Optimierung Rückgekoppelter Hyperpermutationsnetzwerke

Dissertation
zur Erlangung des Doktorgrades
der Mathematisch-Naturwissenschaftlichen Fakultät
der Christian-Albrechts-Universität zu Kiel

vorgelegt von
Tilo Schwarz

Kiel 2000

Referent: .....

Korreferent: .....

Tag der mündlichen Prüfung: .....

Zum Druck genehmigt: Kiel, den .....

.....
Dekan

Inhaltsverzeichnis

1 Einleitung

Diese Arbeit untersucht die Frage, wie man mit einem neuen diskreten RAM-basierten Ansatz zur maschinellen Wahrnehmung, den Hyperpermutationsnetzwerken (Oberländer, 1999), möglichst ressourcensparend Aufgaben der maschinellen Wahrnehmung lösen kann. Hier werden vor allem rückgekoppelte Hyperpermutationsnetzwerke untersucht, da sie die Lösung von Aufgaben der maschinellen Wahrnehmung mit minimalem Ressourcenverbrauch versprechen.

1.1 Maschinelle Wahrnehmung

Die *maschinelle Wahrnehmung* beschäftigt sich mit der Generierung von Information aus Sensordaten. Sensordaten, die nichts anderes als physikalische Meßwerte sind, sollen automatisch in die Welt der Symbole und Bedeutungen transformiert werden. Um diese Aufgabe zu lösen, sind im Laufe der Zeit viele verschiedene Methoden entwickelt worden. Besonders die neuronalen Netze sind in den letzen Jahren in etlichen Bereichen zur Anwendung gekommen.

Inzwischen ist die maschinelle Wahrnehmung (auch durch die immer noch exponentiell steigende Rechnerleistung) auf einem Stand, daß in naher Zukunft Verfahren der maschinellen Wahrnehmung ihren Weg in die Serienprodukte u. a. der Automobilindustrie finden werden. Die Wahrnehmungsaufgaben, die momentan angegangen werden, sind aus menschlicher Perspektive zwar noch sehr einfach. Zum Beispiel soll die Frage beantwortet werden, wo in einem Videobild der Straße Fahrbahnmarkierungen sind oder ob jemand auf dem Beifahrersitz sitzt oder nicht. Für einen Computer sind diese Aufgaben aber nach wie vor schwierig, auch weil er nicht über das „gesammelte Weltwissen“ verfügt, das wir Menschen in jede unserer Wahrnehmungsentscheidungen mit einfließen lassen.

1.2 Motivation für einen RAM-basierten Ansatz

Warum möchten wir einen RAM-basierten Ansatz verwenden? Schon in der Anfangszeit der RAM-basierten neuronalen Netze in den fünfziger Jahren galt, was auch heute noch aktuell ist: Der Digitalcomputer ist bis auf ganz wenige Ausnahmen *das* „Rechenwerk“, auf dem maschinelle Wahrnehmung betrieben wird.

1.2.1 Direkte Implentierbarkeit auf dem Digitalcomputer

Schon vor über dreißig Jahren, nachdem Bledsoe und Browning (1959) ihre n-Tupel-Methode vorgestellt hatten, erkannte Aleksander, daß RAM-basierte Ansätze zur Mustererkennung eine ganz besondere Eigenschaft haben: Das ihnen zugrunde liegende abstrakte mathematische Modell läßt sich direkt in digitaler Hardware umsetzen. Dazu müssen sich die Bausteine des Netzwerks nicht unbedingt wie diskrete Neuronen verhalten, es reicht aus, wenn sie diskret arbeiten. Da das mathematische Modell von RAM-basierten Netzwerken zudem das Nachschlagen in Tabellen als Grundoperation verwendet, ist noch nicht einmal komplizierte Hardware not-

wendig, um die Grundbausteine eines RAM-Netzes zu realisieren. Es reichen, wie der Name schon sagt, einfache RAM-Bausteine dafür aus.

Die direkte Überführbarkeit des mathematischen Modells auf eine physikalische Maschine hat außerdem den Vorteil, daß es zwischen der Theorie und ihrer Implementierung keine künstliche Schranke gibt. Bei Netzwerkmodellen, die auf einer kontinuierlichen Theorie aufbauen, ist das anders. Sie arbeiten mit reellen Zahlen und müssen daher bei ihrer Implementierung auf einem Digitalcomputer immer diskretisiert werden. Wie fein muß aber die Diskretisierung sein, damit die Sätze und Beweise, auf die sich die Theorie stützt, noch gelten. Welche Rechengenauigkeit ist mindestens nötig, wie geht man mit Rundungsfehlern um? Jeder, der schon mit numerischen Verfahren gearbeitet hat weiß, daß diese Fragen oft äußerst schwierig zu beantworten sind. In der Praxis werden solche Probleme üblicherweise so gelöst, daß man numerische Rechnungen erst einmal durchführt und hinterher das Ergebnis auf Plausibilität überprüft. In der Trainingsphase eines Mustererkennungssystems ist dies noch akzeptierbar, während des „Betriebs“ möchte man aber eigentlich ausschließen, daß aufgrund von numerischen Schwierigkeiten Fehler auftreten.

Speziell bei der Berechnung rückgekoppelter kontinuierlicher Netzwerke treten diese Schwierigkeiten in den Vordergrund. Rückgekoppelte Netzwerke sind schon in der Theorie nicht einfach zu handhaben. Da bei der Simulation einer rückgekoppelten Struktur bestimmte Systemausgaben immer wieder als Eingaben verwendet werden, akkumulieren sich numerische Fehler von Iteration zu Iteration auf. In solchen Fällen ist es schwierig, die numerische Stabilität eines Algorithmus zu gewährleisten (Almeida, 1989). Die Simulation eines diskreten Systems, ist dagegen, zumindest aus dieser Perspektive, trivial.

Abgesehen von der „Diskretisierungsfrage“ hat es ein maschinelles Mustererkennungssystem heute praktisch ausnahmslos mit digitalen Eingangs- und Ausgangsdaten zu tun. Auch dies ist eine Motivation, ein Mustererkennungssystem gleich auf einer diskrete Theorie aufzubauen. Nun gibt es auch mathematische Modelle, die sich relativ problemlos in einer Programmiersprache implementieren lassen, die eine für diesen Fall ausreichende Gleitkommaarithmetik anbietet. Nur ist man in diesem Fall auf Hardware angewiesen, die Gleitkommaarithmetik unterstützt.

1.2.2 Möglichst einfache Zielhardware

Man kann dem Argument der direkten Implementierbarkeit entgegenhalten, bei den heutigen Prozessoren (Pentium III etc.) und ihrer Leistungsfähigkeit im Gleitkommabereich spiele es keine Rolle mehr, ob man Gleit-, Festkomma- oder Ganzzahlarithmetik zur Berechnung verwendet. Meistens werden aber einfache Mikrocontroller den Universalprozessoren wie dem Pentium in den Serienprodukten der Automobilindustrie vorgezogen.

Dies hat zwei Gründe: Zum einen sind Prozessoren vom Pentium-Typ zu teuer. Wenn ein technisches Modul in ein Serienfahrzeug eingebaut werden soll, muß es so kostengünstig wie möglich sein. Da ein Fahrzeug aus Tausenden von Einzelteilen besteht, kommt es dort auf jede Mark an. Zum anderen sind solche Prozessoren so komplex, daß es praktisch unmöglich ist, die Fehlerfreiheit eines Prozessors nachzuweisen. Dies wird durch den vor einigen Jahren bekanntgewordenen Fehler in der Gleitkommaeinheit des Pentium-Prozessors bestätigt. Im Fahrzeug trifft man daher vorrangig recht kleine Komponenten an, die unabhängig voneinander getestet werden können. Aus diesen Gründen wird ein Verfahren, das nur einen Prozessor mit Ganzzahlarithmetik benötigt, bei ähnlicher Performanz einem Verfahren, das einen Prozessor mit Gleitkommaarithmetik benötigt, für den Einbau im Serienfahrzeug vorgezogen werden.

RAM-basierte Netzwerke gehen nun noch einen großen Schritt weiter. Je nach Art des Netzwerks benötigen sie nur wenig Ganzzahlarithmetik (Addition) oder *überhaupt keine Arithmetik* mehr, wie z. B. auch Hyperpermutationsnetzwerke. Ist das Netzwerk arithmetikfrei, kann man

auf jede Art von Prozessor verzichten. Hyperpermutationsnetzwerke bestehen in der Implementierung nur noch aus einem Netzwerk von Tabellen. Man kann sie daher entweder mittels am Markt erhältlicher RAM-Bausteine aufbauen, oder aber eigene Hardware entwickeln, was aber durch den sehr einfachen Aufbau eines solchen Netzwerks relativ einfach ist.

Ein weiterer Punkt ist, daß neue Systeme möglichst auf schon im Fahrzeug vorhandener Hardware integriert werden sollen. Denn nur ein einziger weiterer Chip auf einer Platine hat möglicherweise weitreichende Folgen: die Platine wird größer, das die Platine umgebende Gehäuse wird größer, an der bisherigen Stelle im Fahrzeug ist kein Platz mehr für das größer gewordene Gehäuse, es muß eine neue Einbaumöglichkeit gefunden werden usw. Daher sind die Bemühungen groß, an bestehender Hardware möglichst lange festzuhalten. Eine Abteilung der Fahrzeug-Entwicklung kann also durchaus festlegen, daß die Lösung einer bestimmten Aufgabe maximal 500 Byte Speicher benötigen darf, weil auf der bestehenden Hardware nicht mehr Platz ist und man keine neue Hardware hinzufügen wird. Prozessoren, die hier eingesetzt werden, beherrschen teilweise nur 8/16-Bit Ganzzahl-Addition bzw. -Multiplikation.

Auf diesen Gründen basiert die Motivation, für Wahrnehmungsaufgaben im Serienprodukt nach Lösungen mit RAM-basierten Netzwerken zu suchen. Al-Alawi und Stonham (1992) haben in „A Training Strategy and Functional Analysis of Digital Multi-Layer Neural Networks“ gezeigt, daß bei binärem Netzwerkein- und ausgang die funktionale Kapazität eines diskreten Multi-Layer Netzwerks wesentlich größer ist als die eines vergleichbar großen analogen Multi-Layer Netzwerks. Morciniec und Rohwer (1998) haben durch umfangreiche Vergleichtests auf verschiedene Testdaten gezeigt, daß klassische neuronale Netze und RAM-basierte Netze was ihre „Wahrnehmungsleistung“ (Erkennungsrate etc.) angeht, gleichwertig sind. Daher besteht die begründete Hoffnung, daß man auf diesem Weg zu Systemen kommt, die Wahrnehmungsaufgaben mit minimalem Ressourcenverbrauch angemessen lösen können.

Diese Arbeit gliedert sich in die drei Teile „Hyperpermutationsnetzwerke“, „Optimierung“ und „Experimente“. Der erste Teil stellt schon bekannte konnektionistische Modelle vor, führt die Hyperpermutationsnetzwerke ein und untersucht Unterschiede zwischen bekannten Modellen und Hyperpermutationsnetzwerken. Der zweite Teil beschäftigt sich im Detail mit der Frage, wie man vor allem rückgekoppelte Hyperpermutationsnetzwerke optimieren bzw. trainieren kann. Der dritte Teil stellt numerische Experimente zur Optimierung vor.

Teil I

Hyperpermutationsnetzwerke

2 Wahrnehmung

Seit langem versuchen Menschen, die Fähigkeiten von Mensch und Tier auf dem Gebiet der Wahrnehmung zu verstehen. Mit dem Einzug des Computers in Forschung und Technik ist es nun zumindest ansatzweise gelungen, kognitive[1] Fähigkeiten biologischer Wesen mit technischen Mitteln zu erreichen. Da man sich in einer technisierten Welt etliche Anwendungen für kognitive Systeme vorstellen kann, wird auch in vielen Bereichen an kognitiven Systemen geforscht und entwickelt.

Die Wahrnehmung als Mittel zur Lösung bestimmter Aufgaben wird von der Natur schon lange „eingesetzt". Bevor wir zu technischen Lösungen der maschinellen Wahrnehmung kommen, werden wir uns daher die biologische Wahrnehmung etwas genauer ansehen.

2.1 Biologische Wahrnehmung

Lebewesen sind darauf angewiesen, daß ihr Wahrnehmungsapparat effizient ist. Je höher ein Lebewesen entwickelt ist, desto ausgeprägter sind im allgemeinen seine kognitiven Fähigkeiten und desto komplexer ist sein Wahrnehmungsapparat. Die Bandbreite reicht hier von einfachen Einzellern, die „nur" die grobe Richtung des einfallenden Lichts wahrnehmen, bis zu hochentwickelten Primaten und dem Menschen, der mit seinen sprichwörtlichen sechs Sinnen unglaubliche kognitive Fähigkeiten entwickelt hat.

Warum hat die Evolution solche Leistungen hervorgebracht, für die ein Lebewesen kostbare biologische Ressourcen unterhalten muß? Jedes Lebewesen lebt in einer sich ständig ändernden *Umwelt*, die es zum größten Teil *nicht* beeinflussen kann. In dieser Umwelt ist es meistens auf sich gestellt und muß autonom agieren. Im täglichen Kampf ums Überleben ist es wichtig, die Umwelt und vor allem Änderungen der Umwelt wahrnehmen zu können. Plakatives Beispiel sind hier Räuber- und Beutetiere. Räuber würden ohne ausreichende kognitive Fähigkeiten verhungern und Beutetiere gefressen werden.

Einfache Lebewesen beschränken sich bei ihren Aktionen hauptsächlich auf Reaktionen auf Sinnesreize. Man könnte ihren Aktionszyklus mit 'Sinnesreiz → Reaktion → Sinnesreiz → ...' beschreiben. Dabei kann jede Aktion eine Änderung der Umwelt zur Folge haben. Je höher ein Lebewesen entwickelt ist, desto größer wird der Anteil der selbständig ausgelösten Aktionen.

Obwohl hochentwickelte Lebewesen sich bei der Auswahl ihrer Aktionen auch auf ein „inneres Bild" der Umwelt stützen, ist ein explizites Verständnis der Umwelt nicht immer notwendig, um aus einer Wahrnehmung die richtige Handlung abzuleiten. Ein Beispiel hierfür sind Reflexe, die uns handeln lassen, bevor wir überhaupt die Situation erfaßt haben. Auf der anderen Seite liegt eine herausragende Stärke der menschlichen Wahrnehmung in der Verschmelzung von Wahrgenommenen und bereits Bekanntem. Uns gelingt es zum Beispiel, so verschiedene Ob-

[1] Die Verwendung der Begriffe Wahrnehmung und Kognition sind in der Literatur leider nicht einheitlich. Wahrnehmung wird sowohl im Sinne von "Ich habe *etwas* wahrgenommen (bemerkt)" verwendet, als auch im Sinne von "Ich habe *das Auto* wahrgenommen (erkannt)". Kognition (lat. Erkenntnis) beinhaltet Wahrnehmung und geht je nach Auslegung auch darüber hinaus. Wir werden Wahrnehmung und Kognition weitgehend synonym im Sinne von „bemerkt und erkannt" verwenden.

jekte wie Hasenohren, Hundeohren und Menschenohren als Ohren zu erkennen und mit jedem erkannten Objekt wächst unsere Erkennungssicherheit.

Eine sehr wichtiges Merkmal biologischer Wahrnehmungsapparate ist ihre Effizienz. Die zum Leben benötigten kognitiven Fähigkeiten werden mit einem Minimum an biologischen Ressourcen erreicht, da das Lebewesen seinen Wahrnehmungsapparat selbst mit Energie versorgen muß. Ein gutes Beispiel sind hier Insekten. Trotz ihres winzigen „Gehirns“ ist in ihrem kognitiven Repertoire u. a. die Erkennung von Beutetieren, Rivalen und Partnern durch den Sehsinn (Libellen), den Hörsinn (Grillen), den Geruchssinn (Schwärmer) und den Tastsinn (Ameisen).

2.2 Maschinelle Wahrnehmung

Die maschinelle Wahrnehmung beschäftigt sich mit der Frage, wie man Wahrnehmungsaufgaben mit technischen Mitteln lösen kann. Eine Wahrnehmungsaufgabe ist eine Aufgabe, die normalerweise mit Hilfe der Wahrnehmung gelöst werden kann, z. B. die Erkennung von Fahrzeugen im Straßenverkehr. Meistens ist die maschinelle Wahrnehmung nur ein Teil eines Gesamtsystems, das eine Aufgabe mit einer Wahrnehmungskomponente zu bewältigen hat. Zum Beispiel soll ein Automat in der Produktion fehlerhafte Teile visuell erkennen und automatisch aussortieren. Ganz analog zum biologischen Fall haben wir nun ein technisches „Subjekt“ in einer Umwelt. Um über seine Umwelt Informationen zu erhalten, hat ein solches System Sensoren, die Sinnesorganen entsprechen. Hat ein System auch Aktuatoren, kann es Aktionen durchführen und wiederum auf die Umwelt einwirken. Auf diese Weise entsteht ein Regelkreis, den Abbildung 2.1 zeigt.

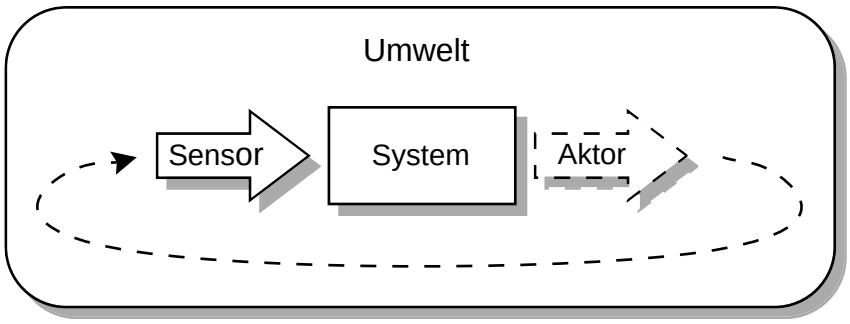


Abbildung 2.1: Der Regelkreis von Umwelt und System. Mit seinen Sensoren nimmt das System Informationen aus der Umwelt auf. Sind auch Aktuatoren vorhanden, wirkt das System auf die Umwelt zurück und ein Regelkreis entsteht.

Es gibt sehr verschiedene Anforderungen an die maschinelle Wahrnehmung. Zu den sicher einfacheren Beispielen gehört die visuelle Erkennung geometrischer Formen (Dreiecke, Vierecke) in einem Videobild, z. B. zur Steuerung einer Sortiermaschine. Sehr schwierig sind dagegen Aufgaben, die uns Menschen typischerweise recht leicht fallen, z. B. die Erkennung von belebten und unbelebten Objekten in einem natürlichen Umfeld. Wir nutzen zur Erkennung meistens mehrere Aspekte eines Objekts. Um andere Menschen zu erkennen können wir in ihr Gesicht sehen, ihren Gang betrachten, ihre Stimme hören und vieles mehr. Besonders schwierig sind Aufgaben, bei denen das technische System praktisch auf sich allein gestellt ist. Solche sogenannten autonomen Systeme können im Zweifelsfall nicht bei einem Menschen nachfragen, was nun zu tun sei, sondern müssen über große Zeiträume alleine handeln. Hier nimmt die Wahrnehmungskomponente einen entscheidenen Teil ein. Prominentes Beispiel für ein zumindest

teilautonomes System ist das Marsfahrzeug Sojourner, das im Rahmer der Mars Pathfinder Mission am 4.7.1997 auf dem Mars gelandet ist.

