Fortgeschrittenenpraktikum

Hebb-Netz & Reinforcement

Thema: Konstruktion von Netzen mit
REINFORCEMENT-Input,
die nur mittels der HEBB-Regel lernen.

Bearbeiter: Marcus Hutter, MatNr.: 0893341

Betreuer: Gerhard Weiß

Abgabedatum: 2.Juli.1990

Inhaltsverzeichnis

1 Idee / Einleitung 1

2 Reinforcement-Hebb-Netze ohne Hidden-Units 3

3 Konstruktionsversuche von RH-Netzen mit Hidden-Units 9

4 Anmerkungen / Ausblicke 12

5 Literaturverzeichnis 13

6 Programmlisting 14

7 Protokoll von Beispiel 4 des Programms 26

1 Idee / Einleitung

Hier, in der Einleitung möchte ich die Idee, die hinter diesem Thema steckt, und das Thema selbst etwas näher erläutern. Dazu sei vorweg erwähnt, daß die meisten Vergleiche mit der Biologie und erst recht die Analogien zum Menschen recht spekulativ sind und nur zur Anregung dienen sollten.

zum Verständnis notwendige Grundkenntnisse

- Reinforcement-Netze
- Hebb-Netze
- Darwinistische Evolution

Übersicht über Netze - Gegenüberstellung

Aus zwei Gründen möchte ich zuerst eine kurze Übersicht über verschiedene Charakteristika neuronaler Netze geben. Da erstens die Vielfalt dieser Netze mit der Idee und damit dem Thema zu tun hat und zweitens ich einige darunter implementiert habe und die Auswahl anhand der Tabelle begründen möchte. Die Tabelle enthält nur die für diesen Artikel notwendigen Punkte.

Supervised ⟷	Unsupervised	⟷ Reinforcement
Backpropagation	⟷	Hebb-Lernregel
Statistische Units	⟷	Deterministische Units
Schwellwert Units	⟷	Sigmoid-Units
Synchroner Update	⟷	Asynchroner Update
Impuls-Units	⟷	Statischer Unit-Output

Idee

Supervised-Netze (z.B. mit Backprop.-Algorithmus) sind eine der erfolgreichsten Netze, haben aber den entscheidenden Nachteil ständig auf einen Lehrer angewiesen zu sein, der ihnen ,,sagt was zu tun ist". So ein Lehrer ist aber in vielen Problemen nicht verfügbar. Wesentlich bescheidener sind da die Reinforcement-Netze (nachfolgend kurz R-Netze), denen man nur ,,sagen" muß, ob ihre Reaktion richtig oder falsch war.

Dies entspricht in vieler Hinsicht auch mehr dem menschlichem Lernen durch 'trial and error'. Allerdings gibt es auch bedeutsame Unterschiede, die dieses Fopra anzupacken versucht. Ganz so sporadisch wie bei den R-Netzen ist die Rückkoppelung in der Natur nämlich nicht. Das R-Signal könnte man als das allgemeine Wohlbefinden des Individuums betrachten, welches es zu maximieren gilt. Doch dieses Signal wird weder ausschließlich

noch direkt von der Umwelt geliefert. Vielmehr nehmen wir mehrere R-Signale in codierter Form über unsere normalen Sinnesorgane auf. Um etwas konkreter zu werden, gebe ich hier einige Beispiele:

- zu helles Licht	schmerzt	in den Augen	(R $<$ 0)
- gute Mahlzeit	bekommt	Zunge & Magen	(R $>$ 0)
- Tritt ans Schienbein	tut weh	am Bein	(R $<$ 0)
- heitere Musik	erfreut	das Gemüt	(R $>$ 0)

Ähnliche Beispiele lassen sich natürlich auch im technischen Bereich finden, etwa für einen Staubsauger-Roboter. Neurologisch scheint unser Gehirn noch weniger ein R-Netz zu sein, als vielmehr ein Unsupervised-Netz (nachfolgend kurz U-Netz), das wesentlich mit Hilfe der Hebb-Regel lernt. Doch unbestreitbar besitzen wir die Fähigkeit des R-Lernens. Die Punkte noch einmal zusammenstellend

- codierte R-Info. an Standardinputs, aber kein R-Eingang
- Unsupervised Netz mit Hebb-Lernregel
- Fähigkeit des R-Lernens

ergibt sich die Frage: Wie kann ein Hebb-Netz R-Lernen ? Dies ist der Titel des Fopras.

Lösung

Obige Annahmen vorausgesetzt, hat es die Evolution anscheinend geschafft, aus einfachen U-Netzen durch Modifikation der Netzstruktur Rähnliche Netze zu konstruieren. Man kann dies auch anders sehen (vergleiche Kap. 3 in [2]): Der Lernprozess im Gehirn stellt auch eine Art Evolution auf einer kleinen Zeitskala dar (somatic time), die im Zuge der (darwinistischen) Evolution im großen Zeitraum von Jahrmilliarden (earth time) entstanden ist und immer mehr ausgefeilt wurde, sodaß heute ,,Lernen" auf zwei Zeitskalen stattfindet. Zuerst besaßen die Tiere nur fest verdrahtete Netze, dann U-Netze und irgendwann erfolgte der Sprung (oder eher der fließende Übergang) zu R-Netzen (und sogar zu S-Netzen). Einen solchen Übergang in einer (natürlich stark vereinfachten) Simulation festzustellen, weckte mein Interesse an Hebb-Netzen. Zuvor war es jedoch ratsam zu untersuchen, ob U-Netze überhaupt zu R-ähnlichen Netzen umfunktioniert (strukturiert) werden können, bevor man versucht, diese Aufgabe einem genetischen Algorithmus anzuvertrauen. Genau dies habe ich mit teilweisem Erfolg versucht:

Die (geschickte) Verschaltung von Hebb-Netzen zu R-ähnlichen Netzen, deren Details im folgenden näher ausgeführt werden. Der Ausspruch ,,Neurononale Netze programmiert man nicht, man konstruiert sie" spiegelt hier besonders deutlich die Verteilung des Arbeitsaufwandes wieder.

2 Reinforcement-Hebb-Netze ohne Hidden-Units

Eine einfache Klasse von Netzen sind Netze ohne Hidden-Units, die zwar nicht besonders leistungsfähig sind, aber sehr gut geeignet sind, Erfahrungen zu sammeln. Deshalb möchte ich diese Netzklasse hier näher untersuchen.

Bei solchen Netzen besteht nämlich kein prinzipieller Unterschied zwischen S- und U-Lernen. Beim S-Netz wird auf den Output der Output-Units gewartet und der Fehler der dabei gemacht wurde an die Output-Units angelegt und daraus die Gewichtsänderung berechnet (Backprop. entfällt bei 2 Layer Netzen). Bei U-Netzen mit nur sichtbaren Units (gleichzeitig Input- und Output-Units) werden beim Lernen verschiedene Muster angeboten und dadurch die Gewichte verändert. Anschließend werden nur noch Bruchstücke angelegt, deren Rest vom Netz rekonstuiert wird. Präsentiert man diesem U-Netz beim Lernvorgang ein Muster, dessen einer Teil dem Input des S-Netzes entspricht und dessen anderer Teil aus dem gewünschten dazugehörigen Output besteht und legt anschließend nur Teil 1 an, so wird das U-Netz Teil 2 rekonstruieren, ohne „zu wissen", daß wir dies als Output zu einem Input auffassen. Insofern ist die Unterscheidung in Output- und Input-Units nur Interpretationssache. Auf jeden Fall sind beide Algorithmen zur Gewichtsänderung in solchen Netzen äquivalent.

Ich verwende hier deterministische Schwellwert-Units mit Out- putbereich -1,+1, wobei ich häufig -1 als FALSE und +1 als TRUE interpretieren werde.

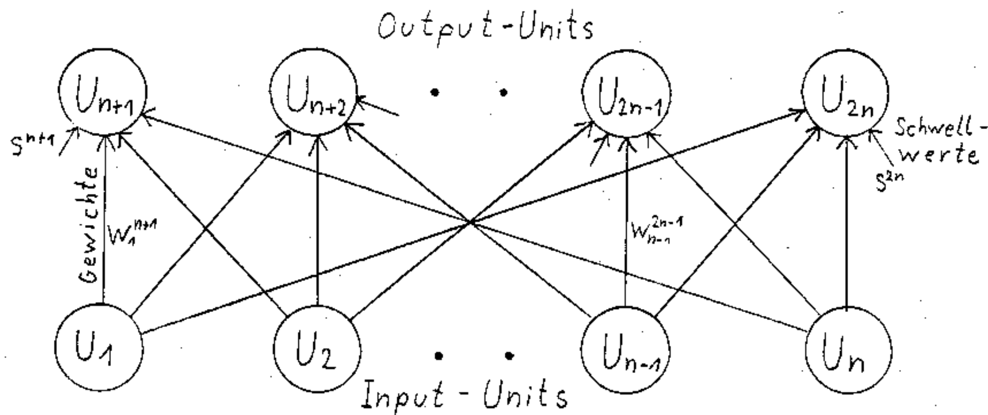

Abbildung 1: Zwei-Lagen-Netz ohne Hidden Units.

Grundkonstruktion

Das sichtlich einfachste Netz, welches man sich vorstellen kann, besteht nur aus je einer Unit im Input- und Output-Layer mit einem nach Hebb unsupervised zu lernenden Gewicht. Dies kann auch wie oben als S-Netz und sogar als R-Netz aufgefaßt werden, da

hier die R-Information gleich dem Betrag der S-Information ist. Mit Hilfe dieser Information läßt sich aber der 'richtige' Output nach Tabelle 1 berechnen. R=-1 soll Bestrafung, R=+1 Belohnung bedeuten.

gelieferter Output	-1	-1	+1	+1
R-Info.	-1	+1	-1	+1
gewünschter Output	+1	-1	-1	+1

Tabelle 1: Berechnung des Outputs aus R-Info.

Dies ist die XNOR-Funktion. D.h. erweitere ich mein Netz um einen weiteren Input (dem R-Input), verknüpfe diesen mit dem Output des ursprünglichen Netzes zu XNOR, so erhalte ich den gewünschten Output, also einen Superviser. Die XNOR-Funktion läßt sich durch ein kleines Netz realisieren. Was fange ich aber in einem Hebb-Netz mit einem Superviser an ? Ganz einfach: Lege ich diesen mit einem sehr starken Gewicht als weiteren Input an die (einzige) Output-Unit, so erzwinge ich den erwünschten Output an der Output-Unit. Diese Situation unterscheidet sich nicht von derjenigen, in der ich direkt Input & zugehörigen Output an beide Units angelegt hätte (bis auf eine zeitliche Verzögerung). Dies ist aber genau die Situation im Unsupervised Fall - Und das Netz lernt diese IO-Kombination.

