Gewichte oder des „Langzeitgedächtnisses" dauert bei ART 2 etwas länger als bei ART 1 (30 bis 50 Trainingszyklen).

6.3 Beurteilung von ARTMAP

Die Anwendungen für überwachtes Lernen sind sehr zahlreich. ARTMAP bietet als einziges überwachtes Lernverfahren Stabilität. Diesen Vorteil muß man sich jedoch teuer erkaufen, denn ARTMAP-Netze sind vergleichsweise groß, sie haben sehr viele Units und Verbindungen.

Weiter läßt das Verfahren nur eine kodierte Ausgabe zu. Die Ausgabe von ARTMAP erfolgt im MAP-Field und nicht dort, wo die Lerneingabe angelegt wird. Das ART^b-Teilnetz kodiert also zunächst die Lerneingabe und dieser Kode ist es dann, der als Ausgabe des Netzes verwendet wird. Auch eine Ermittlung des Netzfehlers ist dadurch nicht in der Form möglich, wie man es von anderen überwachten Lernverfahren gewohnt ist.

In Kapitel 5 war die Rede davon, daß statt zweier ART 1-Netze ebensogut zwei ART 2-Netze verwendet werden können, um ARTMAP zu realisieren. Man bedenke aber, daß der Match-Tracking-Mechanismus für ein ART 2-Netz völlig anders ausehen müßte, da die Bedeutung des Ähnlichkeitsparameters hier eine andere ist.

Insgesamt entspricht ARTMAP also zwar den Anforderungen eines überwachten Lernverfahrens und ist dabei sogar stabil, für reale Anwendungen ist es jedoch aufgrund der Größe der Netze oft ungeeignet.

6.4 Ausblick

ART 2 im Vergleich

Um ART 2 wirklich einschätzen zu können, wäre es sicherlich wichtig, ART 2 anderen Klassifikationsverfahren gegenüber zu stellen. Leider ist ein solcher Vergleich im Rahmen dieser Arbeit nicht möglich. Die wenigen Versuche, für die Ergebnisse von anderen Verfahren vorlagen und die mit ART 2 durchgeführt werden konnten, lassen kein abschließendes Urteil zu.

ART mit dynamischer F_2-Schicht

In sämtlichen ART-Modellen ist man mit dem Problem konfrontiert, die richtige Zahl an Klassifizierungszellen zu wählen. Einerseits möchte man alle Vektoren klassifizieren, andererseits sollte die Zahl an F_2-Units nicht zu groß gewählt werden, da jede von ihnen N Verbindungen mit sich bringt. Diese Verbindungen müssen dann mit jedem Propagierungszyklus berechnet werden, auch wenn die Klasse unbelegt ist.

Im Laufe dieser Arbeit kam deswegen die Idee auf, die Zahl der Klassifizierungszellen dynamisch zu gestalten. Vorstellbar wäre beispielsweise, am Anfang nur eine solche Unit im Netz zu haben. Sobald sie belegt wird, wird eine zweite F_2-Zelle generiert, die dann mit der ersten, jetzt belegten Unit konkurriert. Wird dann die zweite Klasse belegt, so wird eine dritte Unit generiert und so weiter.

Da die Wahl der initialen Verbindungsgewichte in allen ART-Modellen nur von der Zahl der F_1-Zellen, nicht aber von der Zahl der F_2-Zellen abhängt wäre das Generieren neuer Units für die F_2-Schicht, sowie die Initialisierung der Gewichte ihrer Verbindungen kein Problem.

Diese Modelle hätten einen großen Geschwindigkeitsvorteil, da viele unnötige Rechenoperationen, vor allem zu Beginn des Trainings, entfallen würden.

Anhang A

Schnittstellenfunktionen

für die Programmierung von Anwendungen mit ART-Netzwerken

Um Anwendungen mit ART-Netzwerken zu programmieren sind neben Lern-, Update- und Initialisierungsfunktionen einige andere Prozeduren notwendig. So ist es beispielsweise wünschenswert abfragen zu können, ob das Netzwerk ein Muster klassifiziert hat oder welcher Klasse ein Muster zugeordnet wurde.