Vor allem für die kommerzielle Verwendung einer technischen Lösung einer Wahrnehmungsaufgabe gibt es zwei Kriterien, die fast immer über die mögliche Nutzung eines Systems mit entscheiden:

- Das System muß schnell genug sein, um einem meistens von außen vorgegebenen Takt folgen zu können. Dies gilt vor allem für Komponenten in Echtzeitsystemen, wie zum Beispiel Fahrzeuge im Straßenverkehr.
- Das System muß kostengünstig sein, da es sonst als möglicher Kandidat für eine Problemlösung ausscheidet. Dieser Punkt wird bei Komponenten wichtig, die in Serienprodukte eingebaut werden, da bei Serienprodukten der Kostendruck groß ist.

2.3 Realisierungsansätze für maschinelle Wahrnehmung

Angesichts der verschiedenen Wahrnehmungsaufgaben, die man sich vorstellen kann, stellt sich die Frage, wie wir eine Wahrnehmungsaufgabe mit technischen Mitteln lösen können. Die beiden Hauptrichtungen, die sich im Laufe der Zeit etabliert haben, sind modellbasierte Verfahren und statistische Verfahren. Modellbasierte Verfahren modellieren die „Welt“, in der sich die Aufgabe stellt. Bei der Lösung der Aufgabe versucht man dann, die Modelle mit der Realität in Einklang zu bringen. Statistische Verfahren verwenden statistische Eigenschaften der Wahrnehmungsaufgabe, um damit zur Lösung der Aufgabe zu kommen. Trotz aller Unterschiede gibt es auch Überschneidungen zwischen den beiden Richtungen. Zu der immer wieder umstrittenen Frage, welche Verfahren wohl die Natur verwendet, zeigt Abbildung 2.2 eine mögliche „Antwort“.

2.3.1 Modellbasierte Verfahren

Modellbasierte Verfahren verwenden zur Lösung einer Wahrnehmungsaufgabe ein Modell, das einen Teil der Realität hinreichend genau approximiert (Zitat von Hermann Haken aus einer Synergetik-Vorlesung:„Für unsere Zwecke ist ein Pferd in nullter Näherung eine Kugel“). Je besser das Modell für eine bestimmte Aufgabe gewählt wurde, desto besser können später auch die Ergebnisse sein. Die Wahl eines ungeeigneten Modells kann zur Folge haben, daß sich die gestellte Wahrnehmungsaufgabe damit nicht lösen läßt. Aus der großen Zahl von modellbasierten Erkennungverfahren seien hier nur zwei Beispiele kurz angeschnitten.

In der Bildverarbeitung verwendet man überwiegend geometrische und physikalische Modelle. Möchte man z. B. in Videobildern von Straßenszenen Fahrzeuge erkennen, so könnte man zur Modellbildung die geometrischen Formen von Fahrzeugen heranziehen. Auch dynamische Eigenschaften von Fahrzeugen könnten einfließen, z. B. daß sich Fahrzeuge nur mit bestimmten Geschwindigkeiten bewegen, daß sie gewissen Beschleunigungen nicht überschreiten und sich (normalerweise) nicht seitwärts bewegen. Auf dem Gebiet der modellgestützen Objekterkennung gibt es sehr viele Arbeiten, stellvertretend seien hier nur zwei genannt: Chin und Dyer (1986); Lamdan und Wolfson (1988)

In anderen Bereichen, wie z. B. der Sprachverarbeitung, wird stark von Methoden der künstlichen Intelligenz Gebrauch gemacht. Um einen Satz wie „Er ist gestern wiedergekommen.“ verstehen zu können, ist eine Menge Hintergrundwissen erforderlich (Wer ist „Er“, wann war „gestern“, wohin ist er wiedergekommen?). Die hier verwendeten Modelle, z. B. semantische Netze, sind selten physikalischer Natur, sondern sie modellieren meistens Zusammenhänge zwischen realen oder abstrakten Objekten.

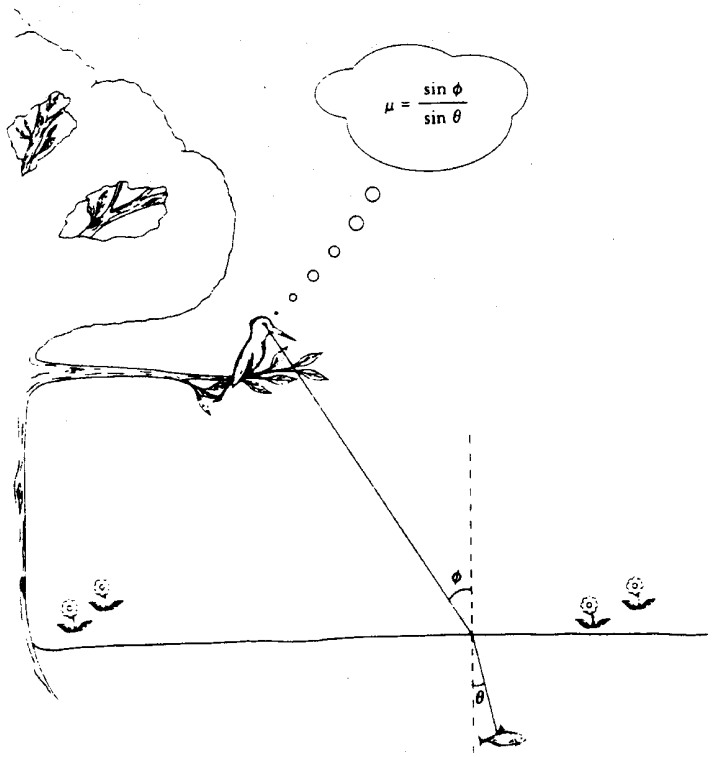


Abbildung 2.2: Kognition in der Natur. Wie fängt der Vogel den Fisch? Modellbasiert oder konnektionistisch? (aus Varela, 1990)

Alle rein modellbasierten Verfahren haben gemein, daß man sich à priori für ein bestimmtes Modell entscheiden muß. Normalerweise bedeutet dies, daß jemand mit dem nötigen Fachwissen ein Modell für eine bestimmte Wahrnehmungsaufgabe erstellt. Approximiert es die Realität für die Aufgabe hinreichend gut, ist das Problem gelöst, sonst muß man das Modell weiter anpassen. Dieses Vorgehen hat zwei Nachteile: Erstens kann ein Modell nur das beinhalten, woran der Modelldesigner auch gedacht hat. Ist die Variabilität der Daten groß, kann es sehr schwierig sein, diese Variabilität zu modellieren. Zweitens gibt es etliche Wahrnehmungsaufgaben, deren zugrundeliegende Prozesse so kompliziert sind, daß sie einem Modell praktisch nicht zugänglich sind.

In dieser Arbeit werden wir nicht weiter auf modellbasierte Verfahren eingehen, sondern uns nun den statistischen Verfahren zuwenden.

2.3.2 Statistische Verfahren

Im Gegensatz zu modellbasieren Verfahren versuchen statistische Verfahren die statistischen Eigenschaften eines Teils der „realen Welt“ zu nutzen, um eine Wahrnehmungsaufgabe zu lösen. Wenn bei einer Wahrnehmungsaufgabe z. B. zwei verschiedene Objekte visuell voneinander unterschieden werden sollen, muß das System die statistischen Unterschiede in den visuellen Repräsentationen der beiden Objekte erkennen können. Die meisten dieser Verfahren haben gemeinsam, daß sie die Lösung einer Wahrnehmungsaufgabe durch Präsentation von Beispieldaten „erlernen“. Dabei steht „Lernen“ für den Vorgang, unbekannte Parameter des Systems durch die Präsentation von Beispieldaten zu ermitteln. Man präsentiert einem System zur Buchstabenerkennung beispielsweise viele Bilder von Buchstaben und kann so die inneren Parameter des Systems adaptieren, so daß es auch bei einem neuen unbekannten Bild eines Buchstabens den richtigen Buchstaben erkennt.

Innerhalb der maschinellen Wahrnehmung faßt man einen großen Bereich unter dem Begriff *Mustererkennung* zusammen, der wörtlich im Sinne von „Muster (wieder-) erkennen“ zu verstehen ist. Diese Namensgebung rührt daher, daß ein Wahrnehmungssystem von seinen Sensoren meistens räumliche und/oder zeitliche Muster geliefert bekommt, z. B. Video- oder Audiodaten, aus denen das System dann die relevante Information extrahieren soll.

Es gibt verschiedenste Ansätze, wie man Mustererkennung mit statistischen Verfahren betreiben kann. Hier seien unter anderem folgende erwähnt: Neuronale Netze (Rumelhart et al., 1986; Personnaz und Dreyfus, 1988), Polynomklassifikatoren (Schürmann, 1996), Support-Vektor-Maschinen (Vapnik, 1995), Hidden-Markov-Modelle (Rabiner, 1989), Observable Operator Models (Jaeger, 1998), endliche Automaten (Breuer, 1995) und vieles mehr, wobei endliche Automaten bis jetzt nur sehr selten zur Mustererkennung eingesetzt worden sind. Wo es erforderlich ist, wird im Text genauer auf die jeweilige Methode eingegangen. Zu dem großen Bereich der konnektionistischen Ansätze zur Mustererkennung kommen wir im folgenden Kapitel.

3 Konnektionistische Ansätze zur Mustererkennung

Die Grundidee konnektionistischer Ansätze ist, daß viele relativ einfache Bausteine zu einem größeren System zusammengefaßt werden. Die vielen Bausteine des Systems wechselwirken miteinander, wodurch das Gesamtsystem trotz der primitiven Bausteine komplexes Verhalten zeigen kann. Die Bausteine des Systems können dabei so einfach wie Boole'sche Funktionen oder so komplex wie mathematische Modelle biologischer Neuronen sein. In der Mustererkennung haben sich unter den konnektionistischen Ansätzen vor allem die neuronalen Netze einen Namen gemacht.

3.1 Neuronale Netze

Neuronale Netze orientieren sich zumindest in ihren Ursprüngen am Vorbild der Natur. Dazu gehört zum einen das Prinzip des Konnektionismus, also der Aufbau eines Systems aus vielen miteinander wechselwirkenden Bausteinen. Bei neuronalen Netzen kommt noch dazu, daß die Bausteine des Systems, die Neuronen, mehr oder weniger stark an biologischen Neuronen ausgerichtet sind. Wegen der ungeheuren Komplexität biologischer Systeme sind diese Modelle meistens nur sehr grobe Annäherungen an ihre biologischen Vorbilder.

Die Anfänge neuronaler Netze reichen weit zurück. Bereits in seinem Buch „Principles of Psychology“ äußerte James (1890) die Vermutung, daß Lernvorgänge „neue Verbindungen im Gehirn erzeugen“ und daß „Nervenströme am leichtesten auf Verbindungen propagieren, die oft benutzt werden“. Fast fünfzig Jahre später schlug Rashevsky (1938) vor, daß das Gehirn auch mit Hilfe von logischen Operationen arbeiten könnte, wenn man hohe Aktionspotentiale als binäre 1 (logisches „wahr“) interpretiert. Erst McCullough und Pitts (1943) stellten die Schwellwertlogik vor und zeigten, daß auch einfache Klassen neuronaler Netze prinzipiell jede arithmetische oder logische Funktion berechnen können. In seinem Buch „The Organization of Behavior“ führte Hebb (1949) mit seiner Hebb'schen Lernregel die unidirektionale Variation der Lernregel von James vor. An dieser Stelle kommt das von Rosenblatt (1958) entwickelte *Perzeptron* ins Spiel, mit dem wir uns noch genauer beschäftigen. Literatur zur Geschichte neuronaler Netze gibt es sowohl als kurze Überblicke (Zell (1994, Abschnitt 1.6) und Olmsted (1998)) als auch als ausführliche Bücher (Anderson und Rosenfeld, 1988; Anderson et al., 1990). An dieser Stelle steigen wir etwas genauer in die neuronalen Netzwerkmodelle ein. Ausführliche Beschreibungen vieler Netzwerktypen sind in (Zell, 1994) zu finden.

3.1.1 Prinzipielle Funktionsweise eines neuronalen Netzes

Welche „Zutaten“ benötigen wir für ein neuronales Netz? Entsprechend seinem biologischem Vorbild sind die Hauptbestandteile eines neuronalen Netzes Neuronen und Verbindungen zwischen den Neuronen. Abbildung 3.1 zeigt ein kleines Beispielnetzwerk. Die Neuronen sind die „Schaltstellen“ des Netzes und führen die arithmetischen Berechnungen durch. Die Neuronen werden über die Verbindungen miteinander verknüpft, so daß die Ergebnisse der Berechnungen

der Neuronen durch das Netzwerk propagieren können, bis sie am Netzwerkausgang ankommen. Um das Netz trainieren zu können, benötigen wir außerem noch einen Lernalgorithmus. Wir stellen nun verschiedene Neuronen- und Netzwerktypen vor.

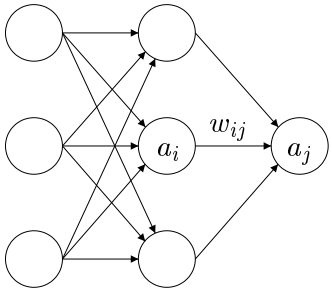


Abbildung 3.1: Ein Beispiel für ein neuronales Netzwerk. Jedes Neuron i hat einen Aktivierungszustand a_i. Eine Verbindung von Neuron i zum Neuron j wird in der Gewichtsmatrix durch das Matrixelement w_{ij} realisiert.

3.1.2 Neuronentypen

Ein Neuron wird durch drei Bestandteile charakterisiert:

- Den Aktivierungszustand a_i. Er gibt an, wie stark das Neuron i aktiviert ist und wird auch *Aktivierungsgrad* genannt.
- Die Aktivierungsfunktion f_a. Die Aktivierungsfunktion bestimmt den Aktivierungsgrad eines Neurons zum Zeitpunkt $t+1$ aus dem Aktivierungsgrad zum Zeitpunkt t:

$$a_i(t+1) = f_\mathrm{a}(a_i(t), n_i(t), \theta_i) \tag{3.1}$$

  Dabei ist $n_i(t)$ die Propagierungsfunktion (Netzeingabe), die angibt, wie sich der Eingang des Neurons i aus den Ausgängen der anderen Neuronen berechnet und θ_i ist Schwellwert des Neurons i.

- Die Ausgabefunktion f_out. Die Ausgabefunktion bestimmt aus der Aktivierung des Neurons die Ausgabe o_i des Neurons:

$$o_i = f_\mathrm{out}(a_i) \tag{3.2}$$

  In der Literatur wird selten zwischen Aktivierungsfunktion, Propagierungsfunktion und Ausgabefunktion unterschieden, oft werden Aktivierungs- und Propagierungsfunktion zusammengefaßt. Ebenso wird oft die Ausgabefunktion gleich der Indentität gesetzt und das gewünschte Ausgangsverhalten eines Neurons gleich mit in die Aktivierungsfunktion integriert.

Verschiedene Neuronentypen lassen sich anhand der Respräsentation ihres Aktivierungszustands unterscheiden, wie Abbildung 3.2 zeigt. Die meisten „klassischen Netzwerkmodelle“ benutzen Neuronen mit reellwertigem Aktivierungszustand, der auf ein Intervall wie $[0,1]$ oder $[-1,1]$ beschränkt ist. Die Beschränkung auf das Intervall $[-1,1]$ ist oft eine Folge der Wahl der Aktivierungs- bzw. Ausgabefunktion. Häufig werden dort die logistische Funktion $f(x) = 1/(1+\exp(-x))$ oder der Tangens Hyperbolicus $f(x) = \tanh(x)$ gewählt, deren Wertebereich gerade dem Intervall $[-1,1]$ entspricht. Zur schnellen Simulation neuronaler Netze in

Ganzzahlarithmetik wurden auch diskrete mehrwertige Neuronen verwendet (Goddard et al. (1989)). Man liest in neuerer Literatur immer wieder, daß die Geschwindigkeit auf modernen Computern keine Rolle mehr spiele, da dort Gleitkomma- und Ganzzahlarithmetik praktisch gleich schnell seinen. Für moderne Hochleistungsprozessoren stimmt das durchaus. Aber wie wir in Abschnitt 1.2.2 schon gesehen haben, gilt dies *nicht* für Hardware, die möglichst klein und kostengünstig ist und in ein Serienprodukt wie z. B. ein Fahrzeug eingebaut werden soll. Auf Netzwerke mit diskreten Neuronen werden wir in Abschnitt 3.2 wieder zurückkommen.

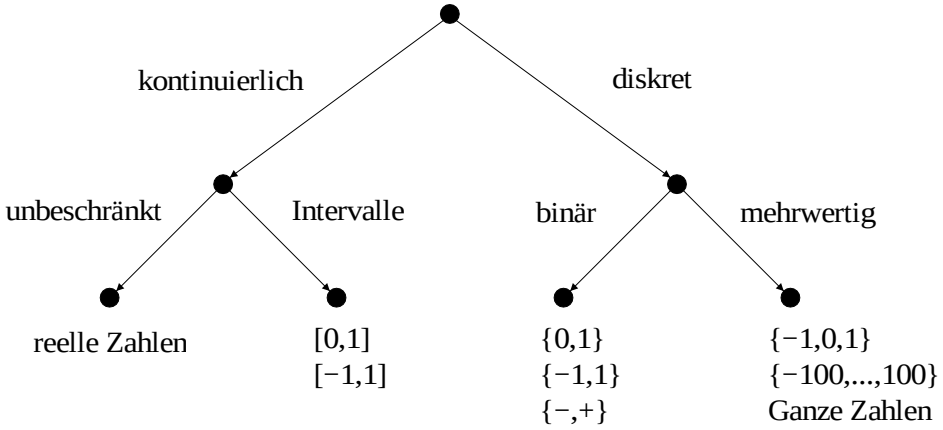


Abbildung 3.2: Realisierungen des Aktivierungszustandes eines Neurons. Zunächst unterscheidet man zwischen Neuronen mit kontinuierlichen und diskreten Aktivierungszuständen. Bei beiden Arten gibt es wiederum unbeschränkte Aktivierungszustände, und solche, die nur innerhalb eines Intervalls definiert sind (aus Zell (1994)).

Ein Neuron, welches in vielen neuronalen Netzen verwendet wird, ist das McCullough und Pitts (1943) Neuron. Seine Aktivierung ergibt sich aus der linear gewichteten Summe seiner Eingänge und seine Ausgabefunktion ist eine Schwellwertfunktion. Dieses Neuron hat „kein Gedächtnis“, da f_{a} nicht von $a_j(t)$ abhängt. Verwendet man für die Ausgabefunktion z. B. die Heavyside-Funktion, nennt man das so entstandene Schwellwert-Neuron auch *Threshold Logic Unit* oder *Linear Threshold Unit (LTU)*.

3.1.3 Netzwerktypen

Die Neuronen als Bausteine eines neuronalen Netzes müssen noch miteinander verbunden werden, was auf viele verschiedene Arten geschehen kann. Die beiden Hauptklassen, die wir unterscheiden, sind Netze ohne Rückkopplungen und Netze mit Rückkopplungen.