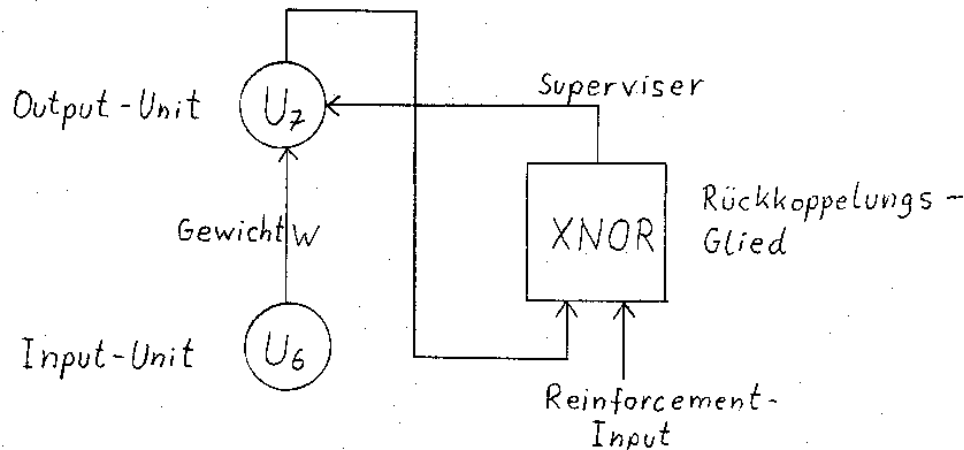

Abbildung 2: Erzeugung eines Supervisers (schematisch).

Hier sind einige Bemerkungen angebracht:

- Der neue R-Input verhält sich zwar so wie ein Reinforcement-Input bei R-Netzen, wird aber hier nicht durch einen mit gesondertem Algorithmus behandelten Input

realisiert, sondern fungiert wie jeder andere Input und erhält nur einen eigenen Namen und weist bei geeigneter Interpretation des Betrachters besondere Eigenschaften auf.

- Da das Netz nach einer Lernphase den gewünschten Output auch ohne R-Signal liefern soll, kann der durch XNOR erhaltene Output nicht direkt als Output des Netzes dienen. Außerdem verhält sich alles nur in diesem äußerst primitiven Netz so schön und übersichtlich.

Bild 3 stellt unser Netz nun komplett dar. Die Units U1 bis U5 bilden ein XNOR-Gatter, wie man sich leicht überzeugen kann und sind wie oben beschrieben mit der Output-Unit U7 verbunden.

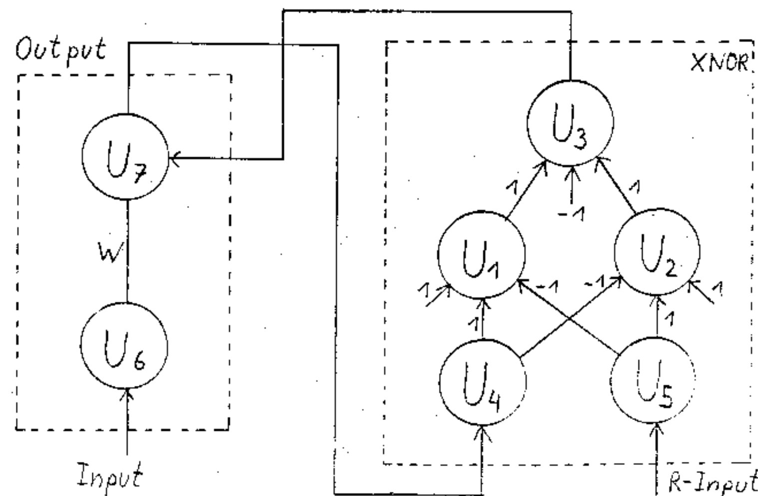

Abbildung 3: Einfachstes U-Netz mit R-Input.

Mit diesem Netz lassen sich zwei Funktionen lernen. Die Identität (Id) mit Gewicht größer Null bzw. die Negation (not) mit Gewicht kleiner Null. Überzeugen wir uns durch Analyse des zeitlichen Verhaltens, daß diese Funktionen bei geeignetem R-Signal tatsächlich auch gelernt werden (und nur in diesem einfachsten aller Beispiele ist dies überhaupt möglich). Nehmen wir an, das Netz soll ein 'not' lernen.

Der initiale Zustand der Units sei beliebig, am Eingang und am Ausgang liege eine 1. Dies ist nicht was wir wünschen, also legen wir am R-Eingang U5 ein negatives Reinforcement -1 an. Dieser wird nun mit dem Output von U7 XNOR verknüpft und gelangt nach 2 Zeitschritten (synchroner Update aller Units) nach U3 als eine -1. Ein kritischer Punkt ist, daß sich U7 in der Zwischenzeit geändert haben kann, z.B. auf -1. Dann wird aber das zeitlich korrespondierende R-Signal +1 angelegt (da dieser Output ja erwünscht ist), und dieses ergibt zusammen nach einer gewissen Verzögerung auch -1 an U3. D.h. in jedem

Zeitschritt ändert sich der Input des XNOR-Gatters, der Ausgang liefert jedoch korrekt immer -1. Daß dieses 'Pipeline'-Prinzip funktioniert, ist nicht selbstverständlich und gilt auch nur bei Net- zen, bei denen für je zwei Units JEDER Weg zwischen diesen die gleiche Länge (d.h. gleiche Zahl an zwischen-Unit) hat, also bei streng geschichteten Netzen, wie das für dieses XNOR der Fall ist.

Nach zwei Zeitschritten liegt also an U3 der gewünschte Output an, unabhängig von der Netzinitialisierung, und wird im nächsten Schritt durch das starke Gewicht auf die Output-Unit U7 übertragen. Nun haben wir einen stabilen (und gewünschten) Output erreicht. In den ersten 3 Zeitschritten wird das Gewicht w aufgrund des zufälligen Outputs durch die Hebbregel zufällig verändert, also im schlimmsten Fall 3* um einen bestimmten Wert vergrößert, im Mittel einfach gleich gelassen. In den nächsten stationären Schritten, solange der Input noch konstant bleibt, wird das Gewicht garantiert verkleinert. Der Fall eines negativen Inputs verläuft analog und bewirkt letztenendes auch eine Verkleinerung von w. Damit konvergiert w gegen einen negativen Wert, der durch die Form der Hebbregel beschränkt bleibt. Dies stellt aber eine Negation wie gewünscht dar. In der anschließenden Rekapitulation des Gelernten, in der der R-Input auf einen beliebigen Wert ≥ 0 gesetzt wird, wird durch die Rückkoppelung von U3 nach U7 der Output höchstens positiv, d.h. in gewünschter Weise beeinflußt. (darauf wird später noch eingegangen). Analog überzeugt man sich, daß bei umgekehrten R-Input wie erwünscht die Identität mittels positivem Gewicht w gelernt wird.

Dies ist das erste Hebb-Netz, welches zwei sich widerspechende Aufgaben lernen kann, einmal 'Id' und einmal 'not', in Abhängigkeit vom R-Signal.

Mehrere IO-Units

Vergrößern wir nun die Zahl der Input und Output-Units. Die Ersetzung einer Unit durch mehrere bedeutet einen qualitativen Sprung. Von mehreren zu noch mehr nur noch einen quantitativen, sodaß alles wesentliche schon an einem Netz mit je zwei Input- und Output-Units zu erkennen ist. Dazu kommen außerdem zwei zu lernende Schwellwerte (Bild 4).

Die Vermehrung der Input-Units bewirkt nur eine Verlangsamung der Lernrate. Für zwei Output-Units benötigen wir zwei XNOR-Rückkoppelungen. Da aber nur ein R-Signal zur Verfügung steht, das beiden zugeführt wird, können die an U10 und U11 erzeugten Werte nur noch im statistischen Mittel den gewünschten Output abgeben (positive Kovarianz; gleiches Phänomen wie beim Standard-R-Lernen). Damit wird eine direkte Verfolgung der Signale (mit der Absicht die Funktionsweise zu verstehen) unmöglich. Abhilfe schafft eine Tabelle, die für ein Beispiel angegeben ist.

U10 habe ein NAND, U11 ein OR zu lernen. Nur wenn beide Output-Units das richtige Ergebnis liefern sei R=1, ansonsten -1. Im allgemeinen wird das R-Signal den mittleren Fehlerabstand, normiert auf ein beliebiges Nullniveau darstellen, Feinheiten spielen für die Funktionstüchtigkeit nur eine untergeordnete Rolle. Legt man einen Input an, so gibt es 4 Output-Möglichkeiten des Netzes.

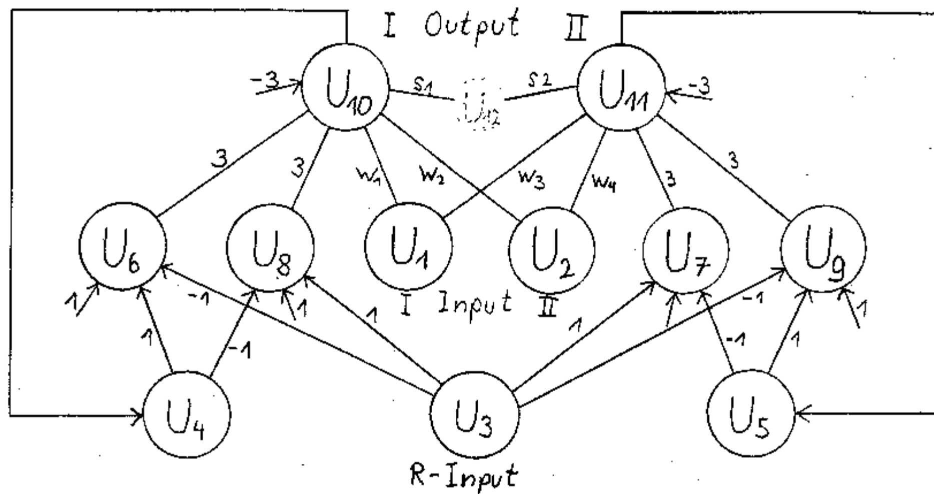

Abbildung 4: U-Netz mit R-Input, 2 Input-, 2 Output-Units.

1. Beide Outputs sind (relativ zum gewünschten Output) falsch:
   Das R-Signal wird -1 und nach 3 Zeitschritten zeigen die Output-Units gegenteiliges Verhalten, d.h. das gewünschte. Nun wird R=1 gesetzt und das System bleibt stabil.