Das Modul

Für die Implementierung dieser Schnittstelle zwischen ART-Kernfunktionen und Anwendung wurde dem SNNS ein neues Modul hinzugefügt: `art_ui.c`

Um die Funktionen nutzen zu können sind neben den normalerweise einzubindenden Header-Files

- `kr_ui.h`
- `glob_typ.h`

die beiden folgenden zusätzlich notwendig:

- `art_ui.h` für die Funktionsprototypen und
- `art_typ.h` für die Deklarationen und Definitionen.

Die Deklarationen

Typdefinitionen

Typname	Erklärung
`art_cl_status`	Mögliche Werte sind • `ART_NO_CLASSIFICATION` • `ART_CLASSIFIED` • `ART_NOT_CLASSIFIABLE` • `ART_DONT_KNOW` Status eines ART-Netzwerks.

Wertdefinitionen

Name	Erklärung
`ART_NO_CLASSIFICATION`	Undefinierter Klassifikationsstatus.
`ART_CLASSIFIED`	Muster klassifiziert.
`ART_NOT_CLASSIFIABLE`	Muster nicht klassifizierbar.
`ART_DONT_KNOW`	Nur bei ARTMAP! Netz kann keine Aussage über eine Kategoriezugehörigkeit machen. (wenn alle MAP-Field-Units aktiv sind)

Funktionen

Funktion:	`artui_getClassificationStatus`
Rückgabewert:	`krui_err` (siehe [Zell92])
Eing.Parameter:	KEINE
Ausg.Parameter:	`art_cl_status *`
Modelle:	ART 1, ART 2, ARTMAP
Erklärung:	Liefert den aktuellen Klassifikationsstatus eines ART-Netzwerks im Ausgangsparameter zurück. Die möglichen Werte sind der Tabelle Typdefinitionen zu entnehmen. Als Funktionswert wird ein SNNS-Fehlercode zurückgeliefert.

Funktion:	`artui_getClassNo`
Rückgabewert:	`krui_err` (siehe [Zell92])
Eing.Parameter:	KEINE
Ausg.Parameter:	`int *`
Modelle:	ART 1, ART 2, ARTMAP
Erklärung:	Liefert eine Zahl zwischen 1 und M im Ausgangsparameter zurück, die die Nummer der momentan aktivierten Klasse angibt. Ist keine Klasse aktiv, dann wird -1 zurückgegeben. Als Funktionswert wird ein SNNS-Fehlercode zurückgeliefert.

Funktion:	`artui_getN`
Rückgabewert:	`int`
Eing.Parameter:	KEINE
Ausg.Parameter:	KEINE
Modelle:	ART 1, ART 2
Erklärung:	Liefert die Zahl der Eingabeneuronen (N) zurück.

Funktion:	`artui_getM`
Rückgabewert:	`int`
Eing.Parameter:	KEINE
Ausg.Parameter:	KEINE
Modelle:	ART 1, ART 2
Erklärung:	Liefert die Zahl der Klassifikationsneuronen (M) zurück.

Funktion:	`artui_getNa`
Rückgabewert:	`int`
Eing.Parameter:	KEINE
Ausg.Parameter:	KEINE
Modelle:	ARTMAP
Erklärung:	Liefert die Zahl der Eingabeneuronen (N^a) des ART^a-Teilnetzes zurück.

Funktion:	`artui_getMa`
Rückgabewert:	`int`
Eing.Parameter:	KEINE
Ausg.Parameter:	KEINE
Modelle:	ARTMAP
Erklärung:	Liefert die Zahl der Klassifikationsneuronen (M^a) des ART^a-Teilnetzes zurück.

Funktion:	`artui_getNb`
Rückgabewert:	`int`
Eing.Parameter:	KEINE
Ausg.Parameter:	KEINE
Modelle:	ARTMAP
Erklärung:	Liefert die Zahl der Eingabeneuronen (N^b) des ART^b-Teilnetzes zurück.

Funktion:	`artui_getMb`
Rückgabewert:	`int`
Eing.Parameter:	KEINE
Ausg.Parameter:	KEINE
Modelle:	ARTMAP
Erklärung:	Liefert die Zahl der Klassifikationsneuronen (M^b) des ART^b-Teilnetzes zurück.