Eine Verbindung von einem Neuron i zu einem Neuron j wird formal mit einer Eins in der sogenannten Verbindungsmatrix gekennzeichet, die sonst Nullen enthält. Da zu einer Verbindung im allgemeinen auch ein Gewicht gehört, kann man die Gewichte auch gleich mit in die Verbindungsmatrix eintragen, die damit zur Gewichtsmatrix wird. Dann bezeichnet w_{ij} (bzw. w_{ji}, je nach Literatur) die Stärke des Gewichts an der Verbindung von einem Neuron i zu einem Neuron j. Ist der Eintrag w_{ij} in der Gewichtsmatrix $\boldsymbol{W}$ null, so gibt es keine Verbindung von Neuron i nach Neuron j.

Netzwerke ohne Rückkopplungen

Die am häufigsten verwendete Netzwerktopologie hat nur Verbindungen von einer Netzwerkschicht zur nächsten, die sogenannten ebenenweise verbundenen Netzwerke ohne Rückkopplungen. Wenn wir die Netzwerkschichten vom Eingang zum Ausgang durchnumerieren, sind

also bei einem solchen Netzwerk nur Verbindungen von Schicht k nach Schicht $l = k + 1$ erlaubt. Auch das *Multi-Layer Perzeptron*, auf das wir später noch kommen, ist von diesem Typ. Bei allgemeinen Netzwerken ohne Rückkopplungen sind auch Verbindungen erlaubt, die eine oder mehrere Ebenen überspringen, d. h. $l \geq k + 1$. Diese sogenannten *Shortcut Connection* sind für manche Aufgaben nützlich (z. B. für das XOR-Problem, siehe Abschnitt 10.5 oder das 2-Spiralen-Problem, siehe Lang und Witbrock (1988)). Abbildung 3.3 zeigt die beiden verschiedenen Möglichkeiten.

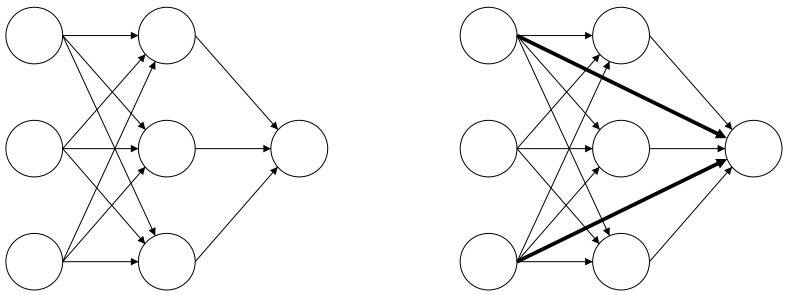


Abbildung 3.3: Neuronales Netzwerk mit und ohne Shortcut Connections. Im linken Netz sind nur Verbindungen von einer Netzwerkschicht zur nächsten erlaubt. Im rechten Netzwerk sind auch zwei Verbindungen, die etwas dicker gezeichneten sog. Shortcut Connections, die eine Netzwerkschicht überspringen.

Netzwerke mit Rückkopplungen

Bei neuronalen Netzen mit Rückkopplungen sind zusätzliche Verbindungen erlaubt. Hierbei unterscheidet man drei Typen von Rückkopplungen:

1. Direkte Rückkopplungen. Ein Neuron hat als einen seiner Eingänge seinen eigenen (gewichteten) Ausgang. Dies sind Verbindungen innerhalb derselben Netzwerkschicht, also $l = k$.

2. Laterale Rückkopplungen. Wie die direkten Rückkopplungen sind dies Verbindungen innerhalb derselben Netzwerkschicht mit $l = k$, aber die Verbindung geht zu einem anderen Neuron derselben Schicht.

3. Indirekte Rückkopplungen. Diese Verbindungen gehen von einer Schicht k zu einer Schicht $l < k$, also zu Neuronen, die sich weiter in Richtung Netzwerkeingang befinden.

Abbildung 3.4 zeigt die verschiedenen Typen von Rückkopplungen. Für ein einzelnes Neuron macht es keinen Unterschied, ob in einem Netzwerk Rückkopplungen existieren oder nicht. Die Netzwerkdynamik ändert sich durch die Rückkopplungen dagegen sehr. Rückgekoppelte neuronale Netze mit reellwertigen Neuronen sind deutlich schwieriger zu beherrschen als neuronale Netze ohne Rückkopplungen. Zum einen ist ihre mathematische Beschreibung komplexer, zum anderen kann man schon mit wenigen reellwertigen Neuronen mathematisch gesehen ein dynamischen System erhalten, das für bestimmte Parameterbereiche chaotisch wird (Pasemann, 1995).

Die erhöhte Komplexität neuronaler Netze mit Rückkopplungen macht sich auch dadurch bemerkbar, daß es für sie deutlich weniger Literatur gibt, als für Netze ohne Rückkopplungen. Besonders die numerische Simulation von Netzwerken mit Rückkopplungen erweist sich als schwierig, da sich auf dem Digitalcomputer keine reellen Zahlen darstellen lassen. Daher ist man zur Näherung mittels Gleitkommazahlen gezwungen, die bei jeder arithmetischen Operation

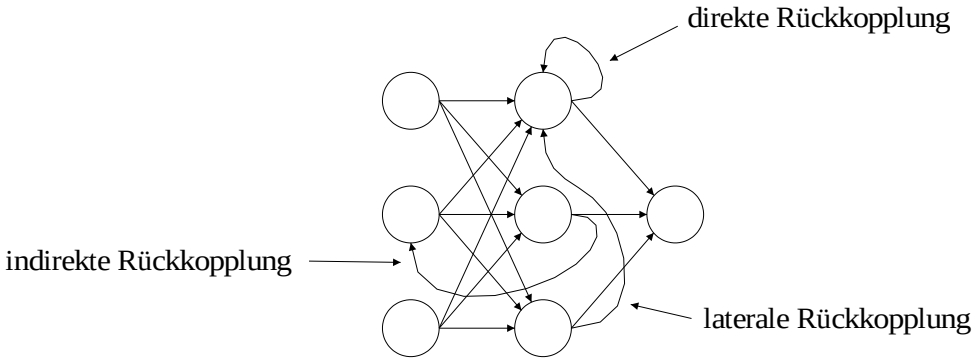


Abbildung 3.4: Neuronales Netzwerk mit Rückkopplungen. Es sind auch Verbindungen innerhalb derselben Schicht bzw. zu Schichten, die näher am Netzwerkeingang liegen, erlaubt.

möglicherweise einen Rundungsfehler verursachen. Möchte man also rückgekoppelte Netzwerke mit reellwertigen Aktivierungszuständen für eine große Zahl von Zeitschritten simulieren, muß man meistens einen erheblichen numerischen Aufwand betreiben.

3.1.4 Mehrschicht-Netzwerke ohne Rückkopplungen

Die in der Praxis mit Abstand am meisten genutzten neuronalen Netzwerke sind rückkopplungsfrei und haben mehrere Schichten, die ebenenweise verbunden sind. Man findet sie meist unter dem Namen *Multi-Layer Network (MLN)*. Ein wichtiger Repräsentant dieser Klasse ist das Multi-Layer Perceptron (MLP), das aus dem von Rosenblatt (1958) entwickelten *Perzeptron* hervorgegangen ist.

3.1.4.1 Das Perzeptron

Wir verstehen hier das *Perzeptron* so, wie es Minsky und Papert (1988) in ihrem Klassiker *Perceptrons* beschrieben haben. Sie untersuchten die Leistungsfähigkeit des Perzeptrons anhand von zweidimensionalen geometrischen Figuren im zweidimensionalen diskreten Raum, in dem jeder Bildpunkt nur die Werte 0 oder 1 annehmen kann.

Ein Perzeptron als neuronales Netz besteht aus folgenden Komponenten:

- Einer Eingabeschicht mit Neuronen, die binäre Aktivierungen haben können.
- Einer weiteren Schicht mit Neuronen, die feste Verbindungen zu den Neuronen der Eingabeschicht haben.
- Der letzten Schicht, die nur ein Ausgabeneuron enthält.

Die Neuronen sind dadurch gekennzeichnet, daß

- sich ihre Eingabe aus der gewichteten Summen ihrer Eingangsverbindungen zusammensetzt

$$a_j = n_j = \sum_i o_i \, w_{ij} \ ,$$

- und daß sich ihre Ausgabe durch eine Schwellwertbildung der Aktivierung ergibt:

$$o_j = \begin{cases} 0 & \text{falls } a_j < \theta_j \\ 1 & \text{falls } a_j \geq \theta_j \end{cases}$$

Abbildung 3.5 zeigt ein Perzeptron und ein Multi-Layer Perzeptron. Wie Minsky und Papert (1988) zeigten, hat das Perzeptron inhärente Schwächen, da es nur wenige Funktionen repräsentieren kann (dazu mehr in Abschnitt 4.5). Die naheliegende Erweiterung des Perzeptrons war, mehrere Neuronenschichten mit trainierbaren Verbindungen einzuführen. Das dreistufige Perzeptron ist das kleinste mehrstufige Perzeptron, welches in n Dimensionen beliebige Mengen voneinander trennen kann, wenn man nur genügend Neuronen zur Verfügung stellt (siehe Zell, 1994) Durch das Hinzufügen weiterer Schichten gewinnt man nichts mehr.

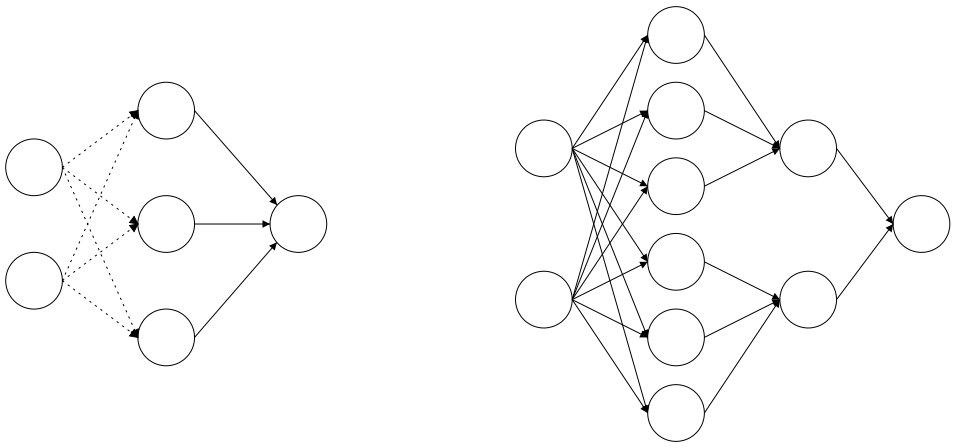


Abbildung 3.5: Die linke Abbildung zeigt ein klassisches Perzeptron, das nur eine Schicht trainierbarer Verbindungen enthält (durchgezogene Linien), die gestrichelten Linien sind feste Verbindungen. Die rechte Abbildung zeigt ein Multi-Layer Perzeptron mit drei trainierbaren Neuronenschichten.

3.1.4.2 Nutzung anderer Ausgabefunktionen

Ist die Ausgabefunktion der Neuronen eine Schwellwertfunktion wie beim Perzeptron, so ist die Ausgabe des Netzwerks in Abhängigkeit der Gewichte weder stetig noch differenzierbar. Beim Training eines neuronalen Netzes möchte man aber oft auf den Gradienten der Fehlerfunktion zurückgreifen, um mit dieser Information die Änderung der Gewichte zu bestimmen. Daher verwendet man häufig für die Ausgabefunktion keine Schwellwertfunktion, sondern beschränkte nichtlineare differenzierbare Funktionen, wie z. B. die logistische Funktion oder der Tangens Hyperbolicus.

Man kann zeigen, daß solche mehrschichtigen Netzwerke (MLNs) sogenannte universelle Approximatoren sind, d. h. sie können, genug Neuronen vorausgesetzt, jede beliebige Funktion beliebig genau approximieren (siehe McCullough und Pitts (1943)). Für die Praxis hilft dies einem allerdings erst mal nicht weiter. Man kennt normalerweise nämlich die Funktion nicht, die approximiert werden soll. Dementsprechend kann man à priori auch keine Netzwerktopologie festlegen, die eine gute Approximation an die noch unbekannte Funktion repräsentieren kann.

3.1.5 Lernverfahren

Schon bei den neuronalen Netzen an sich gibt es eine sehr große Zahl von Variationsmöglichkeiten, wenn man die verschiedenen Netzwerktopologien und Neuronentypen berücksichtigt. Entsprechend gibt es viele verschiedene Lernverfahren, da jeder spezielle Netzwerktyp im allgemeinen mit verschiedenen Trainingsalgorithmen unterschiedlich gut trainiert werden kann. Wir stellen nun die Grundprinzipien vor, nach denen neuronale Netze trainiert werden. Einen

tieferen Einstieg in die Vielfalt der Lernverfahren geben Rumelhart et al. (1986), Zell (1994) und Rojas (1996).

3.1.5.1 Gradientenverfahren zum Training neuronaler Netze

Um ein lernendes System zu trainieren, benötigen wir

1. eine Kostenfunktion, auch Fehlerfunktion genannt, die angibt, wie gut bzw. schlecht das lernende System momentan ist und
2. ein Rezept, nachdem wir die freien Parameter des Systems in Abhängigkeit der Kostenfunktion ändern, so daß die Kosten im Laufe der Zeit minimiert werden.

Soll ein neuronales Netz als Klassifikator betrieben werden, so wählt man typischerweise als Sollausgabe des Netzes für jede Klasse einen Einheitsvektor[1]. Bei drei Klassen erhielten wir also als Soll-Ausgabevektoren $(1,0,0)^{\mathrm{T}}$, $(0,1,0)^{\mathrm{T}}$ und $(0,0,1)^{\mathrm{T}}$.[2] Es ist keineswegs selbstverständlich, daß diese sogenannte Eins-aus-n-Kodierung die beste Ausgangskodierung für ein n Klassen Problem ist. Wie Utschick und Weichselberger (1999) zeigten, kann man durch gleichzeitiges Trainieren eines Netzwerks *und* seiner Ausgangskodierung bessere Klassifikationsergebnisse erhalten, als wenn man eine Ausgangskodierung à priori festlegt.

Als Kostenfunktion E verwendet man meist den quadratischen Abstand, der sich aus der Summe der Abweichungen der Ist-Ausgangsvektoren o'_m zu den Soll-Ausgangsvektoren o_m aller zu lernenden Muster m ergibt:

$$E = \sum_m \frac{1}{2} |o'_m - o_m|^2 \tag{3.3}$$

Nun muß die Kostenfunktion minimiert werden. Im einfachsten Fall ändert man dazu die Gewichte des Netzwerks bei jeder Trainingsiteration ein wenig in Richtung des negativen Gradienten der Kostenfunktion:

$$\Delta w_{ij} = -\eta \frac{\partial}{\partial w_{ij}} E(\boldsymbol{W}) \tag{3.4}$$

Dabei ist η die sogenannte *Lernrate*, je größer die Lernrate ist, desto stärker wird ein Gewicht in Richtung des negativen Gradienten verändert.

Aus den Gleichungen (3.3) und (3.4) läßt sich die sogenannte *Delta-Regel* herleiten (Rumelhart et al., 1986), die für einstufige Netze mit linearer Aktivierungsfunktion bei kleinem η eine sehr gute Approximation an einen Gradientenabstieg ist:

$$\Delta_m w_{ij} = \eta\, o_{mi}\, (o'_{mj} - o_{mj})$$

Hierbei indiziert m das aktuell angelegte Muster, o'_{mj} den Soll- und o_{mj} den Istausgang des Neurons j.

Um nun auch mehrschichtige Netze mit nichtlinearen, differenzierbaren und monoton steigenden Aktivierungsfunktionen trainieren zu können, haben Rumelhart et al. (1986, Seite 324 ff.) die *Generalisierte Delta Regel* hergeleitet, die heute unter dem Namen *Backpropagation* bekannt ist. Beim Backpropagation findet man für die Änderung eines Gewichts w_{ij} zwei verschiedene Aktualisierungsregeln, und zwar in Abhängigkeit davon, ob das Neuron j ein Ausgangsneuron oder ein verstecktes Neuron ist.

[1] Genaugenommen müßten wir hier von *kanonischen* Einheitsvektoren sprechen, da die hier gemeinten Einheitsvektoren nicht nur die Länge Eins haben, sondern auch orthonormal sind und die einzige Komponente ungleich null den Wert eins hat.

[2] $\boldsymbol{x}^{\mathrm{T}}$ bzw. $\boldsymbol{M}^{\mathrm{T}}$ sind die Transponierten eines Vektors $\boldsymbol{x}$ bzw. einer Matrix $\boldsymbol{M}$.

3.1.5.2 Probleme von Gradientenverfahren

Bei Backpropagation hat man genau wie bei anderen gradientenbasierten Lernverfahren mit bestimmten Schwierigkeiten zu kämpfen:

- Lokale Minima: Bei einem Gradientenverfahren besteht immer die Gefahr, daß der Lernalgorithmus in einem lokalen Minimum terminiert. Je größer das Netzwerk, desto größer ist dafür im allgemeinen die Wahrscheinlichkeit (Hecht-Nielsen, 1989).
- Flache Plateaus: Befindet man sich im Parameterraum auf einer „Hochebene“ ist der Gradient sehr klein, obwohl das gesuchte globale Minimum möglicherweise noch weit entfernt ist. Da der Gradient fast null ist, kann man ohne weiteres auch nicht erkennen, ob man sich in einem Minimum oder auf einem Plateau befindet.
- Steile und enge Täler: Ist man im Parameterraum in einer engen Schlucht, so können zwei unangenehme Phänomene auftreten: Zum einen kann es zu Oszillationen zwischen den beiden Seiten der Schlucht kommen, zum anderen kann wegen des großen Gradienten an den Seiten der Schlucht der Lernalgorithmus die steile Schlucht auch zugunsten eines weniger guten Minimums wieder verlassen.

Es ist nicht einfach, diese Schwierigkeiten gleichzeitig in den Griff zu bekommen. Im Laufe der Zeit haben viele Forscher daran gearbeitet, durch die Modifikationen von Backpropagation und durch die Entwicklung von ganz anderen Lernverfahren diese Schwierigkeiten zu überwinden. Einige solcher Verfahren wie werden in Zell (1994) vorgestellt

3.2 RAM-basierte neuronale Netze

Bis jetzt haben wir uns mit Netzen beschäftigt, deren Neuronen kontinuierliche Aktivierungszustände annehmen können. Nun wenden wir uns Neuronentypen zu, deren Aktivierungszustand nur diskrete, oft nur binäre Werte annehmen kann. *RAM-basierte neuronale Netze*, die auch *gewichtslose neuronale Netze* genannt werden, gehen bis auf die Arbeit von Bledsoe und Browning (1959) (bzw. 1966) zurück, die die sogenannte *n-Tupel-Methode* einführten. Wir werden nun einige Varianten RAM-basierter neuronaler Netze kennenlernen, die zur Mustererkennung eingesetzt werden. Relativ zu klassischen neuronalen Netzen gibt es zu RAM-basierten neuronalen Netzen wenig Literatur, trotzdem gibt es auch hier wichtige Referenzen: Alexander (1989), Gurney und Wright (1992), Austin (1994) und Austin (1998).