2. Beide Outputs entsprechen dem gewünschten Output:
   Das R-Signal wird +1 und der Zustand bleibt stabil.

3/4. Eine Unit liefert das richtige, die andere ein falsches Ergebnis:
   Das auf -1 gesetzte R-Signal bewirkt eine Invertierung des Outputs. Dies führt aber wieder in den Fall 3/4. R bleibt -1 und die Outputs schwingen periodisch zwischen +1 und -1.

Im Fall 1 und 2 liefern die Output-Units den richtigen Wert, im Fall 3 und 4 zu 50% den richtigen und 50% den falschen Wert. Die Gewichtsänderung ist im Fall 3 und 4 also 0, da sich positive und negative Kovarianzen aufheben. D.h. nur Fall 1 und 2, die beide (spätestens nach einer Verzögerung) den gewünschten Output liefern, tragen zum Lernen bei. Bei geringsten Unregelmäßigkeiten im Update der Units im Fall 3/4 (also speziell bei asynchronem Update) geht dieser oszillierende in den stabilen Zustand 1/2 über. Somit ist die Abschätzung, daß in 50% der Fälle, also 50% der Zeit, effektiv gelernt wird, im allg. zu schlecht.

In Tabelle 2 ist zu jeder stabilen IO-Kombination die zugehörige und in der letzten Spalte die mittlere Gewichtsänderung angegeben. Die Zahlenwerte bedeuten, daß bei unendlich langer Verweilzeit in diesem Zustand die Gewichte gegen ± 1 konvergieren würden, bei zyklischer Repräsentation der Inputs gegen $\tilde{n} \ll$.

Input I	-1	-1	+1	+1	
Input II	-1	+1	-1	+1	
Output I	+1	+1	+1	-1	W
Output II	-1	+1	+1	+1	
Gewicht W11	-1	-1	+1	-1	$- \ll$
Gewicht W21	-1	+1	-1	-1	$- \ll$
Schwellwert S1	+1	+1	+1	-1	$+ \ll$
Gewicht W12	+1	-1	+1	+1	$+ \ll$
Gewicht W22	+1	+1	-1	+1	$+ \ll$
Schwellwert S2	-1	+1	+1	+1	$+ \ll$

Tabelle 2: Gewichtsänderung im 2-2 Netz.

Wie man sich leicht überzeugen kann sind die zur korrekten Funktion des Netzes notwendigen Gewichte und Schwellwerte gerade die in der Tabelle angegebenen. Das Netz lernt also korrekt. Dies gilt für alle Paare von boolschen Funktionen, solange diese nicht XOR oder XNOR sind, welche prinzipiell nicht durch ein Netz ohne Hidden-Units dargestellt werden können (Siehe Kapitel 3)

Größeres Beipiel

Wie oben schon erwähnt, läßt sich die Argumentation auch auf größere Netze übertragen. Um ganz sicher zu gehen, ist es am besten einfach ein größeres Netz zu konstruieren und zu testen. Dabei nimmt die Lernzeit allerdings schneller mit der Anzahl der Output-Units zu (exponentiell ?) als dies theoretisch notwendig wäre (linear !). Auch wird im allg. nicht mehr fehlerfrei gelernt, alles in allem ergeben sich aber durchaus zufriedenstellende Ergebnisse (wenn man die Einfachheit der Mittel berücksichtigt).

Anmerkungen

- Je mehr Output-Units das Netz enthält, desto chaotischer scheint sich das Netz zu verhalten. Dieses Verhalten ist bestimmt nicht negativ, da durch diese quasi-statistische Ver- haltensweise, die wesentlich durch die Antisymmetrie der Gewichte oder konkreter durch die Rückkoppelung bewirkt wird, der Lösungsraum abgesucht wird. Hier ist die Ähnlichkeit zu Boltzmann-Netzen auffällig, auf die in Kap. 3 noch näher eingegangen wird.

- Bisher wurden nur Schwellwert-Units betrachtet und somit nur Outputs von ± 1. Bei Sigmoiden Units, die auch Zwischenwerte liefern, machte sich folgender Effekt bemerkbar:
  1.Das quasi-statistische Verhalten verstärkte sich, da die XNOR-Rückkoppelung bei

kontinuierlichen Input-Werten eher zum Schwingen neigt.
2.Wurde eine Funktion gut gelernt, so verstärkten sich die Gewichte auch noch in der Remind-Phase, d.h. ohne R-Input, durch die bestehende Rückkoppelung. Bei schlecht gelernten Funktionen trat der gegenteilige Effekt ein, und eine partiell gelernte Funktion wurde wieder vergessen.

- Da die Konstruktion dieser R-lernenden Hebb-Netze nur eine Voruntersuchung war, die zeigen sollte ob dies prinzipiell möglich ist, wurde hier der einfachste Fall eines expliziten R-Signals untersucht. Interessant wird es erst, Netze zu konstruieren (von genetischen Algorithmen konstruieren zu lassen), bei denen das R-Signal implizit in der restlichen Input-Information codiert ist und durch das Netz entschlüsselt wird. Da diese Decodierung jedoch möglich ist, konnte ich mich der Einfachheit halber auf den expliziten Fall beschränken.

- Die vielen festen Gewichte (in den XNOR) muß man sich als Gewichte mit sehr kleiner (null) Flexibilität denken. Somit wird der Rahmen der Hebb-Netze nicht verlassen. Diese Verbindungen wirken aber dennoch etwas künstlich und störend. Versuche auch die Flexibilität dieser Units ungleich Null zu setzen, haben alle zur Zerstörung der XNOR-Funktion und damit der Funktionsfähigkeit des Netzes geführt. Wahrscheinlich sind solche starren Gewichte auch notwendig für komplexer gestaltete Netze. Genetisch erzeugte Netze werden vermutlich ein kontinuierliches Spektrum an Gewichten verschiedenster Flexibilität aufweisen (natürlich nur wenn diese Option zur Verfügung steht).

3 Konstruktionsversuche von RH-Netzen mit Hidden-Units

Um es gleich vorwegzunehmen - der Versuch, die Idee aus Kapitel 2 auf Netze mit Hidden-Units, um die es in diesem Kapitel geht, zu übertragen, ist gescheitert. Welche Probleme dabei auftraten und die Versuche, diese zu lösen, sollen im folgenden erörtert werden. Der Einfachheit halber beziehen sich die meisten nun folgenden Betrachtungen auf Netze mit nur einem Hidden-Layer (ähnlich Bild 5). Es wurden alle Experimente mit

- synchronem / asynchronem Unit-Update
- Schwellwert / Sigmoid-Units
- direkter / akkumulierter Gewichtsänderung

durchgeführt.

Erweiterte Klasse von Netzen

Stellt man sich das Input-Layer der bisher betrachteten 2-Layer-Netze als Hidden-Layer-Schicht vor, die von anderen Units durch Verbindungen mit festen Gewichten gespeist werden, so ändert dies natürlich weder am Lern-Prinzip, noch am Rückkoppelungsmechanismus etwas, d.h. auch solche Netze können R-Lernen. Anders interpretiert stellen diese Units Hidden-Units mit von vornherein durch die Netzstruktur festgelegter Bedeutung dar, und stellen somit keine 'echten' Hidden-Units dar. Man kann theoretisch durch eine solche feste Zwischencodierung zwar alle Probleme auf lineare Probleme reduzieren und damit auf die hier beschriebene Weise lösen, doch nimmt die Zahl der benötigten Hidden-Units exponentiell (also unbrauchbar) mit der Zahl der Input-Units zu. Für zwei Input-Units benötigt man $2^2 = 4$ Hidden-Units. Somit hat ein Netz, das neben den 14 Bool-Funktionen auch XOR und XNOR lernen kann, folgende Gestalt.

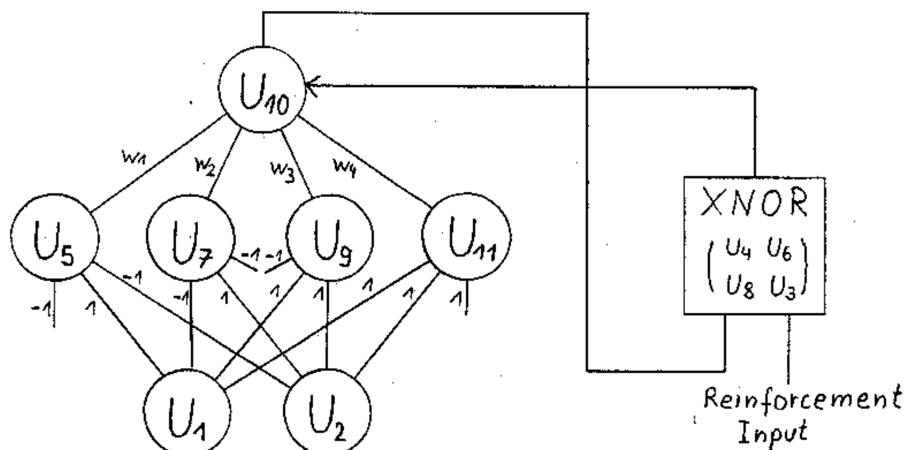

Abbildung 5: Netz, das alle 16 bool.-Funtionen lernen kann.

Diese Methode ist für große Netze aber absolut unbrauchbar und führt an der Idee der Hidden-Units total vorbei.

Rückkoppelungsverzögerung bei Hidden-Units

Eine 'echte' Hidden-Unit, deren Bedeutung erst gelernt wird, besitzt mindestens eine flexible Verbindung inputseitig. Die naheliegendste Idee ist, auch solche Units mit einem XNOR-Rückkoppelungsglied zu verbinden. Die Argumentation könnte etwa wie folgt aussehen:

Jede Hidden-Unit trägt zu einem gewissen Prozentsatz zum Output bei, d.h. es wird eine positive Korrelation zwischen der Korrektheit der Hidden-Unit und der Output-Unit und somit auch zum R-Signal bestehen.

Folgendes muß jedoch im Fall des synchronen Updates unbedingt beachtet werden: Ein Output einer Hidden-Unit beeinflußt erst im nächsten (bzw. nächsten n) Schritt den Netz-Output und damit das dazugehörige R-Signal mit dem er durch die XNOR-Rückkoppelung verbunden wird. Der Output der Hidden-Unit muß somit einen (bzw. mehrere) Schritte 'aufbewahrt werden' bevor er mit dem R-Signal verknüpft werden kann. Diese Aufbewahrung kann durch eine zwischen Hidden-Unit und XNOR-Eingang geschaltete Unit bewerkstelligt werden. (Das Problem der zeitlichen Nicht-lokalität haben viele Netze). Allen Versuchen zum Trotz stellte sich nicht einmal bei einfachsten nicht-linearen Problemen (XOR) ein Lernerfolg ein.