Anhang B

Topologiebeschreibungen

Den folgenden Tabellen sind die exakten Topologiebeschreibungen für die drei behandelten ART-Modelle zu entnehmen.

Für ARTMAP sind die Topologien der beiden ART1-Teilnetze ganau gleich wie in der ART1-Tabelle angegeben. Daher werden in der ARTMAP-Tabelle nur die neu hinzukommenden Units und Verbindungen genannt. Die Unit- und Sitenamen tragen bei den ART1-Teilnetzen ART^a und ART^b als Modifikatoren jeweils ein a oder ein b. Diese Modifikatoren sind in der ART1-Tabelle nicht angegeben.

In der linken Spalte der Tabelle (*Unitname*) ist jeweils die Bezeichnung der Unit(s) angegeben, die in ihrer Zeile beschrieben werden soll(en). Ist diese Bezeichnung mit i oder j indiziert, z. B. $\mathbf{inp}_i$, so bedeutet dies, daß die Beschreibung für alle Units dieser Schicht gilt. Bei der Indizierung i gilt: $1 \leq i \leq N$, bei Indizierung j gilt: $1 \leq j \leq M$.

Ist in der Spalte *Zielunit* der gleiche Index verwendet, wie in der Spalte *Unitname*, dann hat die Quellunit eine Verbindung zu der ihr entsprechenden Unit in der Zielschicht.
(Beispiel: $\mathbf{inp}_i \ldots \mathbf{cmp}_i$).

Ist dagegen ein anderer Index in der Spalte *Zielunit* angegeben und dahinter mit ∀Index versehen, so heißt das, daß die Quellzelle zu allen Units des entsprechenden Layers Verbindungen hat. (Beispiel: $\mathbf{cmp}_i \ldots \mathbf{rec}_j\ \forall j$).

In der Spalte *top. Typ* (= topologischer Typ) bedeuten:

i : INPUT,
h : HIDDEN,
s : SPECIAL.

Modell: ART 1 und ARTMAP-Teilnetze ART^a, ART^b

Sitedefinition		
Sitename	Sitefunktion	
`Site_rst_self`	`Site_WeightedSum`	
`Site_rst_signal`	`Site_at_least_2`	
`Site_inp_g1`	`Site_at_least_1`	
`Site_rec_g1`	`Site_at_most_0`	
`Site_inp_ri`	`Site_WeightedSum`	
`Site_rho_ri`	`Site_WeightedSum`	

Unitdefinition					Verbindungen	
Unit Name	top. Typ	Aktivierungs-funktion	Ausgabe-funktion	Site-Namen	Ziel-Unit	Ziel-Site

inp$_i$	i	`Act_Identity`	`Out_Identity`		**cmp**$_i$
					g$_1$
					ri
					g$_2$
						
						`Site_inp_g1`
						`Site_inp_ri`

cmp$_i$	h	`Act_at_least_2`	`Out_Identity`		**rec**$_j$ $\forall j$ **rc**	
rec$_j$	s	`Act_Identity`	`Out_Identity`		**del**$_j$ **g**$_1$	 `Site_rec_g1`
del$_j$	h	`Act_at_least_2`	`Out_Identity`		**cmp**$_i$ $\forall i$ **d**$_1$ **rst**$_j$	 `Site_rst_signal`
d$_1$	h	`Act_at_least_1`	`Out_Identity`		**d**$_2$	
d$_2$	h	`Act_at_least_1`	`Out_Identity`		**d**$_3$	
d$_3$	h	`Act_at_least_1`	`Out_Identity`		**cl**	
rst$_j$	h	`Act_at_least_1`	`Out_Identity`	`Site_rst_self` `Site_rst_signal`	**rst**$_j$ **rec**$_j$	`Site_rst_self`
cl	h	`Act_at_least_1`	`Out_Identity`			

nc	h	`Act_ART1_NC`
		für ARTMAP:
		`Act_ARTMAP_NCa`,
		`Act_ARTMAP_NCb`
			`Out_Identity`			