3.2.1 Die n-Tupel-Methode

In der ursprünglichen Arbeit von Bledsoe und Browning ging es darum, auf einer zweidimensionalen Bildmatrix mit binären Pixeln Buchstaben zu erkennen. Ihre Idee war, die betrachteten Pixel des Bildes mit logischen Funktionen auszuwerten. Eine logische Funktion faßte Gruppen von Pixeln, die Tupel, zu einem Minterm zusammen. Jeder Klasse von Buchstaben war ein Satz von logischen Funktionen zugeordnet. Um z. B. den Buchstaben ‘T’ vom Buchstaben ‘C’ in Abbildung 3.6 zu unterscheiden, kann man die beiden logischen Funktionen

$$\begin{aligned} F_{\mathrm{T}} &= p_0\, p_1\, p_2 + \bar{p_3}\, p_4\, \bar{p_5} + \bar{p_6}\, p_7\, \bar{p_8} \\ F_{\mathrm{C}} &= p_0\, p_1\, p_2 + p_3\, \bar{p_4}\, \bar{p_5} + p_6\, p_7\, p_8 \end{aligned} \tag{3.5}$$

verwenden. Die Pixel $p_i \in \mathbb{B} = \{0, 1\}$ sind dabei von links oben nach rechts unten zeilenweise durchnumeriert und die drei Tupel bestehen aus (p_0, p_1, p_2), (p_3, p_4, p_5) und (p_6, p_7, p_8). Um

der n-Tupel-Methode zu einer Generalisierungsfähigkeit zu verhelfen, werden die Minterme der logischen Funktionen nicht mit einem AND verknüpft, sondern addiert. Dadurch liefert F_{T} bei einem Bild, was sich nur um ein Pixel von einem 'T' unterscheidet, nicht null sondern zwei als Ausgabe. Durch die lokale Beschränkung der Tupel wird erreicht, daß auch unbekannte Bilder *lokal* einem trainierten Bild gleichen können und damit *global* ähnlich sind. Um nun ein unbekanntes Bild zu klassifizieren, werden die Ausgaben der zu einer Klasse gehörenden logischen Funktionen aufsummiert und diejenige Klasse gewählt, bei der diese Summe am größten ist.

Abbildung 3.6: Das Pixelbild von 'T' und 'C'.

Aleksander und Stonham (1979) führte nun für jedes Tupel einen RAM-Knoten (RAM-Node) ein, der in Abbildung 3.7 gezeigt ist. Dieser RAM-Knoten entspricht gerade einem Minterm aus Gleichung (3.5), jede Kombination der n Eingangsbits eines Tupels wird am Ausgang auf 0 oder 1 abgebildet. Jedem Tupel im Eingangsbild werden nun RAM-Knoten zugeordnet, die eine von k Klassen erkennen sollen. Die zu einer Klasse gehörenden RAM-Knoten werden zu einer Einheit, einem sogenannten Diskriminator (auch *Multi RAM Discriminator (MRD)*) zusammenfaßt. Die Ausgabe eines Diskriminators ist gerade die Summe der Einsen, die sich für ein Bild aus seinen RAM-Knoten ergeben hat. Ein unbekanntes Bild wird dann derjenigen Klasse zugeordnet, deren Diskriminator die größte Ausgabe hatte. Abbildung 3.8 zeigt eine beispielhafte Anordnung. Diese Typen von RAM-Netzen wurden vor allem von Igor Aleksander und seiner Gruppe untersucht (Aleksander und Stonham, 1979; Aleksander, 1982). Die Arbeiten mündeten in dem System *WISARD*, einem schnellen in Hardware realisierten Bilderkennungssystem für industrielle Anwendungen (Aleksander et al., 1984; Alexander, 1985). Aufbauend auf WISARD untersuchten Austin und Stonham (1987) RAM-Netze als assoziative Speicher, die auch in Hardware implementiert wurden (Smith und Austin, 1992; Austin und Buckle, 1994).

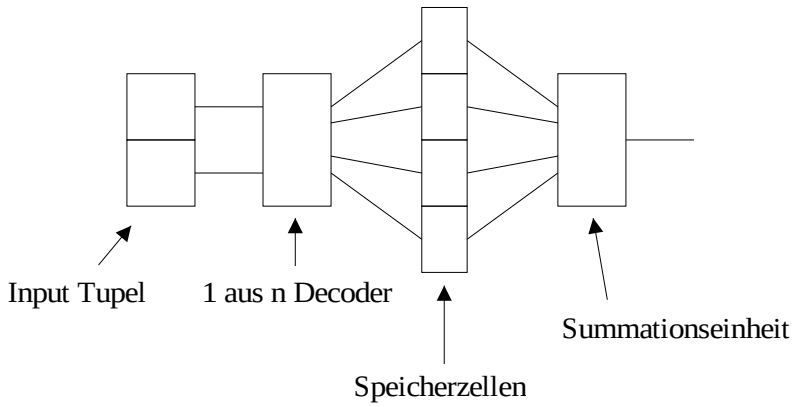


Abbildung 3.7: Ein RAM-Knoten. Je nach Zustand der Eingangstupel wird eine der vier Speicherzellen ausgewählt und deren Inhalt über die Summationseinheit auf den Ausgang gelegt. Die Einheit vom Decoder bis zur Summationseinheit entspricht einem Speicherbaustein (RAM-Baustein).

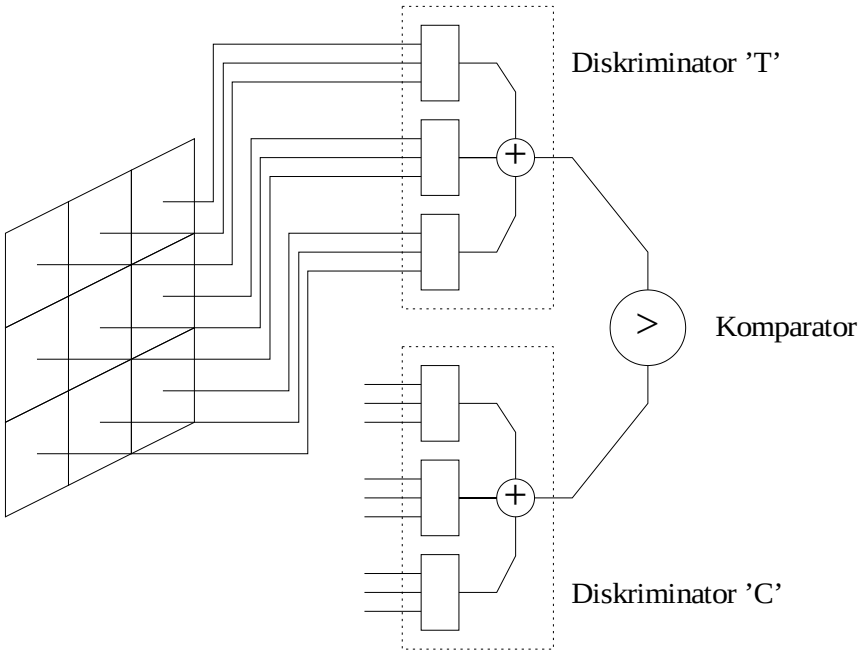


Abbildung 3.8: Ein einstufiges RAM-Netz. Für jede zu erkennende Klasse gibt es einen Multi-RAM-Diskriminator, der die Ausgänge seiner enthaltenen RAM-Bausteine summiert. Das Bild wird dann der Klasse zugeordnet, deren Diskriminator die größte Antwort geliefert hat.

Das Training eines Diskriminator-RAM-Netzes ist recht einfach: Jedes zu lernende Muster einer bestimmten Klasse wird an den Eingang des Netzwerks angelegt. Nun wird in allen RAM-Knoten des zur Klasse gehörenden Diskriminators die durch das Muster aktivierten Speicherzellen auf Eins gesetzt. Da jedes zu lernenden Muster nur einmal trainiert werden muß, ist diese Art von Training sehr schnell, man spricht daher auch von *One Shot Training*.

Genau wie bei kontinuierlichen neuronalen Netzen haben einstufige RAM-Netze Limitierungen, denn sie können nur einen kleinen Teil aller möglichen logischen Funktionen ihrer Eingangsbits repräsentieren. Um diese Einschränkung hinter sich zu lassen, ist man auch bei RAM-Netzen zu mehrschichtigen Netzwerken übergegangen.

3.2.2 Multi-Layer RAM-Netze

Mehrschichtige RAM-Netzwerke sind eine natürliche Erweiterung einschichtiger Netzwerke. Anstatt, wie bei einschichtigen Netzen, den Ausgang der ersten Schicht von RAM-Koten direkt auszuwerten, wird nun eine zweite Schicht von RAM-Knoten hinter die erste geschaltet. Sie hat als Eingänge die Ausgänge der Vorgängerschicht. Durch die Nachschaltung weiterer Stufen hinter die erste Stufe können nun auch logische Funktionen der Ausgangsbits der ersten Schicht gebildet werden, was vorher nicht möglich war. Ein typisches mehrschichtiges RAM-Netzwerk, das auch *Multi-Layer RAM-Netz (MLRN)* oder *RAM-Pyramide* genannt wird, zeigt Abbildung 3.9.

Durch die Erweiterung auf mehrere Schichten erhält man aber nicht nur ein leistungsfähigeres Netzwerk, man muß auch ein neues Lernverfahren suchen, um die RAM-Knoten in den Zwischenschichten zu trainieren. Im Gegensatz zu kontinuierlichen Netzen kann man hier aber nicht die generalisierte Delta-Regel verwenden, da die RAM-Knoten keine differenzierbaren Funktionen realisieren. Dieses Problem hat eine weitere Art von RAM-Netzen hervorgebracht,

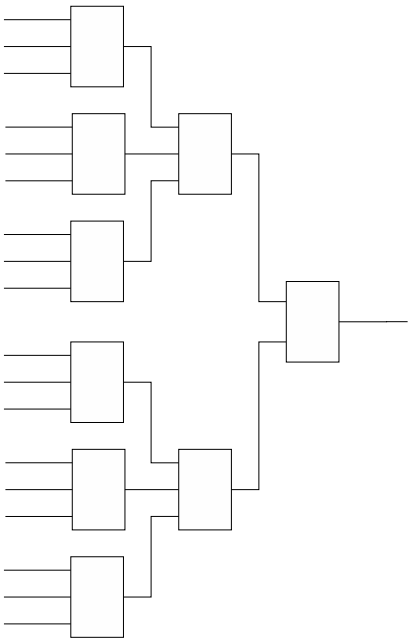


Abbildung 3.9: Ein mehrstufiges RAM-Netz. Die Ausgänge jeder Schicht von RAM-Knoten sind die Eingänge für die nächste Schicht.

und zwar die probabilistischen RAM-Netze. Mit ihnen lassen sich manche Probleme des Trainings eines mehrschichtigen RAM-Netzwerks lösen.

3.2.3 Probabilistische RAM-Netze

Nachdem man bei RAM-Netzen nicht einfach eine Fehlerfunktion minimieren kann, hat es verschiedene Ansätze gegeben, Multi-Layer RAM-Netze mit *Reinforcement Learning* zu trainieren. Reinforcement Learning ist ein Lernverfahren, bei dem das zu trainierende System nur übermittelt bekommt, ob seine Ausgabe für das zu lernende Muster richtig oder falsch war. Diese Information wird jedem RAM-Knoten übermittelt, der daraufhin für das lokal gesehene Muster eine Eins oder Null speichert.

Nun benötigt aber ein RAM-Knoten für das Reinforcement Learning nicht nur zwei, sondern drei verschiedene Zustände:

1. Das Eingangsmuster erzeugte eine richtige Ausgabe.
2. Das Eingangsmuster erzeugte eine falsche Ausgabe.
3. Das Eingangsmuster ist noch nicht aufgetreten.

Den zusätzlich benötigten Zustand realisierte Catherine Myers (1988) beim Training, indem ein RAM-Knoten in einem Zustand 'u' (undefined) war, wenn der Knoten ein Eingangsmuster noch nicht gesehen hatte. Wurde nun ein 'u'-Zustand eines Knotens addressiert, gab der Knoten eine 1 oder 0 mit der Wahrscheinlichkeit 1/2 aus. Dadurch war gewährleistet, daß das Netzwerk bei jedem Eingangsmuster einen gültigen Ausgangswert annahm. Diese Art von Knoten wird *Probabilistic Logic Node (PLN)* genannt.

Ein Nachteil der Probabilistic Logic Node ist, daß man nicht erkennen kann, wie oft ein Muster an einem Knoten aufgetreten ist. Die möglichen Eingangsmuster eines Knoten werden

also alle gleich stark gewichtet, auch wenn sie in den Trainingsdaten unterschiedliches Gewicht haben. Um diesen Nachteil zu vermeiden, benutzten Smith und Austin (1992) während des Trainings ein Verfahren, das die Eingangsmuster der verschiedenen Knoten gewichtete. Für die spätere Implementierung in Hardware konvertierten sie die Gewichte in eine binäre Repräsentation. Myers (1992) erweiterte die drei möglichen Zustände eines PLN auf w Zustände und trainierte dieses *w-PLN* mit Reinforcement Learning. Er zeigte, daß sich durch die Erweiterung der Zustandszahl die Erkennungsrate verbessern ließ.

Gorse und Taylor (1989) erweiterten den RAM-Knoten probabilistisch und nannten ihn *Probabilistic RAM (pRAM)*. Im Gegensatz zum PLN speichert ein pRAM nicht nur, ob ein Muster am Knoteneingang innerhalb einer Klasse überhaupt vorgekommen ist, sondern mit welcher Wahrscheinlichkeit ein Muster in einer Klasse aufgetreten ist. Die zu einem Eingangsmuster gehörenden Wahrscheinlichkeiten werden dann im Netz von Knoten zu Knoten weitergereicht. Dadurch wird es möglich, die Wahrscheinlichkeit für die Zugehörigkeit eines bestimmten Musters zu einer Klasse am Netzwerkausgang zu berechnen. Um die kontinuierlichen Wahrscheinlichkeiten in digitaler Hardware zu implementieren, nutzen Gorse und Taylor (1989) eine Pulscodierung. Wird eine Speicherzelle adressiert, gibt der Knoten in Abhängigkeit der gespeicherten Wahrscheinlichkeit eine Null oder Eins auf seiner Ausgangsleitung aus. Auf diese Weise hat der Strom von Nullen und Einsen gerade den Erwartungswert der gespeicherten Wahrscheinlichkeit. Dieses probabilistisch arbeitende Netzwerk wird mit Reinforcement Learning trainiert.

In dem Netzwerkmodell von Filho et al. (1990, 1991, 1992) wurde die Probabilistic Logic Node dahingehend erweitert, daß ein Knoten nun neben einem binären Ausgangswert auch den 'u'-Zustand (undefined) ausgeben konnte. Ein 'u' auf einer Leitung adressierte dann zwei Speicherzellen des nachfolgenden Knotens, also z. B. 1u0 adressierte 100 und 110. Dieses sogenannte *Goal-Seeking Neuron (GSN)* wurde nicht mehr mit Hilfe des Reinforcement Learnings trainiert, sondern mit einem Verfahren, daß der Tiefensuche mit Backtracking ähnelt. Dieses Verfahren muß zwar jedes Trainingsmuster nur einmal präsentiert bekommen, durch die Tiefensuche kann aber das Training eines einzelnen Musters im schlimmsten Fall recht lange dauern.

Neben diesen Grundtypen von deterministischen und stochastischen RAM-basierten Netzwerken gibt es noch weitere Typen von rückkopplungsfreien Netzen, die sich durch den Knotentyp, die Netzwerktopologie oder das Trainingsverfahren voneinander unterscheiden. Sie sind aber konzeptionell von den später vorgestellten Hyperpermutationsnetzwerken recht weit entfernt, so daß wir hier nicht weiter darauf eingehen.

3.2.4 Rückgekoppelte RAM-Netze

Auch bei RAM-basierten neuronalen Netzwerken gibt es Architekturen, die mit Rückkopplungen arbeiten. Sie haben gegenüber Netzwerken mit kontinuierlichen Neuronen den Vorteil, daß bei deterministisch arbeitenden Knoten kein chaotisches Verhalten auftreten kann. Das bedeutet noch nicht, daß das Training rückgekoppelter RAM-Netz einfach wäre. Aber wir wissen zumindest für den Betrieb solcher Netze, daß es nur abzählbar viele Zustände gibt, in denen sich das Netzwerk befinden kann.

Einen wichtiges Potential rückgekoppelter RAM-Netze spricht Aleksander (1998, Seite 21) an:

> The real promise for neural networks, particularly weightless ones, may be in areas where knowledge storage is required – that is, recursive systems.

Rückgekoppelte RAM-Netze besitzen interne Zustände, mit denen sie sich Information merken können. Dies ist bei jeglicher Art von zeitlichen Eingangsdaten interessant, da ein solches Netz-

werk Eingangsdaten aus verschiedenen Zeitschritten miteinander korrelieren kann. Trotzdem handeln sich RAM-basierte Netze nicht die Schwierigkeiten wie Instabilität oder chaotisches Verhalten ein, die bei kontinuierlichen Netzen mit Rückkopplungen auftreten können.

Die einfachsten Knotentypen realisieren Boole'sche Funktionen wie z. B. das XOR. Solche *Boole'schen Netze* sind sowohl mit deterministisch als auch stochastisch arbeitenden Knoten untersucht worden (Kauffman (1969); Walker (1990)). Oft kam dabei das Interesse aus der statistischen Physik, deren Methoden sich auf Netzwerke mit vielen gleichartigen Knoten anwenden lassen. Es sind aber auch Versuche unternommen worden, mit Boole'schen Netzen Mustererkennung zu betreiben (Martland, 1987).

Es wurden auch kompliziertere Neuronentypen in Netzen mit Rückkopplungen verwendet, wie z. B. das schon erwähnte Goal-Seeking Neuron (GSN). Filho et al. (1992) schlugen eine Architektur vor, bei der das Netz aus RAM-Pyramiden aus GSNs besteht. Die Rückkopplungen wurden dadurch realisiert, daß die Ausgaben mehrerer RAM-Pyramiden mit den neuen Eingabedaten verknüpft wieder in das Netz gegeben wurden. Um mit solchen Netzen arbeiten zu können, muß man die Boole'schen Funktionen erweitern, da das GSN nicht nur 0 oder 1, sondern auch den Wert 'u' emittieren kann.

In dem System *MAGNUS* von Aleksander et al. (1993) (auch Aleksander, 1998) ist das einzelne Neuron durch eine noch mächtigere Struktur, die sog. *General Neural Unit (GNU)* ersetzt worden. Aleksander nennt die General Neural Unit auch *Neural State Machine*. Eine General Neural Unit ist nicht mehr ein einzelnes Neuron, sondern eine Gruppe von Neuronen, die die Eingangsbits des GNU auswerten. Die Neuronen sind auch untereinander verbunden, so daß sich eine GNU sowohl wie ein einschichtiges neuronales feed-forward Netz als auch wie ein Hopfield-ähnliches autoassoziatives Netz verhalten kann. Im *MAGNUS*-System werden nun solche mächtigen „Neuronen" wie die GNUs zu einem Netzwerk verknüpft.

Da rückgekoppelte RAM-Netze im allgemeinen viel schwieriger zu trainieren sind als rückkopplungsfreie RAM-Netze, sind rückgekoppelte RAM-Netze kaum zur Mustererkennung eingesetzt worden. Nur sehr einfache Aufgaben wurden untersucht, wie z. B. die Unterscheidungs eines roten vertikalen Rechtecks auf blauem Hintergrund von einem grünen horizontalen Rechteck auf blauem Hintergrund (Aleksander, 1998). Gelingt es aber, ein rückgekoppeltes Netz zu trainieren, so kommt das gefundene Netz für eine ähnliche Lösung oft mit weniger Ressourcen aus als sein rückopplungsfreies Äquivalent. Es gibt noch diverse andere Architekturen für rückgekoppelte neuronale RAM-Netze. Wir gehen auf diese Varianten nun nicht mehr ein sondern konzentrieren uns in Abschnitt 4.5 auf die Unterschiede zwischen Hyperpermutationsnetzwerken und schon bekannten Ansätzen.