Schwacher Feedback für partiell korrekte Funktionen

Da auch eine Hidden-Unit mit XNOR-Rückkoppelung versucht die gestellte Aufgabe 100% korrekt zu lösen, dies aber aufgrund der Art der Aufgabe nicht möglich ist, sollte sie diese wenigstens partiell korrekt lernen, sodaß in der nächsten Schicht aus den partiell korrekt gelösten Aufgaben der Hidden-Units die exakte Lösung in der 2. Schicht linearkombiniert werden kann. Nun stellte sich aber bei 2-layer Netzen heraus, daß diese bei dem Versuch ein XOR zu lernen, lieber gar nichts lernten, als ein OR, NAND oder etwas ähnlichem, welche aber typische Kandidaten für eine interne Repräsentation/Codierung und anschließender Linearkombination zu einem XOR sind. Abhilfe schafft hier, die Zwangsrückkoppelung von U3 nach U7 in Bild 1 und analog in allen anderen abzuschwächen (Werte kleiner 1), um so dem Netz die Möglichkeit zu geben, entgegen seiner Vorbestimmung zu reagieren. Tatsächlich lernten die Netze weiterhin die auch theoretisch erlernbaren Aufgaben; bei nichtlinearen Aufgaben wurden Approximationen gelernt. Leider führte auch diese Modifikation in Multilayer-Netzen nicht zum Erfolg.

Feedback zu jedem 2. Layer

Ein Problem war folgendes: Da sowohl Output-Layer als auch Hidden-Layer bei negativem Reinforcement zu inversem Verhalten gezwungen werden, ändert sich am Produkt ihrer Aktivationen $((-a)\times(-b)=a\times b)$ und damit an der Richtung, in der die Gewichte geändert werden, nichts. Nur zwischen Input- und erstem Hidden- Layer ändert sich das Vorzeichen und der Rückkoppelungsmechanismus 'greift'. Dem könnte Abhilfe geschaffen werden, indem nur jede 2. Netzschicht rückgekoppelt wird und damit die Lernrichtung bei negativem Reinforcement in jedem Layer umgekehrt wird. In einem 3-Layer-Netz bedeutet dies, nur die mittlere Schicht rückzukoppeln. Wie auch bei allen anderen Modifikationen brachte dies zwar eine konvergenzfördernde Wirkung, aber keineswegs eine befriedigende.

Symmetrische Gewichte

Ersetzt man im Netz die Gewichte durch symmetrische, d.h. fügt Gewichte gleicher Stärke und Flexibilität in entgegengesetzter Richtung hinzu, und betrachtet sowohl Input- als

auch Output-Layer als Inputs, so hat man ein unsupervised lernendes Netz mit Hidden-Units. Hätten sich nicht schon bei dieser Lernmethode unlösbare Schwierigkeiten ergeben, wäre der Versuch, nun die Information am Output-Layer durch eine XNOR-Rückkoppelung mit R-Signal zu approximieren, einer der erfolgsversprechendsten gewesen. Da mir aber kein funktionsfähiges Hebb-Netz mit Hidden-Units bekannt war, konnte ich die Fehlerursache nicht finden.

Boltzmann-Netze

Das passendste, das ich finden konnte, waren Boltzmann-Netze, die von der Idee der Hebb-Netze nicht allzuweit entfernt sind. Die Gewichtsänderung ist im wesentlichen wie bei Hebb-Netzen proportional zur Korrelation benachbarter Unit-Aktivitäten. Die Unterschiede zu Hebb-Netzen sind folgende:

- Der Output ± 1 wird statistisch in Abhängikeit der Aktivation und einer Größe, Temperatur genannt, berechnet.
- Das Netz benötigt eine 'Vergeßphase', in der in Abwesenheit externer Einflüsse (Inputs), die Gewichte mit der inversen Lernregel verändert werden, also in entgegengesetzter Richtung.

Da aber beide Unterschiede über den Rahmen der üblichen Hebb-Netze hinausgehen und mir keine funktionsäquivalenten Simulationen innerhalb dieses Rahmens eingefallen sind, führte jene Entdeckung nur zur Erkenntnis, daß der Rahmen (Hebb-Netze) vielleicht enger als notwendig gesteckt war (Siehe Kap. 4).

Einfluß der Musterrepräsentationshäufigkeit auf die Gewichte

Beim Versuch, die verschiedenen Inputmuster nicht mehr gleichhäufig zu präsentieren, zeigte sich eine auffällige Abhängigkeit der Gewichte hiervon. Diese an sich negative Erscheinung, die auch bei vielen anderen Netzen zu finden ist, konnte an dieser Stelle gewinnbringend eingesetzt werden. Präsentierte man dem Netz Muster, die es gut gelernt hatte, seltener, noch unzureichend gelernte entsprechend häufiger, so ließ sich der Lernvorgang bei 2-Layer Netzen aus Kap.2 beschleunigen. Bei einem 3-Layer-Netz mit 2 Hidden Units ließ sich unter anderem folgendes Problem lösen: Versucht man einem 3-Layer-Netz bei gleichverteilter Präsentationshäufigkeit ein XOR zu lernen, so ist der Output zu 50% 1 und zu 50% -1, d.h. der Schwellwert der Output-Units wird gegen 0 konvergieren. Mit einem solchen Schwellwert ist es aber unmöglich ein XOR zu lernen. Legt man die Muster aber in oben beschriebener Weise an, so konvergiert zumindest der Schwellwert gegen den benötigten Wert (abhängig vom Verhalten der Hidden-Units).

4 Anmerkungen / Ausblicke

Man kann sich fragen, woran es wohl gelegen haben kann, daß alle Konstruktionsversuche von RH-Netzen mit Hidden-Units gescheitert sind. Liest man Kapitel 3 sorgfältig, so stellt man fest, daß an einigen Stellen erfolgversprechende Versuche hätten durchgeführt werden können, wenn man die Beschränkung auf Hebb-Netze aufgehoben hätte. Doch stellt sich dann die Frage, ,,wie weit man geht". Schließlich ist es kein Problem mit 'beliebigen' Units (nämlich R-Units) R-Netze zu konstruieren. Aus dieser Perspektive war ein eindeutig definierter Rahmen für die Charakteristika der Units durchaus sinnvoll. Doch scheint dieser Rahmen zu eng gewählt worden zu sein, aber jede Erweiterung 'im nachhinein' gekünstelt.

Konzentriert man sich dagegen auf die Idee des impliziten Reinforcement (Erklärung siehe Einleitung), so kann man, da dieses Problem meines Wissens noch niemand untersucht hat, auf einen abgrenzenden Rahmen verzichten. Läßt man gleich einen genetischen Algorithmus an dieses Problem, ohne Voruntersuchungen ähnlich zu meinen anzustellen, dann ist eine erfolgreiche Evolution eines Netzes mit implizitem R-Signal auch dann noch beachtlich, wenn man beliebige Units zuläßt. Auch die Verwendung von verschiedenartigsten Units innerhalb eines Netzes könnte für die Evolution von Vorteil sein.

Insgesamt glaube ich, daß mit den heutigen Erkenntnissen und Mitteln, eine solche 'Evolution von (implizitem) Reinforcement', nachgebildet (simuliert) werden kann. Dies wäre auch ein experimenteller Beweis dafür, daß evolutionäre Prozesse auf einer Langzeitskala Lernverfahren auf einer Kurzzeitskala hervorbringen können.

5 Literaturverzeichnis

Literatur zum Fopra

[1] Williams R.J.; 1988 Toward a theory of reinforcement learning; Techn. Report NU-CCS-88-3, College of Computer Science, Northeastern Univ. (Boston)

[2] Gerhard Weiss: 1990 Combining neuronal and evolutionary learning; Techn. Report FKI-132-90; TU-München; weissg@tumult.informatik.tu-muenchen.de

[3] Hinton G.E & Sejnowski T.J.; 1986; Learning and relearning in Boltzmann machines; IN: [5], Chapter 7, pp. 282-317

[4] Schulten K.; 1986 Associative Recognition and Storage in a Model Network of Physiological Neurons. IN: Biological Cybernetics 54,319-335 1986; Springer-Verlag

Allgemeine Literaturhinweise

[5] Rumelhart D.E. & McClelland J.L.; 1986; Parallel distributed processing vol 1 & 2. Book/MIT Press

[6] Goldberg D.E.; 1989; Genetic algorithms in search, optimization machine learning; Addison-Wesley