g$_1$	h	`Act_at_least_2`	`Out_Identity`	`Site_inp_g1` `Site_rec_g1`	**cmp**$_i$ $\forall i$	
g$_2$	h	`Act_at_most_0`	`Out_Identity`		**rec**$_j$ $\forall j$ **cl**	
ri	h	`Act_Product`	`Out_Identity`	`Site_inp_ri` `Site_rho_ri`	**rg**	
rc	h	`Act_Identity`	`Out_Identity`		**rg**	
rg	h	`Act_less_than_0`	`Out_Identity`		**rec**$_j$ $\forall j$ **rst**$_j$ $\forall j$ **cl**	 `Site_rst_signal`
rho	h	`Act_Identity`	`Out_Identity`		**rho** **ri**	 `Site_rho_ri`

Modell: ARTMAP

Sitedefinition		
Sitename	Sitefunktion	
Site_ARTa_G	Site_at_least_1	
Site_ARTb_G	Site_at_least_1	
Site_ARTb_rb	Site_WeightedSum	
Site_rho_rb	Site_WeightedSum	
Site_inpa_qu	Site_Reciprocal	
Site_cmpa_qu	Site_WeightedSum	

Unitdefinition					Verbindungen	
Unit Name	top. Typ	Aktivierungs-funktion	Ausgabe-funktion	Site-Namen	Ziel-Unit	Ziel-Site
map$_j$	h	Act_at_least_2	Out_Identity		**rm**	
cl	h	Act_at_least_2	Out_Identity			
nc	h	Act_at_least_1	Out_Identity			
G	h	Act_exactly_1	Out_Identity	Site_ARTa_G Site_ARTb_G	**map**$_j$ $\forall j$	
d$_1$	h	Act_Identity	Out_Identity		**rb**	Site_ARTb_rb
rb	h	Act_Product	Out_Identity	Site_ARTb_rb Site_rho_rb	**rg**	
rm	h	Act_Identity	Out_Identity		**rg**	
rg	h	Act_less_than_0	Out_Identity		**drho** **cl**	
rho	h	Act_Identity	Out_Identity		**rho** **rb**	 Site_rho_rb
qu	h	Act_Product	Out_Identity	Site_inpa_qu Site_cmpa_qu	**drho**	
drho	h	Act_ARTMAP_DRho	Out_Identity		**drho** **rho**^{a}	
inp$_i^a$	siehe ART1-Tabelle				**qu**	Site_inpa_qu
cmp$_i^a$	siehe ART1-Tabelle				**qu**	Site_cmpa_qu
rec$_i^a$	siehe ART1-Tabelle				**G**	Site_ARTa_G
rec$_i^b$	siehe ART1-Tabelle				**G**	Site_ARTb_G
del$_i^b$	siehe ART1-Tabelle				**d**$_1$	
cl^a	siehe ART1-Tabelle				**cl**	
cl^b	siehe ART1-Tabelle				**cl** **drho**	
nc^a	siehe ART1-Tabelle				**nc**	
nc^b	siehe ART1-Tabelle				**nc**	
rg^a	siehe ART1-Tabelle				**drho**	
rho^a	siehe ART1-Tabelle				**drho**	

Modell: ART 2

Unitdefinition					Verbindungen	
Unit Name	top. Typ	Aktivierungs-funktion	Ausgabe-funktion	Site-Namen	Ziel-Unit	Ziel-Site
$\mathbf{inp}_i$	i	`Act_Identity`	`Out_Identity`		$\mathbf{w}_i$ $\mathbf{r}_i$	
$\mathbf{w}_i$	h	`Act_ART2_Identity`	`Out_Identity`		$\mathbf{x}_i$	
$\mathbf{x}_i$	h	`Act_ART2_NormW`	*Signalfunktion*[1]		$\mathbf{v}_i$	
$\mathbf{v}_i$	h	`Act_ART2_Identity`	`Out_Identity`		$\mathbf{u}_i$	
$\mathbf{u}_i$	h	`Act_ART2_NormV`	`Out_Identity`		$\mathbf{p}_i$ $\mathbf{w}_i$	
$\mathbf{p}_i$	h	`Act_ART2_Identity`	`Out_Identity`		$\mathbf{q}_i$ $\mathbf{r}_i$ $\mathbf{rec}_j\ \forall j$	
$\mathbf{q}_i$	h	`Act_ART2_NormP`	*Signalfunktion*[1]		$\mathbf{v}_i$	
$\mathbf{rec}_j$	s	`Act_ART2_Rec`	`Out_Identity`		$\mathbf{p}_i\ \forall i$ $\mathbf{rst}_j$	
$\mathbf{rst}_j$	h	`Act_ART2_Rst`	`Out_Identity`		$\mathbf{rec}_j$	