4 Hyperpermutationsnetzwerke

Was ist ein Hyperpermutationsnetzwerk (HPN) und inwiefern ist es anders als die im vorigen Kapitel vorgestellten Ansätze? Um eine Vorstellung von Hyperpermutationsnetzwerken (HPNs) zu bekommen, beschreiben wir ihre Funktionsweise erst einmal informell. Danach führen wir das HPN formal ein, besprechen seine Eigenschaften und vergleichen es mit bekannten mathematischen Modellen.

4.1 Informelle Beschreibung eines Hyperpermutationsnetzwerks

Ein HPN besteht aus Knoten und Kanten. Die Kanten sorgen für den Datentransport vom Eingang des Netzes hin zu den Knoten und weiter zum Ausgang des Netzes. Die Knoten lesen die an ihren Eingängen liegenden Daten, transformieren sie und schreiben die transformierten Daten auf die Knotenausgänge. Die Topologie eines HPN hat unter anderem zwei wichtige Merkmale:

- Jeder Knoten hat genausoviele Eingangs- wie Ausgangskanten. Die Zahl der Eingangs- bzw. Ausgangskanten eines Knotens nennen wir die *Dimension* des Knotens oder *Knotendimension*.
- Kanten können sich nicht verzweigen.

Ein HPN hat *interne Kanten*, die die Daten innerhalb des Netzes transportieren. Ebenso gibt es *Eingangs- und Ausgangskanten*, die Daten in das Netz hinein und wieder aus dem Netz heraustransportieren. Ein HPN mit n Eingangs- bzw. Ausgangsleitungen nennen wir ein HPN der Dimension n. Eine mögliche Topologie zeigt Abbildung 4.1.

4.1.1 Netzwerkdynamik

Ein HPN ist ein *synchron getaktetes dynamisches System*, die Dynamik des Netzes verläuft in diskreten Zeitschritten $t, t+1, t+2, \ldots$. Jeder Kante wird eine binäre Zustandsvariable zugeordnet. Die Zustandsvariablen sind „digital“, d. h. sie können genau zwei verschiedene Zustände annehmen: 1 (wahr) oder 0 (falsch). Daher nennen wir die Kanten auch *Leitungen* im Sinne einer Digitalschaltung. Wir beschreiben die Werte der Leitungen durch die Variablen $x_i \in \mathbb{B} = \{0, 1\}$. Die Knoten selbst enthalten keine Zustandsinformation, sie operieren nur auf den Leitungen. Hat ein HPN n Leitungen, so ist sein Zustand durch die Angabe der Werte aller Leitungen zu einem bestimmten Zeitpunkt festgelegt. Ein Knoten mit k Eingangsleitungen „beobachtet“ an seinem Eingang k Bit lange Eingangsmuster. Sie entstehen, indem die aktuellen Werte der Eingangsleitungen zu einem Bitstring aneinandergereiht werden. Ein Bitstring bzw. Binärstring ist eine Folge von Nullen und Einsen. Bei k Bit langen Bitstrings gibt es 2^k verschiedene Eingangsmuster, bei einem Knoten der Dimension zwei beispielsweise die vier Muster $\{00, 01, 10, 11\}$.

Was verbirgt sich nun in den Knoten? In jedem Knoten mit k Eingangsleitungen steht eine Tabelle der Länge 2^k, die jedem möglichem Eingangsmuster ein bestimmtes Ausgangsmuster

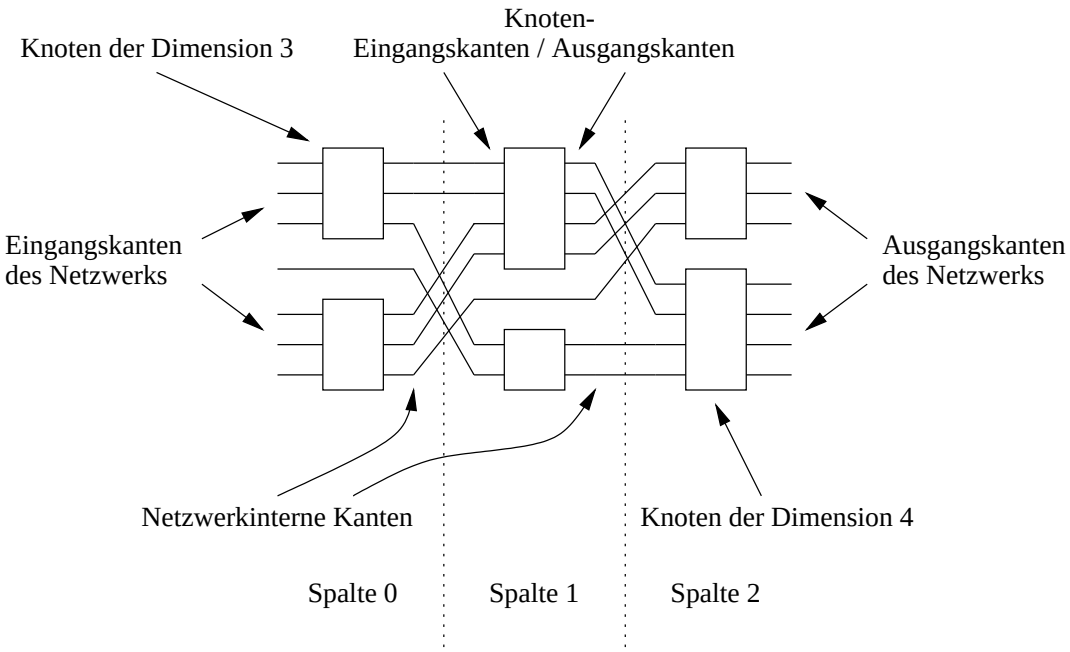


Abbildung 4.1: Ein Beispiel für ein Hyperpermutationsnetzwerk der Dimension 7. Wir visualisieren ein HPN so, daß der Netzwerkeingang links und der Netzwerkausgang rechts ist. Der Eingang jedes Knotens ist auf der linken Knotenseite und der Ausgang auf der rechten Seite. Anschaulich gesprochen „laufen die Daten von links nach rechts durch das Netzwerk."

zuordnet, z. B. für $k = 2$

Eingangsmuster	Ausgangsmuster
0 0	1 1
0 1	1 0
1 0	0 1
1 1	0 0

Wichtig ist, daß diese Zuordnung *eindeutig* und *umkehrbar*, also *bijektiv* ist. Bei jedem Zeittakt des Netzes liest nun jeder Knoten das durch seine Eingangsleitungen definierte Eingangsmuster, schlägt in seiner Tabelle das zugehörige Ausgangsmuster nach und schreibt dieses auf seine Ausgangsleitungen. Die Ausgangsleitungen dieses Zeittakts werden im nächsten Takt für die nachfolgenden Knoten zu Eingangsleitungen. Auf diese Weise werden die am Eingang des Netzes hereinkommenden Daten durch die Knoten lokal und umkehrbar transformiert und werden sozusagen bei jedem Takt einen Knoten „weitergereicht". Das zu einem Eingangsmuster gehörende Ausgangsmuster kommt also je nach Zahl der Knoten zwischen Netzein- und ausgang erst nach einigen Takten wieder aus dem Netz heraus.

Da die lokalen Transformationen umkehrbar sind, ist auch die globale Transformation eines Eingangsmusters durch das Netzwerk umkehrbar. Wie später gezeigt wird, ist diese Eigenschaft keine wirkliche Beschränkung, da wir *un*umkehrbare Transformationen in umkehrbare einbetten können. Die Umkehrbarkeit der Transformation hat aber den Vorteil, daß durch die Transformation zunächst keine Information verloren geht. Am Ausgang des Netzes ist noch alle Information vorhanden, nur liegt sie in einer Darstellung vor, die die gesuchten Aspekte der Daten im Rahmen einer Wahrnehmungsaufgabe einfacher zugänglich macht. Ein HPN mit n Eingangs- und Ausgangsleitungen transformiert also Bitstrings aus $\mathbb{B}^n$ reversibel in Bitstrings

aus $\mathbb{B}^n$.

4.1.2 Ziel der Transformation

Welchen Zweck verfolgen wir mit der Transformation? Stellen wir uns vor, wir wollten n Bit lange Sensordaten nach bestimmten Gesichtspunkten ohne Rückweisung in zwei verschiedene Klassen 0 und 1 unterteilen. Bei jedem Bitstring aus dem Sensor können wir fragen „Gehört dieser Bitstring zur Klasse 0?“ Die Antwort auf diese Frage lautet entweder „Ja“ oder „Nein“, und wenn der Bitstring nicht zu Klasse 0 gehört, gehört er zu Klasse 1. Wir brauchen also nur ein Bit („Ja“ oder „Nein“), um die Klassenzugehörigkeit eines Bitstrings zu kodieren. Wie gewinnen wir aber das benötigte Bit aus dem n Bit langen Bitstring? Denn das „gesuchte Bit“ ist ja im allgemeinen nicht ein bestimmtes Bit des Bitstrings, sondern die gesuchte Information vom Umfang eines Bit ist irgendwie über die n Bit des Bitstrings verteilt.

Betrachten wir dazu zwei Beispiele:

1. Ein System soll Groß- und Kleinbuchstaben in zwei verschiedene Klassen einteilen.

2. Ein System soll Vokale und Konsonanten in zwei verschiedene Klassen einteilen.

Da ein Digitalcomputer mit Buchstaben als solches nichts anfangen kann, müssen wir die Buchstaben in einer digitalcomputer-gerechen Weise repräsentieren, d. h. als Bitstrings. Wir benutzen hier den ASCII-Code, mit dem die erste Aufgabe sehr einfach wird. Die Codes einiger Buchstaben zeigt Tabelle 4.1.

Buchstabe	ASCII-Code	Buchstabe	ASCII-Code
'a'	1100001	'A'	1000001
'e'	1100101	'E'	1000101
'i'	1101001	'I'	1001001
'b'	1100010	'B'	1000010
'k'	1101011	'K'	1001011
'v'	1110110	'V'	1010110

Tabelle 4.1: ASCII-Codes einiger Buchstaben

Wie wir sehen, unterscheiden sich die Bitmuster eines Groß- und Kleinbuchstaben in genau einem Bit, dem Bit Nummer 5. Die Unterscheidung in Groß- und Kleinbuchstaben ist also dadurch zu lösen, daß wir bei einem Buchstaben das Bit Nummer 5 auswerten, ist es 0 handelt es sich um einen Großbuchstaben, ist es 1 handelt es sich um einen Kleinbuchstaben. Die zweite Aufgabe, die Unterscheidung in Vokale und Konsonanten ist dagegen nicht so einfach, da die gesuchte Information nicht schon an einem einzelnen Bit „fertig bereitliegt“, sondern über mehrere Bits verteilt ist. Wir müssen das Bit, was die gesuchte Information trägt erst noch durch eine Transformation generieren.

Die beschriebene Situation ähnelt stark einer Standardsituation in der Mathematik: Man hat ein Gleichungssystem in n Variablen, es gibt aber nur eine unabhängige Variable, die anderen $n-1$ Variablen sind abhängige Variable. In diesen Fall führt man eine Koordinatentransformation durch, damit die $n-1$ abhängigen Variablen zu Konstanten werden und die eine unabhängige Variable übrig bleibt.

Genau dasselbe Ziel hat die Transformation eines Bitstrings mit einem HPN. Das Koordinatensystem des $\mathbb{B}^n$ soll so transformiert werden, daß die gesuchte Information nicht mehr über viele Bit verstreut ist, sondern an möglichst wenigen Bit zugänglich wird (siehe Abbildung

4.2). Ein HPN realisiert eine solche globale Transformation, indem es mehrere lokale Transformationen durchführt. Wie wir die lokalen Transformationen, also die Knotentabellen, für eine bestimmte Aufgabe finden, ist Gegenstand von Teil 2 (Optimierung).

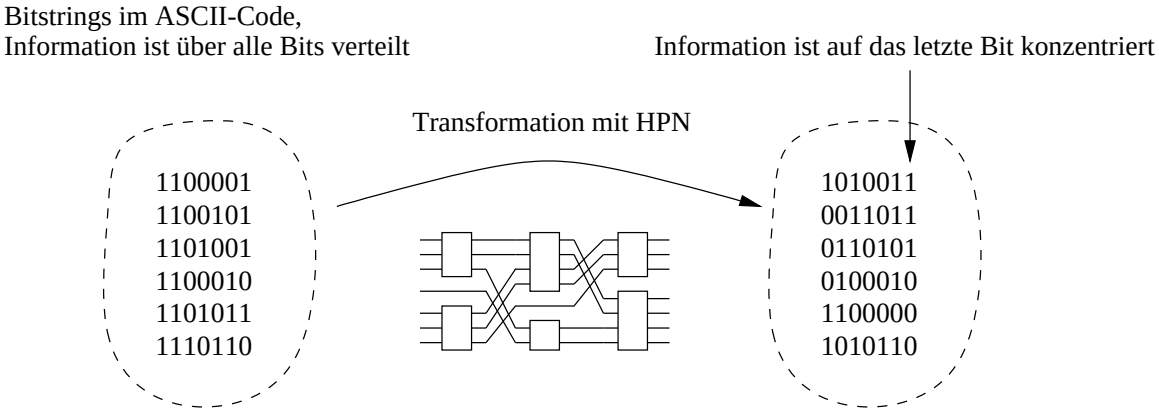


Abbildung 4.2: Transformation von ASCII-Codes mit einem HPN. Die Bitstrings auf der linken Seite sind im ASCII-Code. Auf der rechten Seite sind die transformierten Bitstrings. Hier ist die Aufgabe, die Vokale und Konstanten voneinander zu trennen. Die uns interessierende Information ist auf das letzte Bit der transformierten Bitstrings konzentriert worden. Das letzte Bit ist 1, wenn der Bitstring auf der linken Seite einen Vokal repräsentiert bzw. 0, wenn der Bitstring auf der linken Seite einen Konstant repräsentiert.

4.1.3 Ein HPN als Klassifikatior

Wie können wir ein HPN als Klassifikatior betreiben? Nehmen wir an, wir hätten ein fertig trainiertes HPN H mit n Ein- und Ausgangsleitungen, das die gesuchte Information wie in Abbildung 4.2 auf $m < n$ Ausgangsleitungen verdichtet. Zum Training hatten wir einen Lerndatensatz $\mathcal{L}$, der k verschiedene Klassen enthält. Wenn wir jetzt nur die m informationstragenden Ausgangsleitungen betrachten und damit in den $\mathbb{B}^m$ Unterraum des $\mathbb{B}^n$ Ausgangsraums projizieren, transformiert das HPN nun Bitstrings aus dem $\mathbb{B}^n$ Eingangsraum in den $\mathbb{B}^m$ Ausgangs(unter)raum. Am Eingang des HPNs können jetzt 2^n und am betrachteten Ausgang 2^m verschiedene Bitstrings vorkommen. Die Zahl der Klassen k muß kleiner als 2^m sein, da sonst der Ausgangs(unter)raum des HPN die k verschiedenen Klassen nicht repräsentieren kann.

Transformieren wir nun ein Element l der Trainingsdaten mit der Klasse k_0, erhalten wir ein Ausgangssymbol, von dem wir wissen, daß das zugehörige Eingangssymbol zur Klasse k_0 gehört. Auf diese Weise transformieren wir die gesamten Trainingsdaten und protokollieren in der sogenannten Statistikmatrix, welche Ausgangssymbole und welche Klassen gemeinsam auftreten. Die Statistikmatrix hat also 2^m Zeilen und k Spalten. Nachdem wir den Trainingsdatensatz ausgezählt haben, können wir für jeden Ausgangszustand des HPN die Rückschlußwahrscheinlichkeit für die k Klassen angeben. Nach der Bayes-Regel haben wir genau dann den kleinsten Fehler, wenn wir uns jeweils für die Klasse mit der größten Rückschlußwahrscheinlichkeit entscheiden.

Liegt nun ein *unbekanntes* Symbol am Netzwerkeingang, wird es vom HPN in einen der möglichen Ausgangszustände transformiert. Wir ordnen dem unbekannten Muster dann diejenige Klasse zu, die nach der Trainingsstatistik die höchste Rückschlußwahrscheinlichkeit hat. Haben wir beispielsweise ein Zwei-Klassen-Problem, einen Trainingsdatensatz mit 1000 Elementen und ein HPN mit zwei Ausgangsbits, könnten zwei mögliche Statistikmatrizen so wie

in Tabelle 4.2 aussehen.

HPN 0	Klasse		Entscheidung	Rückschlußw.
Zustand	0	1		
00	122	45	0	73.0%
01	12	243	1	95.3%
10	376	78	0	82.8%
11	51	73	1	58.9%

HPN 1	Klasse		Entscheidung	Rückschlußw.
Zustand	0	1		
00	122	5	0	96.1%
01	12	283	1	95.9%
10	21	143	1	87.2%
11	406	8	0	98.1%

Tabelle 4.2: Beispielhafte Statistikmatrizen eines HPN. Das zur oberen Tabelle gehörende HPN 0 hat nur im Ausgangszustand 01 eine hohe Rückschlußwahrscheinlichkeit von 95.3% für Klasse 1. Im Zustand 11 ist die Rückschlußwahrscheinlichkeit mit 58.9% für Klasse 1 dagegen relativ gering. Sie würde in der Praxis oft nicht ausreichen und das Datum würde zurückgewiesen, denn eine Rückschlußwahrscheinlichkeit von 58.9% ist fast geraten (50%). Möchte man eine Mindestrückschlußwahrscheinlichkeit garantieren, so werden alle Eingangsmuster, die auf einen Ausgangszustand mit zu geringer Rückschlußwahrscheinlichkeit abgebildet werden, zurückgewiesen.
Das HPN 1 ist dagegen wesentlich besser: 83.6% der Trainingsdaten sind in einem Ausgangszustand mit einer Rückschlußwahrscheinlichkeit von 96.1% oder besser. Das Optimum ist also erreicht, wenn in jedem Ausgangszustand nur eine Klasse vertreten ist, da dann alle Rückschlußwahrscheinlichkeiten 100% sind. In der Praxis ist dies normalerweise nicht erreichbar, da die meisten Problemstellungen nicht perfekt lösbar sind.
Die Gütefunktion, die wir später zur Optimierung einführen, wird diese Punkte berücksichtigen und würde demnach HPN 1 besser als HPN 0 bewerten.

4.2 Definition eines Hyperpermutationsnetzwerks

Wir unterteilen die Definition eines HPN in einen statischen Teil, der die Netzwerktopologie betrifft, und einen Teil, der die Dynamik eines HPN definiert.

4.2.1 Statischer Teil

Die von Oberländer (1999) angegebene Definition für ein Hyperpermutationsnetzwerk lautet:

Definition 1. Ein 6-Tupel $\mathrm{H} = (V, E, \mathrm{dim}, \mathrm{in}, \mathrm{out}, \mathrm{hyp})$ heißt *Hyperpermutationsnetzwerk*, wenn es folgende Eigenschaften hat:

1. V ist eine (endliche) Menge von Knoten.
2. E ist eine geordnete (endliche) Menge von Kanten.
3. 'dim' ist eine Abbildung $V \to \mathbb{N} \setminus \{0\}$ (Knotendimensionen).