6 Programmlisting

```
{*******************************************************}
{*    Hebb-Net-Simulation    (c) by Marcus Hutter 25.6.90   *}
{*******************************************************}

PROGRAM HebbNet;

{---------------------}
{ Vars, Consts, Types }
{---------------------}
USES Crt;

CONST
  MaxUNum = 30;              { Max. Unit-Number }
  MaxCNum = MaxUNum*MaxUNum; { Max. Connection-Number }
  ESC     = #27;
  none    = #0;

  synchron = true;           { Synchron Update of Units }
  rpres    = false;          { random pattern-presentation }
  lr       = 0.001;          { lernrate }
  ReInOff  = 0;              { Reinf-Input in remind-mode }
  scl      = 5;              { scaling factor }

TYPE
  TUnitId =  1..MaxUNum;
  TUnit0Id = 0..MaxUNum;
  TUnit = record activ:  real; { Unit-activation }
                 output: real; { Unit-output }
                 kind:   Byte; { Unit-kind (not used) }
          end;
  TUnits = array[TUnit0Id] of TUnit;

  TConn = record weight: real; { Weight }
                 elig:   real; { Connection-eligibility }
                 kind:   Byte; { (not used) }
          end;
  TConns = array[TUnit0Id,TUnitId] of TConn;
  { Weights to Unit 0 are Offsets }

VAR
  Conns:   TConns;             { all connections }
  Units:   TUnits;             { all units }
  XInp:    array[TUnitId] of real; { external net-Input }
  WSt:     Integer;            { External World-Status }
  LernF:   boolean;            { Lern-Flag }
  RE:      real;               { middle Reinforcement }
  c:       char;
  dev:     Text;               { Output-Device }
```

```
{$DEFINE Example4}             { Compile Example }

{-------------------}
{ Special Functions }
{-------------------}
FUNCTION sign(x:real):integer;
  begin
    if x>0 then sign:=+1 else
    if x<0 then sign:=-1 else sign:=0
  end;

FUNCTION bound(x,l,u: real):real;
  begin
    if x>u then bound:=u else
    if x<l then bound:=l else
    bound:=x;
  end;

FUNCTION actf(x: real):real;
{ Activation-Function of Units }
  begin actf:=bound(x*scl,-1,+1) end;

{---------------}
{$IFDEF Example1 }
{---------------}
{ Hebb-Net to lern 14 Bool-Functions }
CONST
  PSTime = 15; { Pattern-Show-Time }
  PNum = 4;    { Pattern-Number }
  PLen = 2;    { Pattern-Lenght }
  OLen = 1;    { Output-Length }
  ReIn = 5;    { ReinforcementInput-UnitId }
  UnitNum = 8; { Unit Number (here constant) }

TYPE
  TPatN = 1..PNum;  { Pattern-Numbers }

CONST   { Table of Input-Units }
  ITab: array[1..PLen] of TUnitId = (6,8); { Input-UnitIds }
  OTab: array[1..OLen] of TUnitId = (7);   { Output-UnitIds }
  IPat: array[TPatN,1..PLen] of integer =  { Input-Patterns }
   ((-1,-1 ),(-1,+1 ),(+1,-1 ),(+1,+1) );
  OPat: array[TPatN,1..OLen] of integer =  { Output-Patterns }
   (  (-1)  ,  (+1)  ,  (+1)  ,  (+1)  );  { Or-Function }
  ConW: array[0..UnitNum,1..UnitNum] of real = { Weight-Array }
   ( ( 1, 1,-1, 0, 0, 0, 0, 0 ),   { OffSets }
     ( 0, 0, 1, 0, 0, 0, 0, 0 ),   { Conn. from 1 }
     ( 0, 0, 1, 0, 0, 0, 0, 0 ),
     ( 0, 0, 0, 0, 0, 0, 1, 0 ),
```

```
    ( 1,-1, 0, 0, 0, 0, 0, 0 ),
    (-1, 1, 0, 0, 0, 0, 0, 0 ),
    ( 0, 0, 0, 0, 0, 0, 0, 0 ),
    ( 0, 0, 0, 2, 0, 0, 0, 0 ),
    ( 0, 0, 0, 0, 0, 0, 0, 0 ) );
  ConE: array[0..UnitNum,1..UnitNum] of real = { Elig.-Array }
   ( ( 0, 0, 0, 0, 0, 0, 2, 0 ),   { OffSets }
    ( 0, 0, 0, 0, 0, 0, 0, 0 ),   { Conn. from 1 }
    ( 0, 0, 0, 0, 0, 0, 0, 0 ),
    ( 0, 0, 0, 0, 0, 0, 0, 0 ),
    ( 0, 0, 0, 0, 0, 0, 0, 0 ),      { ... }
    ( 0, 0, 0, 0, 0, 0, 0, 0 ),
    ( 0, 0, 0, 0, 0, 0, 2, 0 ),
    ( 0, 0, 0, 0, 0, 0, 0, 0 ),
    ( 0, 0, 0, 0, 0, 0, 2, 0 ) );
{$ENDIF}

{----------------}
{$IFDEF Example3 }
{----------------}
{ Hebb-Net to lern 16 Bool-Functions incl. XOR (1 lernlayer) }CONST
  PSTime = 15; { Pattern-Show-Time }
  PNum = 4;    { Pattern-Number }
  PLen = 2;    { Pattern-Lenght }
  OLen = 1;    { Output-Length }
  ReIn = 3;    { ReinforcementInput-UnitId }
  UnitNum = 12;{ Unit Number (here constant) }

TYPE
  TPatN = 1..PNum;  { Pattern-Numbers }

CONST   { Table of Input-Units }
  ITab: array[1..PLen] of TUnitId = (1,2);   { Input-UnitIds }
  OTab: array[1..OLen] of TUnitId = (10);    { Output-UnitIds }
  IPat: array[TPatN,1..PLen] of integer = { Input-Patterns }
   ((-1,-1),(-1,+1),(+1,-1),(+1,+1) );
  OPat: array[TPatN,1..OLen] of integer = { Output-Patterns }
   ( (-1)  , (-1)  , (-1)  , (+1)   );    { Or/And-Function }
  ConW: array[0..UnitNum,1..UnitNum] of real =
  { to_1__2__3__4__5__6__7__8__9_10_11_12_ }
   ( ( 0, 0, 0, 0,-1,-1,-1,-1,-1,-3, 1, 1 ),   { OffSets }
    ( 0, 0, 0, 0, 1, 0,-1, 0, 1, 0, 1, 0 ),   { Conn. from 1 }
    ( 0, 0, 0, 0,-1, 0, 1, 0, 1, 0, 1, 0 ),
    ( 0, 0, 0, 0, 0,-1, 0, 1, 0, 0, 0, 0 ),      { ... }
    ( 0, 0, 0, 0, 0, 1, 0,-1, 0, 0, 0, 0 ),
    ( 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ),
    ( 0, 0, 0, 0, 0, 0, 0, 0, 0,-3, 0, 0 ),
    ( 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ),
    ( 0, 0, 0, 0, 0, 0, 0, 0, 0,-3, 0, 0 ),
    ( 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ),
```

```
    ( 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0 ),
    ( 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ),
    ( 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ) );
  ConE: array[0..UnitNum,1..UnitNum] of real =
   ( ( 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ),   { OffSets }
    ( 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ),   { Conn. from 1 }
    ( 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ),
    ( 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ),
    ( 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ),      { ... }
    ( 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0 ),
    ( 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ),
    ( 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0 ),
    ( 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ),
    ( 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0 ),
    ( 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ),
    ( 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0 ),
    ( 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ) );
{$ENDIF}

{----------------}
{$IFDEF Example4 }
{----------------}
{ Hebb-Net to lern 14*14 Bool-Functions }
CONST
  PSTime = 20; { Pattern-Show-Time }
  PNum = 4;    { Pattern-Number }
  PLen = 2;    { Pattern-Lenght }
  OLen = 2;    { Output-Length }
  ReIn = 3;    { ReinforcementInput-UnitId }
  UnitNum = 12;{ Unit Number (here constant) }

TYPE  TPatN = 1..PNum;  { Pattern-Numbers }

CONST   { Table of Input-Units }
  ITab: array[1..PLen] of TUnitId = (1,2);   { Input-UnitIds }
  OTab: array[1..OLen] of TUnitId = (10,11); { Output-UnitIds }
  IPat: array[TPatN,1..PLen] of integer = { Input-Patterns }
   ((-1,-1),(-1,+1),(+1,-1),(+1,+1) );
  OPat: array[TPatN,1..OLen] of integer = { Output-Patterns }
   ((-1,-1),(+1,-1),(+1,-1),(+1,+1) );    { Or/And-Function }
  ConW: array[0..UnitNum,1..UnitNum] of real =
  { to_1__2__3__4__5__6__7__8__9_10_11_12_ }
   ( ( 0, 0, 0, 0, 0, 1, 1, 1, 1,-3,-3, 1 ),   { OffSets }
    ( 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ),   { Conn. from 1 }
    ( 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ),
    ( 0, 0, 0, 0, 0,-1,-1, 1, 1, 0, 0, 0 ),      { ... }
    ( 0, 0, 0, 0, 0, 1, 0,-1, 0, 0, 0, 0 ),
    ( 0, 0, 0, 0, 0, 0, 1, 0,-1, 0, 0, 0 ),
    ( 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0 ),
    ( 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0 ),
```

```
     ( 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0 ),
     ( 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0 ),
     ( 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0 ),
     ( 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0 ),
     ( 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ) );
  ConE: array[0..UnitNum,1..UnitNum] of real =
   ( ( 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ),   { OffSets }
     ( 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 3, 0 ),   { Conn. from 1 }
     ( 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 3, 0 ),
     ( 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ),
     ( 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ),      { ... }
     ( 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ),
     ( 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ),
     ( 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ),
     ( 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ),
     ( 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ),
     ( 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ),
     ( 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ),
     ( 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 3, 0 ) );
{$ENDIF}

{----------------}
{$IFDEF Example5 }
{----------------}
{ Hebb-Net to lern Room-Inventory-Relation }
{ Test !! }