[1] Wahlweise: `Out_ART2_Noise_ContDiff` oder `Out_ART2_Noise_PLin`.

Literaturverzeichnis

[Carpenter89] G.A. Carpenter: *Neural Network Models for pattern recognition and associative memory*; Neural Networks, Vol. 2, S. 243-257, 1989

[Carpenter87a] G.A. Carpenter, S. Grossberg: *A Massively Parallel Architecture for a Self-organizing Neural Pattern Recognition Machine*; Computer Vision, Graphics and Image Processing, Vol. 37, S. 54-115, 1987

[Carpenter87b] G.A. Carpenter, S. Grossberg: *ART 2: Stable self-organization of pattern recognition codes for analog input patterns*; Applied Optics, Vol. 26, S. 4919-4930, 1987

[Carpenter88] G.A. Carpenter, S. Grossberg: *The ART of Adaptive Pattern Recognition by a self-organizing neural network*; IEEE Computer, Vol. 21, S. 77-88, 1988

[Carpenter90] G.A. Carpenter, S. Grossberg: *Adaptive Resonance Theory: Neural Network Architectures for self-organizing pattern recognition*; aus: R. Eckmiller, G. Hartmann, G. Hauske: *Parallel Processing in Neural Systems and Computers*, Elsevier Science Publishers B.V. (North Holland), 1990

[Carpenter91a] G.A. Carpenter, S. Grossberg: *ART 3: Hierarchical Search Using Chemical Transmitters in Self-Organizing Pattern Recognition Architectures*; Neural Networks, Vol. 3, No. 2, S. 129-152, 1991

[Carpenter91b] G.A. Carpenter, S. Grossberg: *ARTMAP: Supervised Real-Time Learning and Classification of Nonstationary Data by a Self-Organizing Neural Network*; Neural Networks, Vol. 4, No. 5, S. 543-564, 1991

[Carpenter91c] G.A. Carpenter, S. Grossberg, J.H. Reynolds: *A Self-Organizing ARTMAP Neural Architecture for Supervised Learning and Pattern Recognition*; aus: R.J. Mammone, Y. Zeevi, *Neural Networks*, Academic Press Inc., Harcourt Brace Jovanovich, Publishers, S. 43-80, 1991

[Carpenter91d] G.A. Carpenter, S. Grossberg, D.B. Rosen: *Fuzzy ART: Fast Stable Learning and Categorization of Analog Patterns by an Adaptive Resonance System*; Neural Networks, Vol. 4, S. 759-771, 1991

[Kemke88] C. Kemke: *Der neuere Konnektionismus* Informatik Spektrum, Vol. 11, S. 143-162, 1988

[Moore88] B. Moore: *ART 1 and Pattern Clustering*; Proceedings of the 1988 Connectionist Models Summer School, 1988

[McClelland88] J.L. McClelland, D.E. Rumelhart: *Explorations in Parallel Distributed Processing*; MIT Press; 1988

[Rumelhart86] D.E. Rumelhart, J.L. McClelland, The PDP Research Group: *Parallel Distributed Processing*; Vol. 1, Vol. 2, MIT Press, Cambridge, MA, 1986

[Zell92] A. Zell, N. Mache, R. Hübner, M. Schmalzl, T. Somer, G. Mamier, M. Vogt: *SNNS User Manual, Version 2.1*; Bericht Nr. 8/92; Institut für Parallele und Verteilte Höchstleistungs-Rechner (IPVR); Universität Stuttgart; Selbstverlag; 1992