4. 'in' ist eine injektive Abbildung $\{(v,k) \mid v \in V \wedge k \in \{0,\ldots,\dim(v-1)\} \subset \mathbb{N}\} \to E$ (In die Knoten hereinkommende Kanten).

5. 'out' ist eine injektive Abbildung $\{(v,k) \mid v \in V \wedge k \in \{0,\ldots,\dim(v-1)\} \subset \mathbb{N}\} \to E$ (Aus den Knoten herauslaufende Kanten).

6. a) 'hyp' ist eine Abbildung $V \to \bigcup_{n=1}^{\infty} (\mathbb{B}^n \times \mathbb{B}^n)$ (Knotentabellen).
   b) Für jeden Knoten $v \in V$ liefert 'hyp' eine bijektive Abbildung $\mathrm{h}_v \in (\mathbb{B}^k \times \mathbb{B}^k)$ mit $k = \dim(v)$.

Zur Verdeutlichung gehen wir die Punkte der Definition einzeln durch:

1. V ist die Menge der Knoten des HPN. Die Knoten führen die lokalen Transformationen auf den Daten aus, die sie über die Kanten erhalten.

2. Die Menge E enthält die Kanten bzw. Leitungen des HPN, über die die Daten durch das HPN geschickt werden. Wir werden im folgenden anstelle von Kante synonym auch von *Leitung* sprechen. Da die Menge geordnet ist, können wir später über einen Index auf einzelne Leitungen zugreifen.

3. Die Dimension $d = \dim(v)$ eines Knotens v ist die Zahl seiner Ein- bzw. Ausgänge, an die Leitungen angeschlossen werden können. Jeder Knoten hat genausoviele Eingangs- wie Ausgangsleitungen.

4. Die Abbildung 'in' liefert zu einem Knoten $v \in V$ und einer natürlichen Zahl $k \in \{0,\ldots,\dim(v-1)\}$ eine Leitung $e = \mathrm{in}(v,k) \in E$, in diesem Fall eine *Eingangsleitung* eines Knotens. Die Injektivität von 'in' garantiert,

   a) daß eine Leitung nur Eingangsleitung eines einzigen Knoteneingangs sein kann
   b) und daß jeder „Anschluß" am Eingang eines Knotens mit genau einer Leitung belegt ist.

   Durch den Index k lassen sich die verschiedenen Eingangsleitungen eines Knotens unterscheiden. Wir nennen eine Eingangsleitung $e = \mathrm{in}(v,k)$ auch k-te Eingangsleitung des Knotens v. Da 'in' nur injektiv und nicht bijektiv ist, kann es Leitungen geben, die zu keinem Knoten Eingangsleitung sind. Diese Leitungen führen aus dem Netzwerk heraus. Sie sind die Ausgangsleitungen des HPN, deren Menge wir E_{out} nennen. Eine Leitung, die mit *beiden* Enden mit einem Knoten verbunden ist, nennen wir *interne Leitung*, die Menge der internen Leitungen ist E_{int}.

5. Das Gegenstück zur Abbildung 'in' ist die Abbildung 'out'. Sie liefert zu einem Knoten $v \in V$ und einer natürlichen Zahl $k \in \{0,\ldots,\dim(v-1)\}$ die zugehörige *Ausgangsleitung*. Die Leitungen, die zu keinem Knoten Ausgangsleitung sind, führen in das Netzwerk hinein. Sie sind die Eingangsleitungen des HPN, deren Menge wir E_{in} nennen.

6. Die Abbildung 'hyp' liefert zu jedem Knoten v des Netzwerks die zum Knoten gehörende bijektive Abbildung $\mathrm{h}_v : \mathbb{B}^{\dim(v)} \to \mathbb{B}^{\dim(v)}$. Diese Abbildung führt die lokalen Transformation der Daten durch. Sie wird in den Knoten als Lookup-Tabelle realisiert.

Als Folge der Punkte 4. und 5. ergibt sich

1. daß Leitungen *nicht* an andere Leitungen angeschlossen werden dürfen, also auch *nicht* verzweigen dürfen,

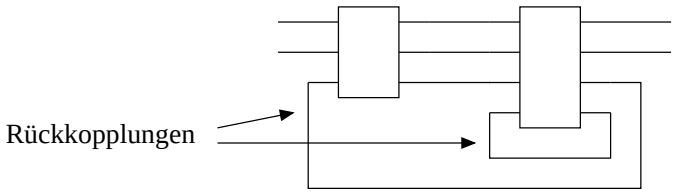


Abbildung 4.3: Beispiel für Rückkopplungen in einem HPN. Rückkopplungen sind über verschiedene Knoten und Spalten des Netzwerks hinweg möglich.

2. daß innerhalb des Netzwerks Zyklen der Leitungen erlaubt sind, wie z. B. Abbildung 4.3 zeigt.

Zur späteren Verwendung definieren wir noch drei globale Größen eines HPN:

Definition 2.

- Die *Dimension* dim(H) eines HPN bezeichnet die Gesamtzahl seiner Eingangs- bzw. Ausgangsleitungen $|E_{\text{in}}|$ bzw. $|E_{\text{out}}|$.[1]
- Die *Ordnung* ord(H) eines HPN bezeichnet die maximal vorkommende Knotendimension $\dim(v)$ im Netzwerk.
- Die *Tiefe* depth(H) eines HPN bezeichnet die maximale auftretende Anzahl von Knoten zwischen einer Eingangsleitung und einer Ausgangsleitung des Netzwerks.

Da die Bijektionen h_v in den Knoten eine zentrale Rolle spielen, geben wir ihnen einen eigenen Namen:

Definition 3. Die zu einem Knoten $v \in V$ der Dimension d gehörende Bijektion $\mathrm{h}_v : \mathbb{B}^{(2^d)} \to \mathbb{B}^{(2^d)}$ nennen wir *Hyperpermutation*.

Ein Knoten führt also auf mehreren Leitungen eine *Hyperpermutation* durch. Nun kann man sich fragen, was das Präfix „Hyper-“ bedeutet. Angenommen, wir sprächen nur davon, daß ein Knoten auf mehreren Leitungen eine *Permutation* durchführt. Dann hätten die meisten von uns die Vorstellung, daß ein Knoten die *Anordnung* der Leitungen permutiert, so wie es Abbildung 4.4 zeigt.

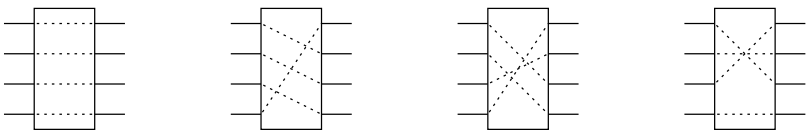


Abbildung 4.4: Permutationen der Leitungen eines HPN. Der linke Knoten reicht die Leitungen unverändert weiter, realisiert also die Identische Abbildung. Die anderen drei Knoten permutieren die Leitungen auf verschiedene Möglichkeiten. Um solche Permutationen zu realisieren, könnten wir den Knoten weglassen und statt dessen die Leitungen auf die richtige Weise „umhängen“.

Ein Netzwerkknoten ändert aber nicht nur die Anordnung der Leitungen, sondern er transformiert die Bitmuster, die diese Leitungen repräsentieren können. Diese Transformationen sind

[1] $|\mathcal{M}|$ bezeichnet die Anzahl von Elementen einer Menge $\mathcal{M}$

deutlich mächtiger als bloße Permutationen der Leitungen: Bei k Leitungen gibt es $k!$ Permutationen der Leitungen, aber es gibt $(2^k)!$ Hyperpermutationen, die die Permutationen der Leitungen einschließen.

4.2.2 Dynamischer Teil

Bis jetzt haben wir nur die „Verdrahtung“ eines HPN definiert, nicht wie es operiert. Was noch fehlt, ist die Definition der Zustandsvariablen und ihrer dynamischen Gleichungen.

4.2.2.1 Zustandsvariablen

Wir müssen festlegen, wie die Leitungen, die bis jetzt nur als „Verbindungselemente“ zwischen den Knoten aufgetreten sind, und die Zustandsvariablen eines HPN zusammenhängen. Wir gehen dazu wie bei Oberländer (1999) vor und definieren eine Funktion, die zu einer Leitung e und einem Zeitpunkt t den Zustand dieser Leitung liefert:

Definition 4. Die Funktion

$$\sigma^{\mathrm{ko}} : \begin{array}{l} E \times \mathbb{Z} \to \mathbb{B} \\ (e,t) \mapsto \sigma^{\mathrm{ko}}(e,t) \end{array} \tag{4.1}$$

ordnet der Leitung e zum Zeitpunkt t den Zustand $\sigma^{\mathrm{ko}}(e,t) \in \mathbb{B}$ zu.
Die zu einer Leitung e bzw. e_i gehörende Zustandsvariable wird durch

$$x_e(t) = \sigma^{\mathrm{ko}}(e,t) \qquad \text{bzw.} \qquad x_i(t) = \sigma^{\mathrm{ko}}(e_i,t) \tag{4.2}$$

definiert. Hier nutzen wir die Möglichkeit, eine Leitung e auch über ihren Index i in einer geordneten Menge von Leitungen zu indizieren. Ist der Zusammenhang unmißverständlich, sprechen wir auch kurz von „der Leitung x_e“ anstelle des „Zustands x_e der Leitung e“.

Damit sind durch σ^{ko} zu jedem Zeitpunkt alle Zustandsvariablen des System und damit der *Gesamtzustand des Systems* festgelegt, die Menge der Zustandsvariablen ist $\{x_e \,|\, e \in E\}$. Das 'ko' bei σ^{ko} steht für *Komponentendarstellung*, dazu mehr in den nächsten Abschnitten.

4.2.2.2 Leitungsbündel

Da wir es oft mit Gruppen mehrerer Leitungen zu tun haben werden, vereinfachen wir die Behandlung solcher Gruppen mit

Definition 5.

1. Ein Tupel von n Leitungen $\boldsymbol{e} = (e_0, \ldots, e_{n-1})$, $e_k \in E$ wird *Leitungsbündel* der Breite n genannt.

2. Ein Tupel von n Zustandsvariablen $\boldsymbol{x} = (x_0, \ldots, x_{n-1})$, $x_k \in \mathbb{B}$ repräsentiert den Zustand des dazugehörenden Leitungsbündels und wird kurz Zustand genannt.

Eine Komponente eines Zustands $\boldsymbol{x}$ bezeichnen wir neben x_i auch mit $[\boldsymbol{x}]_i$.

Wir nennen diese Darstellung des Zustands einer Menge von Leitungen *Komponentendarstellung*, da jede *Komponente* des Zustands $\boldsymbol{x}$ den Zustand einer Leitung repräsentiert. Eine Leitung e kann die beiden Werte 0 oder 1 annehmen, entsprechend kann ein Zustand $\boldsymbol{x}$ aus n verschiedenen Leitungen 2^n verschiedene Werte annehmen. Wir notieren diese Werte nicht

nur als Tupel, sondern auch, indem wir die Zustände der einzelnen Leitungen als Dualzahl schreiben. Der Zustand $(1,0,0,1)$ eines Bündels ergäbe also die Dualzahl 1001.[2]

Wir können nun Leitungsbündel und deren Zustände beschreiben, brauchen aber noch eine Möglichkeit, mehrere Bündel zu einem neuen Bündel zusammenzufassen:

Definition 6. Wir fassen k Leitungsbündel $\boldsymbol{e}_0 = (e_{0,0}, \ldots, e_{0,n_0-1})$, $\boldsymbol{e}_1 = (e_{1,0}, \ldots, e_{1,n_1-1})$ bis $\boldsymbol{e}_{k-1} = (e_{k-1,0}, \ldots, e_{k-1,n_{k-1}-1})$ der Breiten n_0 bis n_{k-1} durch Konkatenierung der jeweiligen Komponenten zu einem gemeinsamen Bündel $\boldsymbol{e}$ zusammen:

$$\boldsymbol{e} = (\boldsymbol{e}_0, \boldsymbol{e}_1, \ldots, \boldsymbol{e}_{k-1}) = (e_{0,0}, \ldots, e_{0,n_0-1}, e_{1,0}, \ldots, e_{1,n_1-1}, \ldots, e_{k-1,0}, \ldots, e_{k-1,n_{k-1}-1})$$

Ebenso werden die dazugehörenden Zustände zusammengefaßt:

$$\boldsymbol{x} = (\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_{k-1}) = (x_{0,0}, \ldots, x_{0,n_0-1}, x_{1,0}, \ldots, x_{1,n_1-1}, \ldots, x_{k-1,0}, \ldots, x_{k-1,n_{k-1}-1})$$

Die Zustände $\boldsymbol{x}_0 = (0,1)$ und $\boldsymbol{x}_1 = (1,0)$ werden also zu einem Bündel $\boldsymbol{x} = (\boldsymbol{x}_0, \boldsymbol{x}_1) = (0,1,1,0)$ zusammengefaßt.

Auch die umgekehrte Operation wird benötigt, die es erlaubt, nur ein Teilbündel eines Leitungsbündels zu betrachten:

Definition 7. Sei $\boldsymbol{x} = (x_0, \ldots, x_{n-1})$ der Zustand eines Leitungsbündels der Breite n. Dann nennen wir die Abbildung

$$\phi^{\mathrm{ko}}{}_{i,j} : \begin{array}{c} \mathbb{B}^n \to \mathbb{B}^m \\ (x_0, \ldots, x_{n-1}) \mapsto (x_i, \ldots, x_j) \end{array}$$
$$\text{mit} \quad m = j - i + 1\,; \quad m \in [1, \ldots, n]\,; \quad i, j \in [0, \ldots, n-1]$$

einen *Fokus* auf $\boldsymbol{x}$ mit der Breite m. Ein Beispiel für einen Fokus ist $\boldsymbol{x}' = \phi^{\mathrm{ko}}{}_{1,3}(0,1,1,0,1,1) = (1,1,0)$, siehe Abbildung 4.5.

Bis jetzt haben wir keine andere Möglichkeit, den Zustand eines Leitungsbündels zu bezeichnen, als den Zustand explizit als Bitstring aufzuschreiben. Um auch über einen Index auf die Zustände zugreifen zu können, ordnen wir jedem Zustand eine natürliche Zahl zu:

---

[2] Hier treffen zwei an sich inkompatible Konventionen aufeinander: In der Mathematik schreibt man die Elemente eines Tupels (meistens) von niedrigen zu hohen Indizes, z. B. $\boldsymbol{x} = (x_0, \ldots, x_{n-1})$. Zahlen dagegen schreibt man genau umgekehrt, von höher- zu niederwertigen Ziffern:

$$\begin{aligned} 4_{\text{Basis } 10} &= 100_{\text{Basis } 2} = 1 \cdot 4 + 0 \cdot 2 + 0 \cdot 1 \\ &= a_2 \cdot 2^2 + a_1 \cdot 2^1 + a_0 \cdot 2^0 \qquad \text{mit} \qquad a_2 = 1,\ a_1 = 0,\ a_0 = 0 \end{aligned}$$

Wir müssen mit beiden Konventionen zurechtkommen und schreiben daher Tupel von niedrigen zu hohen Indizes, Dualzahlen und Bitstrings dagegen von hohen zu niedrigen Indizes, als Beispiel:

$$\boldsymbol{x} = (x_0, x_1, x_2) = (0,0,1) \quad \text{bzw.} \quad \boldsymbol{x} = x_2 x_1 x_0 = 100 \quad \text{mit} \quad x_0 = 0,\ x_1 = 0,\ x_2 = 1$$

Um Tupel und Dualzahlen klar auseinanderhalten zu können, schreiben wir Tupel als geklammerte Liste (x_0, x_1, x_2) und Dualzahlen als Aneinanderreihung von 0 und 1 bzw. x_i ohne trennende Kommas $x_2 x_1 x_0$. Die in der Informatik übliche Bezeichung *Bitstring* für eine Zeichenkette aus 0 und 1 betrachten wir als *Zahlenfolge* und scheiben sie wie ein Tupel als geklammerte Liste $(x_0, x_1, x_2, \ldots)$.

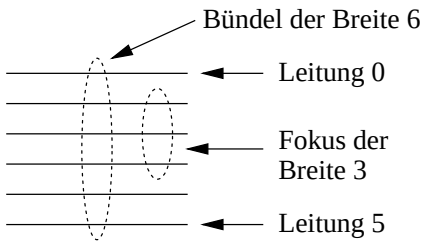


Abbildung 4.5: Fokus $\boldsymbol{x}'$ vom Bit 1 bis zum Bit 3 eines 6 Bit breiten Leitungsbündels $\boldsymbol{x}$, wobei die „oberste“ Leitung die Leitung 0 ist. Der Fokus wird durch $\boldsymbol{x}' = {\phi^{\mathrm{ko}}}_{1,3}(\boldsymbol{x}) = (x_1, x_2, x_3)$ definiert. Die anderen Leitungen werden nicht weiter beachtet.

Definition 8. Die Funktion ‘ind’ ordnet den verschiedenen Zuständen $\boldsymbol{x}$ eines Leitungsbündels der Breite n auf folgende Weise natürliche Zahlen zu:

$$\begin{aligned} & & \mathbb{B}^n &\to \mathbb{N} \\ \mathrm{ind}: & & \boldsymbol{x} &\mapsto \sum_{i=0}^{n-1} x_i\, 2^i \end{aligned}$$

Die Umkehrfunktion $\mathrm{ind}_d^{-1} : \mathbb{N} \to \mathbb{B}^d$ ordnet einer natürlichen Zahl den dazugehörigen Zustand der Dimension d zu.

Auf diese Weise ist $\boldsymbol{x}$ als Dualzahl geschrieben gerade die Dualzahldarstellung von $\mathrm{ind}(\boldsymbol{x})$, z. B. $\boldsymbol{x} = 100$ und $\mathrm{ind}(\boldsymbol{x}) = 4$.

4.2.2.3 Die Zeitentwicklung

Die Beziehung zwischen Leitungen und Zustandsvariablen haben wir bereits definiert, nun legen wir fest, wie sich der Systemzustand zum Zeitpunkt $t+1$ aus dem Systemzustand zum Zeitpunkt t ergibt:

Definition 9. Die zeitliche Entwicklung eines HPN ist durch die Gleichung

$$x_e(t+1) = [\mathrm{h}_v(\boldsymbol{z}(t))]_k \qquad \text{mit} \qquad \begin{aligned} & t \geq 0 \\ & e = \mathrm{out}(v,k) \\ & z_i(t) = x_{\mathrm{in}(v,i)}(t) = \sigma^{\mathrm{ko}}(\mathrm{in}(v,i), t) \\ & v \in V;\ k, i = 0, \ldots, \dim(v) - 1 \end{aligned} \tag{4.3}$$

festgelegt.

Jeder Knoten v des HPN führt also von einem Zeitschritt zum nächsten folgendes durch:

1. Der Knoten liest seine $\dim(v)$ Eingangsleitungen $\mathrm{in}(v,i)$ zum Zeitpunkt t und faßt ihre Zustände in einem Tupel $\boldsymbol{z}$ mit $z_i(t) = \sigma^{\mathrm{ko}}(\mathrm{in}(v,i), t)$ zusammen.

2. Der Knoten transformiert mit Hilfe seiner Hyperpermutation h_v sein Eingangsbündel: $\boldsymbol{z}(t) \mapsto \mathrm{h}_v(\boldsymbol{z}(t))$. Praktisch gesprochen schlägt der Knoten den Eintrag für den Bitstring $\boldsymbol{z}$ in seiner Knotentabelle nach.