CONST
  PSTime = 20; { Pattern-Show-Time }
  PNum = 5;    { Pattern-Number }
  PLen = 9;    { Pattern-Lenght }
  OLen = 5;    { Output-Length }
  ReIn = 30;   { ReinforcementInput-UnitId }
  UnitNum = 30;{ Unit Number (here constant) }

TYPE
  TPatN = 1..PNum;  { Pattern-Numbers }

CONST   { Table of Input-Units }
  ITab: array[1..PLen] of TUnitId =       { Input-UnitIds }
    (1,2,3,4,5,6,7,8,9);
  OTab: array[1..OLen] of TUnitId =       { Output-UnitIds }
    (10,11,12,13,14);
  IPat: array[TPatN,1..PLen] of integer = { Input-Patterns }
   ((-1,-1,+1,-1,-1,+1,+1,-1,+1),
    (-1,+1,+1,+1,-1,+1,-1,-1,+1),
    (-1,-1,+1,-1,-1,-1,-1,+1,-1),
    (+1,-1,-1,+1,-1,+1,+1,-1,-1),
    (-1,+1,-1,-1,+1,+1,-1,-1,+1));
  OPat: array[TPatN,1..OLen] of integer = { Output-Patterns }
   ( (+1,-1,-1,-1,-1),
```

```
     (-1,+1,-1,-1,-1),
     (-1,-1,+1,-1,-1),
     (-1,-1,-1,+1,-1),
     (-1,-1,-1,-1,+1) );

{ ConW and ConE are defined in InitExample }

{$ENDIF}

{----------------------}
{ Forward-Declarations }
{----------------------}
PROCEDURE chdo(ch:char); forward;
FUNCTION event:char;     forward;

{-------------}
{ IO-Features }
{-------------}
PROCEDURE wrConns;
{ Write Connections }
  var c: TUnit0Id;
      u: TUnitId;
  begin writeln(dev,'Connection-Strength:');
    for c:=0 to UnitNum do begin
      for u:= 1 to UnitNum do
        write(dev,100*Conns[c,u].weight:5:0);
      writeln(dev);
    end;
  end;

PROCEDURE wrUnits;
{ Write Activation of Units (in one line) }
  var u: TUnitId;
  begin
    for u:=1 to UnitNum do
      write(dev,100*Units[u].activ:5:0);
    writeln(dev);
  end;

PROCEDURE SaveStatus;
{ Save Status (Connection-Weights) to Disk }
  var sdev: file;
  begin
    writeln(dev,'Saving Status as Status.Dat');
    assign(sdev,'D:\pascal\myprogs\status.dat');
    rewrite(sdev,1);
    blockwrite(sdev,Conns,SizeOf(Conns));
    close(sdev);
  end;
```

```
PROCEDURE LoadStatus;
{ Load Status (Connection-Weights) from Disk }
  var sdev: file;
  begin
    writeln(dev,'Loading Status from Status.Dat');
    assign(sdev,'D:\pascal\myprogs\status.dat');
    reset(sdev,1);
    blockread(sdev,Conns,SizeOf(Conns));
    close(sdev);
  end;

PROCEDURE wrStatus;
{ Write Status (Connection-Weights) }
  begin wrConns end;

{----------------}
{ Initialisation }
{----------------}
PROCEDURE InitExample;
{ Example-Specific-Initialistion-Routine }
  var c: TUnit0Id;
      u: TUnitId;
  begin
  {$IFDEF Example5 }
  { Create Net with Input-Units 1..9, Output-Units 10..14,
    Feedback-Units 14..29, R-Input-Unit 30 }
    FillChar(Conns,SizeOf(Conns),0);
    for c:=1 to OLen do begin
      for u:=1 to PLen do Conns[u,PLen+c].elig:=1;
      Conns[0,PLen+c].weight:=-5;
      Conns[PLen+OLen-2+3*c,PLen+c].weight:=-5;
      Conns[PLen+OLen-1+3*c,PLen+c].weight:=-5;
      Conns[0,PLen+OLen-2+3*c].weight:=-1;
      Conns[0,PLen+OLen-1+3*c].weight:=-1;
      Conns[PLen+OLen+3*c,PLen+OLen-2+3*c].weight:=+1;
      Conns[PLen+OLen+3*c,PLen+OLen-1+3*c].weight:=-1;
      Conns[ReIn,PLen+OLen-2+3*c].weight:=-1;
      Conns[ReIn,PLen+OLen-1+3*c].weight:=+1;
      Conns[PLen+c,PLen+OLen+3*c].weight:=1;
      Conns[PLen+OLen+3*c,PLen+OLen+3*c].weight:=1;
    end;
  {$ELSE}
  { Copy Net from Example to Conns-Array }
    for c:=0 to UnitNum do
      for u:=1 to UnitNum do begin
        Conns[c,u].weight:=ConW[c,u];
        Conns[c,u].elig:=ConE[c,u];
      end;
  {$ENDIF}
  end;
```

```
PROCEDURE InitConns;
{ Initialize weights with 0 and eligibility with 1 }
  var c: TUnit0Id;
      u: TUnitId;
  begin
    for c:=0 to MaxUNum do
      for u:=1 to MaxUNum do begin
        Conns[c,u].weight:=0;
        Conns[c,u].elig:=1;
      end;
  end;

PROCEDURE InitUnits;
{ Random-Initialisation of Unit-Output }
  var u:TUnitId;
  begin    FillChar(Units,SizeOf(Units),0);
    for u:=1 to MaxUNum do  { Random-Init }
      Units[u].output:=integer(random(3))-1;
    Units[0].output:=1;
    Units[0].activ:=1;
  end;

PROCEDURE InitNet;
{ Initialize Net }
  begin
    InitConns;
    InitUnits;
    FillChar(XInp,SizeOf(XInp),0);
    InitExample;
    RE:=OLen;
  end;

{--------------}
{ Net-Dynamics }
{--------------}
PROCEDURE EvalUnit(u: TUnitId);
{ Calculate new Unit-Activation with old Unit-Output }
  var s: real;
      c: TUnit0Id;
  begin
    s:=XInp[u];
    for c:=0 to UnitNum do
      s:=s + Units[c].output * Conns[c,u].weight;
    Units[u].activ:=s;
  end;

PROCEDURE EvalConn(c:TUnit0Id; u: TUnitId);
{ Change Conns[c,u].weight with Hebb-Rule }
  var change: real;
```

```
  begin
    if Conns[c,u].elig<>0 then begin
      change:=bound(Units[u].activ,-1,1) *
              bound(Units[c].activ,-1,1);
      Conns[c,u].weight:=
        bound( (1-lr)*Conns[c,u].weight
               + lr*change*Conns[c,u].elig ,-1,+1);
    end;
  end;

PROCEDURE UpdUOut(u: TUnitId);
{ Update Unit-Output (Activation --> Output) }
  begin Units[u].output:=actf(Units[u].activ) end;

PROCEDURE SetXInp;
{ Set external Input (Pattern[WSt] & Reinf.) ,
  Calculate Reinforcement-Signal and expectation of R. }
  var n: integer;
      R: real;
  begin
    FillChar(XInp,SizeOf(XInp),0); { not needed }
    XInp[ReIn]:=ReInOff;
    for n:=1 to PLen do XInp[ITab[n]]:=IPat[WSt,n];
    if LernF then begin  { Set Reinforce-Input }
      R:=0;
      for n:=1 to OLen do R:=R +
        Abs(Units[OTab[n]].output-OPat[WSt,n]);
      XInp[ReIn]:=OLen-R;      RE:=RE+0.002*(R-RE);
    end;
  end;

PROCEDURE TimeStep;
{ One Simulation-Step: SetXInp -> EvalUnit --> (EvalConn) }
  var c: TUnit0Id;
      u,v: TUnitId;
  begin
    SetXInp;
    for u:=1 to UnitNum do
      if synchron then EvalUnit(u)
      else begin
        v:=random(integer(UnitNum))+1;
        EvalUnit(v); UpdUOut(u);
      end;
    if LernF then
      for c:=0 to UnitNum do
        for u:=1 to UnitNum do
          EvalConn(c,u);
    for u:=1 to UnitNum do UpdUOut(u);
  end;
```

```
{------------}
{ Simulation }
{------------}
PROCEDURE dis1Pat;
{ Can be used to disturb disturb Inputpattern }
  var i,k: integer;
  begin end;

PROCEDURE dis2Pat;
{ Can be used to disturb disturb Inputpattern }
  begin end;

PROCEDURE lern;
{ 10 Lern-Cycles for a Set of Patterns defined in Example }
{ Show Pattern PSTime-TimeSteps }
  var t,k,p: integer;
  begin
    writeln(dev,'Pattern-Lerning');
    lernF:=true;
    for k:=1 to 10 do
    begin
      for p:=1 to PNum do begin
        if rpres then WSt:=random(PNum)+1 else WSt:=p;
        writeln(dev,'Pattern ',WSt,'   RE=',RE:0:2);
        InitUnits;
        for t:=1 to PSTime do begin
          TimeStep; wrUnits;
          if event=ESC then exit;
        end;
      end;
      wrConns;
    end;
  end;

PROCEDURE remind;
{ Remind Output to every Pattern without R-Signal }
  var t: integer;
  begin
    writeln(dev,'Pattern-Reminding');    lernF:=false;
    for WSt:=1 to PNum do begin
      writeln(dev,'Pattern ',WSt);
      InitUnits;
      for t:=1 to PSTime do begin
        TimeStep; wrUnits;
        if event=ESC then exit;
      end;
    end;
  end;

{----------------}
```

```
{ Test-Procedures }
{-----------------}
PROCEDURE Test1;
  begin
    writeln(dev,'Init Example');
    InitExample;
  end;

PROCEDURE Test2;
  begin
    writeln(dev,'Test-Routine 2');
  end;

{--------------}
{ Menu-Control }
{--------------}
PROCEDURE wrMenu;
{ Write Menu in first screenline }
  begin
    window(1,1,80,1);
    TextBackGround(Blue);
    ClrScr;
    write('Save Load sTatus Help lerN reMind',
            ' DisturB Intr Cont test1/2 R');
    window(1,3,80,48);
    TextBackGround(Black);
  end;

PROCEDURE Menu;
{ wait for Key / Menu-selection }
  var ch:char;
  begin
    repeat
      ch:=ReadKey;
      chdo(ch);
    until ch='c';
  end;

PROCEDURE chdo(ch:char);
{ Branch on Menu-Selection ch }
  begin
    case ch of
     's': SaveStatus;  { Save status }
     'l': LoadStatus;  { Load status }
     't': wrStatus;    { write sTatus }
     'h': Menu;        { Help (not implemented) }
     'n': lern;        { lerN }
     'm': remind;      { reMind }
     'd': dis1pat;     { disturb 1 }
     'b': dis2pat;     { disturb 2 }
```

```
     'i': Menu;        { Interrupt }
     'c':              { Continue procedure }
          writeln('Procedure Continued');
     '1': Test1;
     '2': Test2;
     'r': wrMenu;      { Refresh }
    end;
  end;

FUNCTION event:char;
{ look for keyboard-event }
  var ch:char;
  begin
    event:=none;
    if KeyPressed then begin
      ch:=ReadKey;
      chdo(ch);
      event:=ch;
    end;
  end;

{--------------}
{ Main Program }
{--------------}
BEGIN
  TextMode(Font8x8+CO80);    { Screen-Init. }
  TextColor(Yellow);         { Color-Init. }
  Randomize;                 { Random-Generator-Init. }
  wrMenu;                    { write Menu }
  assignCrt(dev);            { Output-Device-Init. }
  rewrite(dev);
  InitNet;                   { Net-Init }
  Menu;                      { Wait for Menu-Selection }
  close(dev);
END.
```