3. Der Knoten schreibt das k-te Bit des transformierten Bitstrings $\mathrm{h}_v(\boldsymbol{z}(t))$ auf seine k-te Ausgangsleitung $x_{\mathrm{out}(v,k)}(t+1)$.

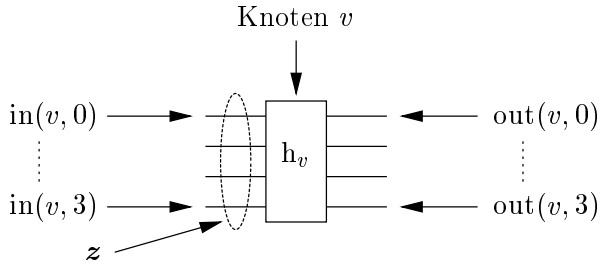


Abbildung 4.6: Veranschaulichung der Zeitentwicklungsgleichung an einem Knoten v mit $d = \dim(v) = 4$. Bei jedem Zeitschritt werden erst die Zustände der Eingangsleitungen $\mathrm{in}(v,0),\ldots,\mathrm{in}(v,3)$ des Knotens v zu einem Zustand $\boldsymbol{z}$ zusammengefaßt. Dann wird der Zustand $\boldsymbol{z}$ mit h_v transformiert und das Ergebnis der Transformation auf die Ausgangsleitungen $x_{\mathrm{out}(v,0)}(t+1),\ldots,x_{\mathrm{out}(v,3)}(t+1)$ geschrieben.

Abbildung 4.6 veranschaulicht diesen Prozeß.

Nun legen wir noch die Anfangsbedingungen der internen Leitungen für $t = 0$ durch die Funktion $\mathrm{Init}_e : E_{\mathrm{int}} \to \mathbb{B}$ und die Werte der Eingangsleitungen für $t \geq 0$ durch die Funktion $\mathrm{Input}_e(t) : E_{\mathrm{in}} \to \mathbb{B}$ fest, da diese durch Gleichung 4.3 noch nicht bestimmt sind. Damit ist der Gesamtzustand des HPN, also die Menge $\{x_e \,|\, e \in E\}$, zu jedem Zeitpunkt $t \geq 0$ festgelegt.

Was ist aber mit den Zeitpunkten $t \leq 0$? Da jeder Knoten eine umkehrbare Abbildung durchführt, ist auch die Transformation des gesamten Netzwerks umkehrbar. Wir können das Netzwerk also in der Zeit rückwärts laufen lassen, wenn wir die Rollen von Netzwerkeingang und -ausgang vertauschen. Wollen wir zu Zeiten $t \leq 0$ vorstoßen, müssen wir den Wert der Ausgangsleitungen des Netzes mit Hilfe von $\mathrm{Output}_e(t) : E_{\mathrm{out}} \to \mathbb{B}$ festlegen, die Werte der Eingangsleitungen werden dann durch die Netzwerktransformation festgelegt. Zusammengefaßt ergibt sich damit die für alle Zeiten t gültige

Definition 10.

$$x_e(t) = \begin{cases} \mathrm{Input}_e(t) & t \geq 0\,; \quad e \in E_{\mathrm{in}} \\ \mathrm{Output}_e(t) & t \leq 0\,; \quad e \in E_{\mathrm{out}} \\ \mathrm{Init}_e & t = 0\,; \quad e \in E_{\mathrm{int}} \\ [\mathrm{h}_v(\boldsymbol{z}(t-1))]_k & \text{mit} \quad \begin{array}{l} t > 0 \\ e = \mathrm{out}(v,k) \\ z_i(t) = \sigma^{\mathrm{ko}}(\mathrm{in}(v,i),t) \\ v \in V;\ k,i = 0,\ldots,\dim(v)-1 \end{array} \\ [\mathrm{h}_v^{-1}(\boldsymbol{z}(t+1))]_k & \text{mit} \quad \begin{array}{l} t < 0 \\ e = \mathrm{in}(v,k) \\ z_i(t) = \sigma^{\mathrm{ko}}(\mathrm{in}(v,i),t) \\ v \in V;\ k,i = 0,\ldots,\dim(v)-1 \end{array} \end{cases} \tag{4.4}$$

Trotz dieser vielleicht etwas kompliziert anmutenden formalen Definition ist die Implementierung eines Hyperpermutationsnetzwerkes in Hardware einfach: Die Netzwerkknoten lassen sich durch RAM-Bausteine realisieren und die Netzwerkleitungen durch die Verdrahtung auf einem Chip.

4.2.2.4 Die Netzwerkfunktion eines vorwärtsgerichteten Netzwerks

Wenn wir ein rückkopplungsfreies Netzwerk vor uns haben, können wir ihm eine *zeitunabhängige* Netzwerkfunktion zuordnen: Wir legen einen Bitstring $\mathbf{a}$ am Eingangsbündel $\boldsymbol{x}$ des HPN für $t \geq 0$ an und lassen das Netz solange laufen, bis sich der Bitstring $\mathbf{b}$ am Ausgangsbündel $\boldsymbol{y}$ des HPN nicht mehr ändert. Dies führt zur

Definition 11. Sei H ein rückkopplungsfreies HPN der Dimension $n = \dim(\mathrm{H})$ und der Tiefe $d = \mathrm{depth}(\mathrm{H})$. Sei $\boldsymbol{x}(t)$ der Zustand der Eingangsleitungen des HPN und $\boldsymbol{y}(t)$ der Zustand der Ausgangsleitungen. Dann wird die *Netzwerkfunktion* $\mathrm{h_H}$, die einen Bitstring $\mathbf{a}$ in einen Bitstring $\mathbf{b}$ transformiert, durch

$$\mathrm{h_H}: \begin{array}{l} \mathbb{B}^n \to \mathbb{B}^n \\ \mathbf{a} \mapsto \mathbf{b} = \boldsymbol{y}(d) \end{array} \qquad \text{mit } \boldsymbol{x}(t) = \mathbf{a} \text{ für } t \geq 0 \tag{4.5}$$

festgelegt.

Bei rückkopplungsfreien Netzen können wir jetzt auch davon sprechen, daß das Netz eine Abbildung $\mathbf{b} = \mathrm{h_H}(\mathbf{a})$ auf Bitstrings realisiert, ohne daß wir die Zeit explizit in Betracht ziehen müssen.

Hat ein HPN Rückkopplungen, läßt es sich auch als endlicher Automat interpretieren, was in Abschnitt 8.1.2 näher beleuchtet wird.

4.3 Eigenschaften eines Hyperpermutationsnetzwerks

Nach der formalen Definition eines Hyperpermutationsnetzwerks interessiert uns natürlich, was dieses Berechnungsmodell leistet bzw. nicht leistet. Dazu sehen wir uns zunächst einen und dann mehrere Knoten an.

In manchen Punkten ist ein HPN sehr ähnlich zu dem Modell, das Toffoli (1980a) beschreibt. In Abschnitt 4.5.4 wird genauer auf die Gemeinsamkeiten bzw. Unterschiede zwischen HPN und dem „Reversible Computing“ von Toffoli eingegangen.

4.3.1 Funktionale Kapazität eines Knotens

Ein einzelner Knoten ist formal ein einknotiges Netzwerk. Wir fassen die d Eingangs- bzw. Ausgangsleitungen eines Knotens der Dimension d jeweils in den Bündeln $\boldsymbol{x}$ und $\boldsymbol{y}$ zusammen (siehe Abbildung 4.7). Welche Funktionen können nun von einem Knoten realisiert werden?

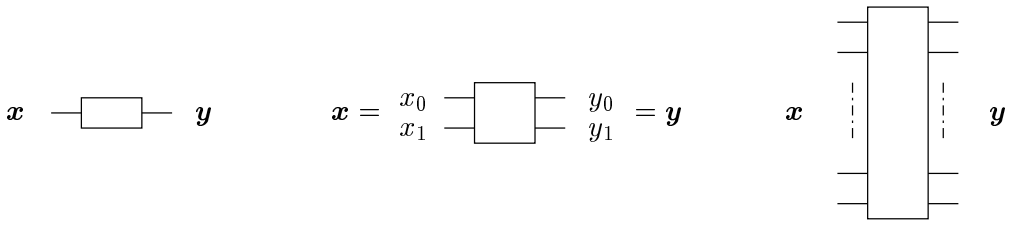


Abbildung 4.7: HPN-Knoten mit der Dimension eins, zwei und d. Die oberste Leitung eines Knotens bzw. Bündels hat den Index 0, die unterste Leitung den Index (Knotendimension − 1).

4.3.1.1 Umkehrbare Funktionen

Wir beginnen mit

Definition 12. Eine Abbildung $f : X \to Y$ heißt *endlich*, wenn X und Y endliche Mengen sind.

Eine Hyperpermutation $h : \mathbb{B}^d \to \mathbb{B}^d$ im Innern eines Knotens ist also eine endliche bijektive Funktion. Eine Hyperpermutation h läßt sich einfach dadurch realisieren, daß wir für jedes Urbild $\boldsymbol{x}$ das Bild $\boldsymbol{y} = h(\boldsymbol{x})$ tabellieren.

Der einfachste Knoten ist der 1-Bit-Knoten. Es gibt nur eine Eingangs- und Ausgangsleitung, $\boldsymbol{x}$ und $\boldsymbol{y}$ kommen aus der Menge $\mathbb{B} = \{0, 1\}$. Es gibt nur zwei bijektive Funktionen $f^{\text{bij}} : \mathbb{B} \to \mathbb{B}$, nämlich die identische Funktion und die Negation:

Identische Funktion :

$\boldsymbol{x}$	$\boldsymbol{y} = h(\boldsymbol{x})$
0	0
1	1

Negation :

$\boldsymbol{x}$	$\boldsymbol{y} = h(\boldsymbol{x})$
0	1
1	0

Der 2-Bit-Knoten ist immer noch recht einfach. Er hat zwei Eingangs- und Ausgangsleitungen, und $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{B}^2 = \{00, 01, 10, 11\}$. Da die Knotentabelle $2^2 = 4$ Zeilen hat, gibt es $4!$ Möglichkeiten, die vier Ausgangssymbole $\boldsymbol{y}$ auf der rechten Tabellenseite anzuordnen und damit $4! = 24$ mögliche Hyperpermutationen von $\mathbb{B}^2 \to \mathbb{B}^2$.

Am Eingang eines n-Bit Knotens können 2^d verschiedene Bitstrings vorkommen, daher hat die Tabelle im Knoten die Länge 2^d und die Zahl der möglichen Hyperpermutationen ist $(2^d)!$.

Da wir nur bijektive Funktionen betrachten, steht uns nur ein Teil der prinzipiell möglichen Funktionen $f : \mathbb{B}^d \to \mathbb{B}^d$ zur Verfügung. Wie groß ist dieser Teil? Da wir alle Funktionen f durch jeweils eine Tabelle realisieren können, läßt sich die Zahl der möglichen Funktionen leicht abzählen: Es gibt 2^d Tabelleneinträge, und in jedem Eintrag haben wir 2^d verschiedene Möglichkeiten, das sind ingesamt $(2^d)^{(2^d)}$ Funktionen. Daran haben die umkehrbaren Funktionen „nur“ einen Anteil von $(2^d)!/[(2^d)^{(2^d)}]$. Damit wir eine Vorstellung von der Größenordnung dieser Zahlen bekommen, sind sie bis zur Dimension $d = 8$ in Tabelle 4.3 aufgeführt.

Bits	Tabellengröße	Hyperpermutationen	alle Funktionen	Quotient
1	2	2	4	0.5
2	4	24	256	0.094
3	8	$4.0 \cdot 10^{4}$	$1.7 \cdot 10^{7}$	$2.4 \cdot 10^{-3}$
4	16	$2.1 \cdot 10^{13}$	$1.8 \cdot 10^{19}$	$1.1 \cdot 10^{-6}$
5	32	$2.6 \cdot 10^{35}$	$1.5 \cdot 10^{48}$	$1.8 \cdot 10^{-13}$
6	64	$1.3 \cdot 10^{89}$	$4.0 \cdot 10^{115}$	$3.2 \cdot 10^{-27}$
7	128	$3.9 \cdot 10^{215}$	$5.3 \cdot 10^{269}$	$7.3 \cdot 10^{-55}$
8	256	$8.6 \cdot 10^{506}$	$3.2 \cdot 10^{616}$	$2.7 \cdot 10^{-110}$

Tabelle 4.3: Tabellengrößen und Zahl der Hyperpermutationen eines Knotens. Die Hyperpermutationen entsprechen den umkehrbaren Funktionen, in der Spalte „Quotient“ steht die Zahl der Hyperpermutationen dividiert durch die Gesamtzahl der möglichen Funktionen.

Wir stellen fest, daß die Zahl der Hyperpermutationen schon für moderate Knotengrößen explosionsartig anwächst. Trotzdem machen die umkehrbaren Funktionen nur einen verschwindend geringen Teil der überhaupt möglichen Funktionen aus. Wir müssen uns fragen, ob wir uns dadurch nicht zu sehr einschränken. Die Antwort ist „nein“, wie wir im folgenden zeigen.

4.3.1.2 Unumkehrbare Funktionen

Wie können wir trotz der Beschränkung auf umkehrbare Funktionen unumkehrbare Funktionen realisieren? Indem wir eine unumkehrbare Funktion in einen Raum höherer Dimension einbetten. In dem Raum höherer Dimension wird die unumkehrbare Funktion umkehrbar. Dies präzisiert der folgende Satz aus Toffoli (1980a):

Satz 1. *Für jede endliche Funktion $f : \mathbb{B}^m \to \mathbb{B}^n$ existert eine umkehrbare endliche Funktion $f^{bij} : \mathbb{B}^r \times \mathbb{B}^m \to \mathbb{B}^n \times \mathbb{B}^{r+m-n}$, mit $r \leq n$, so daß*

$$f_i^{bij}(\underbrace{0,\ldots,0}_{r}, x_0, \ldots, x_{m-1}) = f_i(x_0, \ldots, x_{m-1}), \qquad (i = 0, \ldots, n-1)$$

Beweis. Für den Beweis siehe Toffoli (1980a). □

Folgerung 1. Wir können mit einem Knoten unumkehrbare Funktionen realisieren, indem wir seine ursprüngliche Dimension d auf $d' > d$ vergrößern und die $\Delta d = d' - d$ hinzugekommenen Eingangsleitungen auf einen konstanten Wert festsetzen. Die Zahl der effektiv zu speichernden Funktionswerte für f^{bij} mit der Dimension d' ist wegen der konstanten Δd Eingangsleitungen genauso groß wie für f mit der Dimension d.

Wir kommen damit zur

Definition 13. Eine Eingangsleitung eines Knotens, die für alle Zeiten auf den Wert 0 festgesetzt wird, heißt *Blindbit*.

Wie dies in einem konkreten Fall aussieht, zeigt

Beispiel 1. Wir möchten die Funktion $y = f(x)$ mit der Zuordnungsvorschrift

f :

x	y
0	0
1	0

mit einem HPN-Knoten realisieren. Mit einem 1-Bit-Knoten ist dies direkt nicht möglich, da f nicht umkehrbar ist. Wir erweitern die Knotendimension auf $d = 2$ und legen das neu entstandene Eingangsbit im Sinne von Satz 1 auf den Wert 0 fest, siehe Abb. 4.8. Das erweiterte Eingangsbündel $\boldsymbol{x}$ kann die beiden Werte 0 0 und 0 1 annehmen. Die ursprüngliche Leitung x_0 könnten wir aus $\boldsymbol{x}$ mit dem Fokus $x_0 = {\phi^{\mathrm{ko}}}_{0,0}(\boldsymbol{x})$ wieder erhalten.

Von den beiden Ausgangsleitungen interessiert uns nur eine Leitung, und zwar $y_0 = {\phi^{\mathrm{ko}}}_{0,0}(\boldsymbol{y})$, da f nur eindimensional ist. Wir fordern, daß die Leitung $y_0 = f(x_0) = y$ ist. Nun können wir die Funktion f in die Funktion f^{bij} einbetten, die wir z. B. so wählen können:

f^{bij} :

$x_1\, x_0$	$y_1\, y_0$
0 0	0 0
0 1	1 0
1 0	0 1
1 1	1 1

Die Funktion f^{bij} ist umkehrbar und es gilt die geforderte Beziehung $y_0 = f(x_0)$ für $x_1 = 0$. Es gibt noch drei weitere Funktionen f^{bij}, die die geforderte Beziehung erfüllen: Wir können sowohl die beiden Einträge 0 0 und 1 0 oben rechts in der Tabellen miteinander vertauschen, als auch die beiden Einträge 0 1 und 1 1 unten rechts miteinander vertauschen. Zusammen sind das $2 \cdot 2 = 4$ mögliche Funktionen f^{bij}, die $y_0 = f(x_0)$ erfüllen. Da das Blindbit x_1 immer den Wert 0 hat, kann die untere Tabellenhälfte nie adressiert werden. In einer Implementierung wird daher nur die obere Hälfte der Tabelle gespeichert.

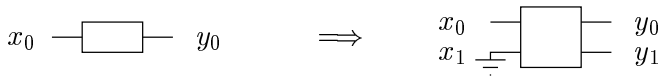


Abbildung 4.8: Ein HPN-Knoten der Dimension 1 wird auf die Dimension 2 erweitert. Die neu entstandene Eingangsleitung benutzen wir als Blindbit. Es ist durch das Erdungssymbol gekennzeichnet und hat immer den Wert 0.

4.3.1.3 Boole'sche Funktionen

Nach dem Beispiel ist es nun klar, wie wir mit einem Hyperpermutationsknoten die Boole'schen Funktionen

not :

x	y
0	1
1	0

xor :

x	y
00	0
01	1
10	1
11	0

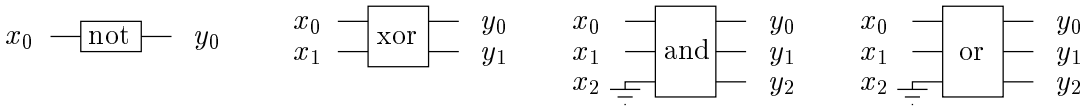
and :

x	y
00	0
01	0
10	0
11	1

or :

x	y
00	0
01	1
10	1
11	1

realisieren. Das 'not' läßt sich direkt mit einem 1-Bit Knoten realisieren, wie wir oben schon gesehen haben. Für das 'xor' brauchen wir einen 2-Bit Knoten, dessen zweites Ausgangsbit y_1 wir ignorieren. Für das 'and' und 'or' müssen wir die Knotendimension mit einem Blindbit um eins erhöhen. Eine Möglichkeit der Realisierung der Boole'schen Funktionen mit Hyperpermutationsknoten zeigt Abbildung 4.9.



x_0	y_0
0	1
1	0

x_1x_0	y_1y_0
00	00
01	01
10	11
11	10

$x_2x_1x_0$	$y_2y_1y_0$
000	000
001	010
010	100
011	111
100	001
101	011
110	101
111	110

$x_2x_1x_0$	$y_2y_1y_0$
000	000
001	011
010	101
011	111
100	001
101	010
110	100
111	110

Abbildung 4.9: Realisierung der Boole'schen Funktionen mit Hyperpermutationsknoten. Das Ausgangsbit y_0 liefert den Wert der jeweiligen Boole'schen Funktion. In der Wahl der restlichen Bits sind wir frei, solange wir die Bijektivität nicht verletzen. Wir wählen sie so, daß die nicht benötigten Ausgangsbits jeweils den Wert eines ('xor') oder der beiden ('and', 'or') Eingangsbits haben.