7 Protokoll von Beispiel 4 des Programms

Die Lernrate lr wurde auf 0.002 gesetzt, um auch bei diesem kleinen Beispiel die Oszillationen der Output-Units am Anfang der Lernphase gut erkennen zu knnen. In jeder Zeile sind die Aktivationen der 12 Units in Prozent ausgedruckt.

Zur Bedeutung der einzelnen Units vergleiche Bild 3 Kap. 2.

```
Pattern-Lerning
   U1   U2   U3   U4   U5   U6   U7   U8   U9  U10  U11  U12

Pattern 1   RE=2.00
 -100 -100 -200  200  200  100 -100  100  300    0 -300  100
 -100 -100  100    0 -200  300  300 -100 -100  300 -302  100
 -100 -100    0  200 -200    0 -100  200  300 -298 -304  100
 -100 -100  200 -200 -200  200    0    0  200   -0 -305  100
 -100 -100  100   -0 -200 -100 -100  300  300   -0   -7  100
 -100 -100   36   -0  -72   -0 -100  200  300 -300 -307  100
 -100 -100  200 -200 -200   -0 -100  200  300   -5 -309  100
 -100 -100  123  -45 -200 -100 -100  300  300   -5 -311  100
 -100 -100  123  -46 -200 -100 -100  300  300 -302 -313  100
 -100 -100  200 -200 -200 -100 -100  300  300 -304 -314  100
 -100 -100  200 -200 -200 -100 -100  300  300 -306 -316  100
 -100 -100  200 -200 -200 -100 -100  300  300 -307 -318  100
 -100 -100  200 -200 -200 -100 -100  300  300 -309 -320  100
 -100 -100  200 -200 -200 -100 -100  300  300 -311 -321  100
 -100 -100  200 -200 -200 -100 -100  300  300 -313 -323  100
 -100 -100  200 -200 -200 -100 -100  300  300 -314 -325  100
 -100 -100  200 -200 -200 -100 -100  300  300 -316 -327  100
 -100 -100  200 -200 -200 -100 -100  300  300 -318 -328  100
 -100 -100  200 -200 -200 -100 -100  300  300 -320 -330  100
 -100 -100  200 -200 -200 -100 -100  300  300 -322 -332  100
Pattern 2   RE=1.94
 -100  100  100    0 -200  100 -100  100  300 -608  -11  100
 -100  100  -44 -200 -112    0 -100  200  300  290 -311  100
 -100  100  200  200 -200  100  100  100  100   -8 -313  100
 -100  100   61  -77 -200  100 -100  100  300  292  285  100
 -100  100    0  200  200 -100 -100  300  300  294 -313  100
 -100  100  200  200 -200  200  200    0    0 -304 -315  100
 -100  100    0 -200 -200  100 -100  100  300   -6  -17  100
 -100  100   53  -60 -167    0    0  200  200  294 -317  100
 -100  100  200  200 -200 -100 -100  300  300   -4  -19  100
 -100  100   72  -43 -187  100 -100  100  300 -304 -319  100
 -100  100    0 -200 -200 -100 -100  300  300  294 -321  100
 -100  100  200  200 -200    0    0  200  200 -304 -323  100
 -100  100    0 -200 -200  100 -100  100  300   -6  -24  100
 -100  100   69  -61 -200    0    0  200  200  294 -325  100
 -100  100  200  200 -200 -100 -100  300  300   -4  -26  100
```

```
 -100  100   78  -44 -200  100 -100  100  300 -304 -327  100
 -100  100    0 -200 -200 -100 -100  300  300  294 -329  100
 -100  100  200  200 -200    0    0  200  200 -304 -330  100
 -100  100    0 -200 -200  100 -100  100  300   -6  -32  100
 -100  100   69  -63 -200    0    0  200  200  294 -333  100
Pattern 3   RE=1.91
  100 -100 -100 -200    0    0  100  200  100 -605 -334  100
  100 -100    0 -200 -200  100  200  100    0  -10  295  100
  100 -100 -151 -102  200    0    0  200  200  290   -3  100
  100 -100  115  200  -29  100  300  100 -100   -9   -3  100
  100 -100  -28  -85  -30  100 -100  100  300  291 -303  100
  100 -100  200  200 -200  100  100  100  100  293 -305  100
  100 -100  200  200 -200  100 -100  100  300  295  293  100
  100 -100    0  200  200  100 -100  100  300  297 -305  100
  100 -100  200  200 -200  200  200    0    0  299 -307  100
  100 -100  200  200 -200  100 -100  100  300    0   -8  100
  100 -100   44    4  -84  100 -100  100  300  300 -309  100
  100 -100  200  200 -200   18 -100  182  300  302 -310  100
  100 -100  200  200 -200  100 -100  100  300  277 -312  100
  100 -100  200  200 -200  100 -100  100  300  306 -314  100
  100 -100  200  200 -200  100 -100  100  300  308 -316  100
  100 -100  200  200 -200  100 -100  100  300  309 -317  100
  100 -100  200  200 -200  100 -100  100  300  311 -319  100
  100 -100  200  200 -200  100 -100  100  300  313 -321  100
  100 -100  200  200 -200  100 -100  100  300  315 -323  100
  100 -100  200  200 -200  100 -100  100  300  316 -324  100
Pattern 4   RE=1.87
  100  100 -100    0 -200  100 -100  100  300 -912  264  100
  100  100    0 -200  200  200  100    0  100  315 -303  100
  100  100    0  200 -200    0  200  200    0   17  295  100
  100  100  184  168  200  200    0    0  200   17   -3  100
  100  100   70  171  -31  100  100  100  100   17   -3  100
  100  100   71  173  -32  100 -100  100  300  318  297  100
  100  100  200  200  200  100 -100  100  300  319 -301  100
  100  100    0  200 -200  100  100  100  100  321 -303  100
  100  100    0  200 -200  200    0    0  200  323  295  100
  100  100  200  200  200  200    0    0  200   25   -3  100
  100  100   84  200  -32  100  100  100  100   25   -3  100
  100  100   84  200  -32  100 -100  100  300  325  297  100
  100  100  200  200  200  100 -100  100  300  327 -301  100
  100  100    0  200 -200  100  100  100  100  329 -303  100
  100  100    0  200 -200  200    0    0  200  331  295  100
  100  100  200  200  200  200    0    0  200   32   -3  100
  100  100   84  200  -33  100  100  100  100   33   -3  100
  100  100   83  200  -33  100 -100  100  300  333  297  100
  100  100  200  200  200  100 -100  100  300  335 -302  100
  100  100    0  200 -200  100  100  100  100  337 -303  100
Connection-Strength:
    0    0    0    0    0  100  100  100  100 -300 -300  100
    0    0    0    0    0    0    0    0    0   22   10    0
```

```
    0    0    0    0    0    0    0    0    0    7   10    0
    0    0    0    0    0 -100 -100  100  100    0    0    0
    0    0    0    0    0  100    0 -100    0    0    0    0
    0    0    0    0    0    0  100    0 -100    0    0    0
    0    0    0    0    0    0    0    0    0  300    0    0
    0    0    0    0    0    0    0    0    0    0  300    0
    0    0    0    0    0    0    0    0    0  300    0    0
    0    0    0    0    0    0    0    0    0    0  300    0
    0    0    0  200    0    0    0    0    0    0    0    0
    0    0    0    0  200    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0   10  -25    0
Pattern 1   RE=1.84
 -100 -100  100 -200    0  100    0  100  200  -12 -934  100
 -100 -100  160 -120 -200 -100    0  300  200  281  -46  100
 -100 -100    0  200 -200 -100 -100  300  300 -317  -47  100
 -100 -100  200 -200 -200  200    0    0  200 -319 -348  100
 -100 -100  200 -200 -200 -100 -100  300  300  -21  -49  100
 -100 -100  200 -200 -200 -100 -100  300  300 -321 -350  100
 -100 -100  200 -200 -200 -100 -100  300  300 -323 -352  100
 -100 -100  200 -200 -200 -100 -100  300  300 -325 -354  100
 -100 -100  200 -200 -200 -100 -100  300  300 -326 -355  100
 -100 -100  200 -200 -200 -100 -100  300  300 -328 -357  100
 -100 -100  200 -200 -200 -100 -100  300  300 -330 -359  100
 -100 -100  200 -200 -200 -100 -100  300  300 -332 -360  100
 -100 -100  200 -200 -200 -100 -100  300  300 -333 -362  100
 -100 -100  200 -200 -200 -100 -100  300  300 -335 -364  100
 -100 -100  200 -200 -200 -100 -100  300  300 -337 -365  100
 -100 -100  200 -200 -200 -100 -100  300  300 -338 -367  100
 -100 -100  200 -200 -200 -100 -100  300  300 -340 -369  100
 -100 -100  200 -200 -200 -100 -100  300  300 -342 -370  100
 -100 -100  200 -200 -200 -100 -100  300  300 -344 -372  100
 -100 -100  200 -200 -200 -100 -100  300  300 -345 -374  100
Pattern 2   RE=1.78
 -100  100 -200 -200  200  200  200    0    0    0 -265  100
 -100  100  101    1 -200  100  300  100 -100  -15  -36  100
 -100  100   27 -145 -200    6 -100  194  300  285 -337  100
 -100  100  200  200 -200 -100 -100  300  300   74 -338  100
 -100  100  200  200 -200  100 -100  100  300 -312 -340  100
 -100  100    0 -200 -200  100 -100  100  300  287 -342  100
 -100  100  200  200 -200    0    0  200  200  288 -343  100
 -100  100  200  200 -200  100 -100  100  300  -10  -45  100
 -100  100   51  -97 -200  100 -100  100  300  290 -346  100
 -100  100  200  200 -200 -100 -100  300  300  292 -348  100
 -100  100  200  200 -200  100 -100  100  300 -306 -349  100
 -100  100    0 -200 -200  100 -100  100  300  292 -351  100
 -100  100  200  200 -200    0    0  200  200  294 -353  100
 -100  100  200  200 -200  100 -100  100  300   -4  -54  100
 -100  100   78  -44 -200  100 -100  100  300  296 -355  100
 -100  100  200  200 -200 -100 -100  300  300  297 -357  100
 -100  100  200  200 -200  100 -100  100  300 -301 -359  100
```

```
 -100  100    0 -200 -200  100 -100  100  300  297 -360  100
 -100  100  200  200 -200    0    0  200  200  299 -362  100
 -100  100  200  200 -200  100 -100  100  300    1  -64  100
Pattern 3   RE=1.74
  100 -100  100    0 -200  300  300 -100 -100 -345  -39  100
  100 -100    0 -200 -200    0 -100  200  300 -293 -324  100
  100 -100    0 -200 -200    0    0  200  200    5 -326  100
  100 -100  126   52 -200    0    0  200  200    5  -27  100
  100 -100  126   52 -200  100 -100  100  300    5  -28  100
  100 -100  127   53 -200  100 -100  100  300  305 -328  100
  100 -100  200  200 -200  100 -100  100  300  307 -330  100
  100 -100  200  200 -200  100 -100  100  300  309 -332  100
  100 -100  200  200 -200  100 -100  100  300  311 -333  100
  100 -100  200  200 -200  100 -100  100  300  313 -335  100
  100 -100  200  200 -200  100 -100  100  300  314 -337  100
  100 -100  200  200 -200  100 -100  100  300  316 -339  100
  100 -100  200  200 -200  100 -100  100  300  318 -340  100
  100 -100  200  200 -200  100 -100  100  300  320 -342  100
  100 -100  200  200 -200  100 -100  100  300  321 -344  100
  100 -100  200  200 -200  100 -100  100  300  323 -346  100
  100 -100  200  200 -200  100 -100  100  300  325 -347  100
  100 -100  200  200 -200  100 -100  100  300  327 -349  100
  100 -100  200  200 -200  100 -100  100  300  328 -351  100
  100 -100  200  200 -200  100 -100  100  300  330 -352  100
Pattern 4   RE=1.68
  100  100    0 -200  200    0  100  200  100  312  247  100
  100  100  200  200  200    0  200  200    0   57  287  100
  100  100  200  200  200  100  100  100  100   58  -12  100
  100  100   42  200 -117  100  100  100  100  359  288  100
  100  100  200  200  200  100 -100  100  300  361  290  100
  100  100  200  200  200  100  100  100  100  363 -308  100
  100  100    0  200 -200  100  100  100  100  364  290  100
  100  100  200  200  200  200    0    0  200  366  292  100
  100  100  200  200  200  100  100  100  100   68   -6  100
  100  100   68  200  -63  100  100  100  100  369  294  100
  100  100  200  200  200  100 -100  100  300  370  295  100
  100  100  200  200  200  100  100  100  100  372 -303  100
  100  100    0  200 -200  100  100  100  100  374  295  100
  100  100  200  200  200  200    0    0  200  375  297  100
  100  100  200  200  200  100  100  100  100   77   -1  100
  100  100   95  200  -10  100  100  100  100  378  299  100
  100  100  200  200  200  100  -50  100  250  380  301  100
  100  100  200  200  200  100  100  100  100  382 -297  100
  100  100    0  200 -200  100  100  100  100  383  301  100
  100  100  200  200  200  200    0    0  200  385  303  100
Connection-Strength:
    0    0    0    0    0  100  100  100  100 -300 -300  100
    0    0    0    0    0    0    0    0    0   41   24    0
    0    0    0    0    0    0    0    0    0   22   25    0
    0    0    0    0    0 -100 -100  100  100    0    0    0
```

```
    0    0    0    0    0  100    0 -100    0    0    0    0
    0    0    0    0    0    0  100    0 -100    0    0    0
    0    0    0    0    0    0    0    0    0  300    0    0
    0    0    0    0    0    0    0    0    0    0  300    0
    0    0    0    0    0    0    0    0    0  300    0    0
    0    0    0    0    0    0    0    0    0    0  300    0
    0    0    0  200    0    0    0    0    0    0    0    0
    0    0    0    0  200    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0   23  -44    0

[ ... 7 weitere Lernrunden  ... ]

Pattern 1   RE=0.74
 -100 -100 -200  200  200 -100  100  300  100 -300  300  100
 -100 -100    0 -200  200  300  300 -100 -100 -399   27  100
 -100 -100    0 -200  200    0  200  200    0 -400 -572  100
 -100 -100  200 -200 -200    0  200  200    0 -101 -273  100
 -100 -100  200 -200 -200 -100 -100  300  300 -103 -274  100
 -100 -100  200 -200 -200 -100 -100  300  300 -404 -575  100
 -100 -100  200 -200 -200 -100 -100  300  300 -405 -577  100
 -100 -100  200 -200 -200 -100 -100  300  300 -406 -578  100
 -100 -100  200 -200 -200 -100 -100  300  300 -407 -579  100
 -100 -100  200 -200 -200 -100 -100  300  300 -408 -580  100
 -100 -100  200 -200 -200 -100 -100  300  300 -410 -581  100
 -100 -100  200 -200 -200 -100 -100  300  300 -411 -582  100
 -100 -100  200 -200 -200 -100 -100  300  300 -412 -583  100
 -100 -100  200 -200 -200 -100 -100  300  300 -413 -584  100
 -100 -100  200 -200 -200 -100 -100  300  300 -414 -585  100
 -100 -100  200 -200 -200 -100 -100  300  300 -415 -585  100
 -100 -100  200 -200 -200 -100 -100  300  300 -416 -586  100
 -100 -100  200 -200 -200 -100 -100  300  300 -417 -587  100
 -100 -100  200 -200 -200 -100 -100  300  300 -418 -588  100
 -100 -100  200 -200 -200 -100 -100  300  300 -419 -589  100
Pattern 2   RE=0.73
 -100  100    0 -200 -200  100  200  100    0 -300 -598  100
 -100  100    0 -200 -200    0    0  200  200  379  -91  100
 -100  100  200  200 -200    0    0  200  200   81  -93  100
 -100  100  200  200 -200  100 -100  100  300   82  -94  100
 -100  100  200  200 -200  100 -100  100  300  383 -396  100
 -100  100  200  200 -200  100 -100  100  300  384 -397  100
 -100  100  200  200 -200  100 -100  100  300  386 -399  100
 -100  100  200  200 -200  100 -100  100  300  387 -400  100
 -100  100  200  200 -200  100 -100  100  300  388 -401  100
 -100  100  200  200 -200  100 -100  100  300  389 -402  100
 -100  100  200  200 -200  100 -100  100  300  391 -404  100
 -100  100  200  200 -200  100 -100  100  300  392 -405  100
 -100  100  200  200 -200  100 -100  100  300  393 -406  100
 -100  100  200  200 -200  100 -100  100  300  394 -407  100
 -100  100  200  200 -200  100 -100  100  300  395 -408  100
 -100  100  200  200 -200  100 -100  100  300  397 -410  100
```

```
 -100  100  200  200 -200  100 -100  100  300  398 -411  100
 -100  100  200  200 -200  100 -100  100  300  399 -412  100
 -100  100  200  200 -200  100 -100  100  300  400 -413  100
 -100  100  200  200 -200  100 -100  100  300  401 -414  100
Pattern 3   RE=0.71
  100 -100    0  200  200  100  200  100    0 -388 -500  100
  100 -100    0 -200 -200  200  200    0    0  372  -86  100
  100 -100  200  200 -200    0    0  200  200   73  -87  100
  100 -100  200  200 -200  100 -100  100  300   75  -88  100
  100 -100  200  200 -200  100 -100  100  300  376 -389  100
  100 -100  200  200 -200  100 -100  100  300  377 -390  100
  100 -100  200  200 -200  100 -100  100  300  379 -391  100
  100 -100  200  200 -200  100 -100  100  300  381 -393  100
  100 -100  200  200 -200  100 -100  100  300  382 -394  100
  100 -100  200  200 -200  100 -100  100  300  384 -395  100
  100 -100  200  200 -200  100 -100  100  300  386 -396  100
  100 -100  200  200 -200  100 -100  100  300  387 -397  100
  100 -100  200  200 -200  100 -100  100  300  389 -399  100
  100 -100  200  200 -200  100 -100  100  300  390 -400  100
  100 -100  200  200 -200  100 -100  100  300  392 -401  100
  100 -100  200  200 -200  100 -100  100  300  394 -402  100
  100 -100  200  200 -200  100 -100  100  300  395 -403  100
  100 -100  200  200 -200  100 -100  100  300  397 -405  100
  100 -100  200  200 -200  100 -100  100  300  398 -406  100
  100 -100  200  200 -200  100 -100  100  300  400 -407  100
Pattern 4   RE=0.69
  100  100 -100    0 -200 -100    0  300  200 -413  -15  100
  100  100 -177 -200 -155  200  100    0  100  -30   77  100
  100  100    0 -200  200  100  100  100  100  269  378  100
  100  100  200  200  200    0  200  200    0  570  380  100
  100  100  200  200  200  100  100  100  100  272   81  100
  100  100  200  200  200  100  100  100  100  573  383  100
  100  100  200  200  200  100  100  100  100  574  384  100
  100  100  200  200  200  100  100  100  100  575  386  100
  100  100  200  200  200  100  100  100  100  577  387  100
  100  100  200  200  200  100  100  100  100  578  389  100
  100  100  200  200  200  100  100  100  100  579  391  100
  100  100  200  200  200  100  100  100  100  580  392  100
  100  100  200  200  200  100  100  100  100  582  394  100
  100  100  200  200  200  100  100  100  100  583  396  100
  100  100  200  200  200  100  100  100  100  584  397  100
  100  100  200  200  200  100  100  100  100  585  399  100
  100  100  200  200  200  100  100  100  100  586  400  100
  100  100  200  200  200  100  100  100  100  588  402  100
  100  100  200  200  200  100  100  100  100  589  404  100
  100  100  200  200  200  100  100  100  100  590  405  100
Connection-Strength:
    0    0    0    0    0  100  100  100  100 -300 -300  100
    0    0    0    0    0    0    0    0    0   99   92    0
    0    0    0    0    0    0    0    0    0   92  100    0
```

```
    0    0    0    0    0 -100 -100  100  100    0    0    0
    0    0    0    0    0  100    0 -100    0    0    0    0
    0    0    0    0    0    0  100    0 -100    0    0    0
    0    0    0    0    0    0    0    0    0  300    0    0
    0    0    0    0    0    0    0    0    0    0  300    0
    0    0    0    0    0    0    0    0    0  300    0    0
    0    0    0    0    0    0    0    0    0    0  300    0
    0    0    0  200    0    0    0    0    0    0    0    0
    0    0    0    0  200    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0  100  -85    0

Pattern-Reminding
   U1   U2   U3   U4   U5   U6   U7   U8   U9  U10  U11  U12

Pattern 1
 -100 -100    0  200    0  300  200 -100    0 -607 -892  100
 -100 -100    0 -200 -200  200  100    0  100 -390 -277  100
 -100 -100    0 -200 -200    0    0  200  200  -90   23  100
 -100 -100    0 -200  200    0    0  200  200  -90 -277  100
 -100 -100    0 -200 -200    0  200  200    0  -90 -277  100
 -100 -100    0 -200 -200    0    0  200  200  -90 -277  100
 -100 -100    0 -200 -200    0    0  200  200  -90 -277  100
 -100 -100    0 -200 -200    0    0  200  200  -90 -277  100
 -100 -100    0 -200 -200    0    0  200  200  -90 -277  100
 -100 -100    0 -200 -200    0    0  200  200  -90 -277  100
Pattern 2
 -100  100    0    0 -200  200  300    0 -100 -110  192  100
 -100  100    0 -200  200  100    0  100  200   93 -377  100
 -100  100    0  200 -200    0  200  200    0  393  -77  100
 -100  100    0  200 -200  200    0    0  200   93  -77  100
 -100  100    0  200 -200  200    0    0  200   93  -77  100
 -100  100    0  200 -200  200    0    0  200   93  -77  100
 -100  100    0  200 -200  200    0    0  200   93  -77  100
 -100  100    0  200 -200  200    0    0  200   93  -77  100
 -100  100    0  200 -200  200    0    0  200   93  -77  100
 -100  100    0  200 -200  200    0    0  200   93  -77  100
Pattern 3
  100 -100    0 -200    0  100  100  100  100 -207 -977  100
  100 -100    0 -200 -200    0  100  200  100  407  207  100
  100 -100    0  200  200    0    0  200  200  107  207  100
  100 -100    0  200  200  200  200    0    0  107  -93  100
  100 -100    0  200 -200  200  200    0    0  107  -93  100
  100 -100    0  200 -200  200    0    0  200  107  -93  100
  100 -100    0  200 -200  200    0    0  200  107  -93  100
  100 -100    0  200 -200  200    0    0  200  107  -93  100
  100 -100    0  200 -200  200    0    0  200  107  -93  100
  100 -100    0  200 -200  200    0    0  200  107  -93  100
Pattern 4
  100  100    0 -200    0 -100    0  300  200 -690 -577  100
  100  100    0 -200 -200    0  100  200  100  -10  107  100
```

```
  100  100    0  -96  200    0    0  200  200  290  407  100
  100  100    0  200  200    0  200  200    0  290  107  100
  100  100    0  200  200  200  200    0    0  290  107  100
  100  100    0  200  200  200  200    0    0  290  107  100
  100  100    0  200  200  200  200    0    0  290  107  100
  100  100    0  200  200  200  200    0    0  290  107  100
  100  100    0  200  200  200  200    0    0  290  107  100
  100  100    0  200  200  200  200    0    0  290  107  100
```

[Ende des Protokolls]