Im allgemeinen muß man an einen Hyperpermutationsknoten der Dimension n nur *ein* Blindbit hinzufügen, um mit dem Knoten alle Boole'schen Funktionen der n nichtkonstanten Eingangsleitungen realisieren zu können. Mit Hyperpermutationsknoten der Dimension $d \leq 3$ lassen sich die Boole'schen Grundfunktionen realisieren. Wie ein Knoten Boole'sche Funktionen

auch lernen kann, wird in Abschnitt 10 auf Seite 131 gezeigt. Da man aus den Boole'schen Grundfunktionen jede beliebige Boole'sche Funktion zusammensetzen kann, können wir auch mit einem HPN jede beliebige Boole'sche Funktion realisieren, auch wenn wir nur Knoten der Dimension $d \leq 3$ verwenden.

4.3.2 Vollständigkeit eines Hyperpermutationsnetzwerks

Beliebige Boole'sche Funktionen $\mathbb{B}^n \to \mathbb{B}$ können wir mit einem HPN also realisieren. Können wir auch beliebige Hyperpermutationen $\mathbb{B}^n \to \mathbb{B}^n$ realisieren, wenn wir nur Knoten einer Dimension $d < n$ verwenden? Wäre dies nicht der Fall, so gäbe es Hyperpermutationen, die sich nicht in ein Netzwerk aus Knoten mit geringerer Dimension als der Netzwerkeingang selbst zerlegen ließen, was eine Einschränkung wäre.

Diese Frage beantwortet ein Satz von Oberländer (1995). Er besagt, daß man jede invertierbare Funktion $\mathbb{B}^n \to \mathbb{B}^n$ mit einem Hyperpermutationsnetzwerk realisieren kann, das nur aus Knoten der Dimension $d \leq 3$ besteht, wenn man zusätzlich zu den n Datenbits am Eingang des Netzwerks noch maximal $n-2$ Blindbits zur Verfügung stellt. Diese Blindbits werden im Netzwerk als temporärer Speicher verwendet und haben am Netzwerkausgang wieder denselben Wert wie am Netzwerkeingang. Die Beweisidee ist, die Permutation im $\mathbb{B}^n$ in eine Folge von Transpositionen zu zerlegen, die jeweils genau zwei Bitmuster miteinander vertauschen. Von diesen Transpositionen wird dann gezeigt, daß sie mit Knoten einer Dimension $d \leq 3$ realisiert werden können.

Interessanterweise hat Toffoli (1980a) in einem Artikel zum *Reversible Computing* (mehr dazu in Abschnitt 4.5.4) praktisch die gleiche Aussage für eine andere Art von Netzwerkknoten bewiesen. Toffoli arbeitet bei seinen Netzwerken mit Knoten, die wie HPN-Knoten die gleiche Zahl von Ein- und Ausgangsbits haben und invertierbare Funktionen realisieren. Die invertierbare Funktion in einem Toffoli-Knoten ist aber nicht wie beim HPN-Knoten frei wählbar, sondern sie ist die sog. generalisiere AND/NAND-Funktion der Dimension n:

$$\begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_{n-2} \\ x_{n-1} \end{pmatrix} \mapsto \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_{n-2} \\ x_{n-1} \oplus (x_0 \cdot x_1 \cdot \dots \cdot x_{n-2}) \end{pmatrix} \tag{4.6}$$

Hierbei bezeichnet '$\oplus$' das XOR und '$\cdot$' das AND. Toffoli beweist nun einen Satz der besagt, daß man jede invertierbare Funktion $\mathbb{B}^n \to \mathbb{B}^n$ mit einem Netzwerk aus generalisieren AND/NAND-Funktionen der Dimension 3 realisieren kann, wenn man $3n-3$ „Zwischenspeicher-Leitungen" zur Verfügung stellt. Diese Zwischenspeicher-Leitungen haben genau wie die Blindbits am Netzwerkausgang den gleichen Wert wie am Netzwerkeingang. Nun läßt sich die Abbildung aus Gleichung (4.6) aber direkt mit einem HPN-Knoten realisieren, weshalb der Satz von Toffoli auch für HPNs anwendbar ist. Es fällt auf, daß Toffoli mit $3n-3$ Zwischenspeicher-Leitungen deutlich mehr Zwischenspeicher benötigt als die $n-2$ Blindbits aus dem Satz von Oberländer (1995). Dies ist aber plausibel, denn Toffoli läßt im Gegensatz zu Oberländer nur eine einzige der $(2^3)! = 40320$ möglichen invertierbaren Funktionen auf drei Bit als Grundbaustein des Netzwerks zu.

4.4 Ein Hyperpermutationsnetzwerk als lineare Funktion

Ein Hyperpermutationsnetzwerk läßt sich im sogenannten *Produktraum*, der jetzt eingeführt wird, als lineare Funktion darstellen. Diese Darstellung wird benötigt, da später die Optimie-

rung auch im Produktraum durchgeführt wird.

4.4.1 Der Produktraum

Bisher haben wir den Zustand einer Leitung e durch die Zustandsvariable $x_e \in \mathbb{B}$ beschrieben. Das heißt, der Zustand einer Leitung wurde einfach durch $x_e = 0$ oder $x_e = 1$ festgelegt. Diese Darstellung des Zustands einer Menge von Leitungen haben wir *Komponentendarstellung* genannt. Nun gehen wir zu einer anderen Darstellung über, die wir Darstellung im *Produktraum* oder auch *Produktdarstellung* nennen. Eine ähnliche Darstellung hat Davio (1981) für Vertauschungsnetzwerke verwendet. In der Produktdarstellung beschreiben wir den Zustand einer Leitung durch einen Einheitsvektor:

Definition 14. Die Funktion

$$\sigma^{\mathrm{pr}}: \quad \begin{array}{l} E \times \mathbb{Z} \to \mathbb{B}^2 \\ (e,t) \mapsto \sigma^{\mathrm{pr}}(e,t) = \boldsymbol{\epsilon}_i \qquad \text{mit} \qquad i = x_e(t) \end{array} \tag{4.7}$$

ordnet der Leitung e zum Zeitpunkt t den (kanonischen) Einheitsvektor $\boldsymbol{\epsilon}_{x_e(t)} \in \mathbb{B}^2$ zu. Dabei ist $\boldsymbol{\epsilon}_i$ durch

$$\epsilon_{i,j} = \delta_{i,j}$$

definiert, wobei $\epsilon_{i,j}$ die j-te Komponente des Vektors $\boldsymbol{\epsilon}_i$ bezeichnet und δ das Kronecker-δ

$$\delta_{i,j} = \begin{cases} 0: & \text{falls } i \neq j \\ 1: & \text{falls } i = j \end{cases}$$

ist.
Der zu einer Leitung e bzw. e_i gehörende Zustandsvektor wird mit

$$\boldsymbol{\chi}_e(t) = \sigma^{\mathrm{pr}}(e,t) \qquad \text{bzw.} \qquad \boldsymbol{\chi}_i(t) = \sigma^{\mathrm{pr}}(e_i,t) \tag{4.8}$$

bezeichnet.

Wir haben also die äquivalenten Zustandsdarstellungen

$$x_e = 0 \quad \leftrightarrow \quad \boldsymbol{\chi}_e = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \qquad \text{und} \qquad x_e = 1 \quad \leftrightarrow \quad \boldsymbol{\chi}_e = \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

Sie ähneln der Spindarstellung in der Physik, nur daß dort normalerweise die symmetrischen Zustandsvektoren $\binom{-1}{1}$ und $\binom{1}{-1}$ bevorzugt werden.

4.4.1.1 Kombination von Leitungsbündeln

Im Komponentenraum war klar, wie wir zwei Leitungen x_0 und x_1 zusammenfassen: Wir bilden ein Leitungsbündel $\boldsymbol{x} = (x_0, x_1)$, das die vier möglichen Zustände $(0,0)$, $(1,0)$, $(0,1)$ und $(1,1)$ annehmen kann. Das Analogon im Produktraum ist, daß wir die vier möglichen Zustände zweier Leitungen durch die vier Einheitsvektoren $\boldsymbol{\epsilon}_i \in \mathbb{B}^4$ darstellen:

$$\boldsymbol{x} = (0,0) \quad \leftrightarrow \quad \boldsymbol{\chi} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}; \qquad \boldsymbol{x} = (1,0) \quad \leftrightarrow \quad \boldsymbol{\chi} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}$$

$$\boldsymbol{x} = (0,1) \quad \leftrightarrow \quad \boldsymbol{\chi} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}; \qquad \boldsymbol{x} = (1,1) \quad \leftrightarrow \quad \boldsymbol{\chi} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

Um die Kombination von Zustandsvektoren mathematisch durchzuführen, benötigen wir das *Kronecker-Produkt*:

Definition 15. Das Kronecker-Produkt $\boldsymbol{A}\otimes\boldsymbol{B}$ einer $(r_\mathrm{a}, c_\mathrm{a})$-Matrix $\boldsymbol{A}$ mit einer $(r_\mathrm{b}, c_\mathrm{b})$-Matrix $\boldsymbol{B}$ ist eine $(r_\mathrm{a}\, r_\mathrm{b}, c_\mathrm{a}\, c_\mathrm{b})$-Matrix $\boldsymbol{M}$, die durch

$$m_{i,j} = a_{i_\mathrm{a},j_\mathrm{a}}\, b_{i_\mathrm{b},j_\mathrm{b}} \qquad \text{mit} \qquad i = i_\mathrm{a}\, r_\mathrm{b} + i_\mathrm{b}\,;\; j = j_\mathrm{a}\, c_\mathrm{b} + j_\mathrm{b}$$

definiert ist. In Matrixschreibweise:

$$\boldsymbol{M} = \boldsymbol{A}\otimes\boldsymbol{B} = \begin{pmatrix} a_{0,0}\,\boldsymbol{B} & \cdots & a_{0,c_\mathrm{a}-1}\,\boldsymbol{B} \\ \vdots & \ddots & \vdots \\ a_{r_\mathrm{a}-1,0}\,\boldsymbol{B} & \cdots & a_{r_\mathrm{a}-1,c_\mathrm{a}-1}\,\boldsymbol{B} \end{pmatrix}$$

Beispiel 2. Das Kronecker-Produkt der beiden Matrizen

$$\boldsymbol{A} = \begin{pmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{pmatrix} \qquad \text{und} \qquad \boldsymbol{B} = \begin{pmatrix} b_{00} & b_{01} \\ b_{10} & b_{11} \end{pmatrix}$$

ist gegeben durch

$$\boldsymbol{C} = \boldsymbol{A}\otimes\boldsymbol{B} = \begin{pmatrix} a_{00}\,\boldsymbol{B} & a_{01}\,\boldsymbol{B} \\ a_{10}\,\boldsymbol{B} & a_{11}\,\boldsymbol{B} \end{pmatrix} = \begin{pmatrix} a_{00}\,b_{00} & a_{00}\,b_{01} & a_{01}\,b_{00} & a_{01}\,b_{01} \\ a_{00}\,b_{10} & a_{00}\,b_{11} & a_{01}\,b_{10} & a_{01}\,b_{11} \\ a_{10}\,b_{00} & a_{01}\,b_{01} & a_{11}\,b_{00} & a_{11}\,b_{01} \\ a_{10}\,b_{10} & a_{10}\,b_{11} & a_{11}\,b_{10} & a_{11}\,b_{11} \end{pmatrix}.$$

Wie wir nun mehrere Leitungen in der Produktdarstellung zusammenfassen, zeigt

Definition 16. Der Zustandsvektor $\boldsymbol{\chi}$ eines Leitungsbündels $\boldsymbol{e} = (e_0, \ldots, e_{n-1})$ ist durch

$$\boldsymbol{\chi} = \boldsymbol{\chi}_{n-1}\otimes\boldsymbol{\chi}_{n-2}\otimes\cdots\otimes\boldsymbol{\chi}_0 = \bigotimes_{k=n-1}^{0}\boldsymbol{\chi}_k \qquad \text{mit} \qquad \boldsymbol{\chi}_k(t) = \sigma^{\mathrm{pr}}(e_k, t)$$

gegeben.

Die $\boldsymbol{\chi}_k$ sind zweidimensionale Einheitsvektoren, daher sind beliebige Kronecker-Produkte der $\boldsymbol{\chi}_k$ Einheitsvektoren in höheren Dimensionen. Jeder der 2^n möglichen Zustände eines n Bit breiten Leitungsbündels wird also durch einen Einheitsvektor repräsentiert.

Beispiel 3. Wir wollen den Zustandsvektor $\boldsymbol{\chi}$ eines Leitungsbündels (e_0, e_1) aus den beiden Zustandsvektoren $\boldsymbol{\chi}_0$ und $\boldsymbol{\chi}_1$ der einzelnen Leitungen e_0 und e_1 bestimmen. Nach Definition 16 berechnen wir:

$$\boldsymbol{\chi} = \boldsymbol{\chi}_1\otimes\boldsymbol{\chi}_0 = \begin{pmatrix} \chi_{1,0} \\ \chi_{1,1} \end{pmatrix}\otimes\begin{pmatrix} \chi_{0,0} \\ \chi_{0,1} \end{pmatrix} = \begin{pmatrix} \chi_{1,0}\,\chi_{0,0} \\ \chi_{1,0}\,\chi_{0,1} \\ \chi_{1,1}\,\chi_{0,0} \\ \chi_{1,1}\,\chi_{0,1} \end{pmatrix}$$

Die vier möglichen Zustände des Leitungsbündels (e_0, e_1) sind also in der Komponenten- bzw. Produktdarstellung:

$$\boldsymbol{x} = (0,0) \quad \leftrightarrow \quad \boldsymbol{\chi} = \begin{pmatrix}1\\0\end{pmatrix} \otimes \begin{pmatrix}1\\0\end{pmatrix} = \begin{pmatrix}1\\0\\0\\0\end{pmatrix} \quad ; \quad \boldsymbol{x} = (1,0) \quad \leftrightarrow \quad \boldsymbol{\chi} = \begin{pmatrix}1\\0\end{pmatrix} \otimes \begin{pmatrix}0\\1\end{pmatrix} = \begin{pmatrix}0\\1\\0\\0\end{pmatrix}$$

$$\boldsymbol{x} = (0,1) \quad \leftrightarrow \quad \boldsymbol{\chi} = \begin{pmatrix}0\\1\end{pmatrix} \otimes \begin{pmatrix}1\\0\end{pmatrix} = \begin{pmatrix}0\\0\\1\\0\end{pmatrix} \quad ; \quad \boldsymbol{x} = (1,1) \quad \leftrightarrow \quad \boldsymbol{\chi} = \begin{pmatrix}0\\1\end{pmatrix} \otimes \begin{pmatrix}0\\1\end{pmatrix} = \begin{pmatrix}0\\0\\0\\1\end{pmatrix}$$

Wir erhalten also genau das gewünschte Verhalten: Jeder Zustand des Leitungsbündels wird als einer von vier Zwei-Tupeln bzw. als einer von vier Einheitsvektoren repräsentiert. Wir geben nun eine Abbildung an, die direkt zwischen der Komponentendarstellung und der Produktdarstellung vermittelt:

Satz 2. *Die Funktion*

$$\begin{aligned} \pi : \quad & \mathbb{B}^n \to \bigcup_{i=0}^{2^n-1} \boldsymbol{\epsilon}_i \subset \mathbb{B}^{2^n} \\ & \boldsymbol{x} \mapsto \boldsymbol{\epsilon}_i \qquad \textit{mit} \qquad i = \mathrm{ind}(\boldsymbol{x}) \end{aligned} \tag{4.9}$$

ordnet einem Zustand $\boldsymbol{x}$ im Komponentenraum den zugehörigen Zustandsvektor $\boldsymbol{\chi} = \boldsymbol{\epsilon}_{\mathrm{ind}(\boldsymbol{x})}$ im Produktraum zu.
π ist bijektiv.

Beweis. Wir führen den Beweis mittels Induktion über n, die Breite des Leitungsbündels. Sei nun

$n = 1$: Das Leitungsbündel enthält nur eine Leitung e. Die äquivalenten Darstellungen der beiden möglichen Zustände von e sind

$$x_e = 0 \quad \leftrightarrow \quad \boldsymbol{\chi}_e = \begin{pmatrix}1\\0\end{pmatrix} \qquad \text{und} \qquad x_e = 1 \quad \leftrightarrow \quad \boldsymbol{\chi}_e = \begin{pmatrix}0\\1\end{pmatrix}$$

und die durch π vermittelte Zuordnung

$$\pi(0) = \boldsymbol{\epsilon}_{\mathrm{ind}(0)} = \boldsymbol{\epsilon}_0 = \begin{pmatrix}1\\0\end{pmatrix} \qquad \text{und} \qquad \pi(1) = \boldsymbol{\epsilon}_{\mathrm{ind}(1)} = \boldsymbol{\epsilon}_1 = \begin{pmatrix}0\\1\end{pmatrix} .$$

Nun kommt der Schritt

$n \to n+1$: Sei $\boldsymbol{e} = (e_0, \ldots, e_n)$ ein Bündel der Breite $n+1$, das ein Bit breiter ist als das Bündel $\boldsymbol{e}' = (e_0, \ldots, e_{n-1})$ der Breite n. Die für $\boldsymbol{e}$ äquivalenten Darstellungen der Zustände sind

$$\boldsymbol{x} = (x_0, \ldots, x_n) \qquad \leftrightarrow \qquad \boldsymbol{\chi} = \bigotimes_{k=n}^{0} \boldsymbol{\chi}_k \ .$$

Außerdem ist

$$\mathrm{ind}(\boldsymbol{x}) = \sum_{i=0}^{n} x_i \; 2^i = x_n \; 2^n + \sum_{i=0}^{n-1} x_i \; 2^i = x_n \; 2^n + \mathrm{ind}(\boldsymbol{x}') \tag{4.10}$$

und

$$\boldsymbol{\chi} = \bigotimes_{k=n}^{0} \boldsymbol{\chi}_k = \boldsymbol{\chi}_n \otimes \left(\bigotimes_{k=n-1}^{0} \boldsymbol{\chi}_k \right) = \boldsymbol{\chi}_n \otimes \boldsymbol{\chi}' \; . \tag{4.11}$$

Im Falle $x_n = 0$ erhalten wir mit (4.11)

$$\boldsymbol{\chi} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \otimes \boldsymbol{\chi}' = \begin{pmatrix} 1 \cdot \boldsymbol{\chi}' \\ 0 \cdot \boldsymbol{\chi}' \end{pmatrix}$$

und mit (4.9) und (4.10)

$$\boldsymbol{\chi} = \pi(\boldsymbol{x}) = \boldsymbol{\epsilon}_{\mathrm{ind}(\boldsymbol{x})} = \boldsymbol{\epsilon}_{\mathrm{ind}(\boldsymbol{x}')} = \begin{pmatrix} 1 \cdot \boldsymbol{\chi}' \\ 0 \cdot \boldsymbol{\chi}' \end{pmatrix} \; .$$

Die Eins im Einheitsvektor steht also wie erwartet nach wie vor an der $\mathrm{ind}(\boldsymbol{x}')$-ten Stelle.

Im Falle $x_n = 1$ erhalten wir mit (4.11)

$$\boldsymbol{\chi} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \otimes \boldsymbol{\chi}' = \begin{pmatrix} 0 \cdot \boldsymbol{\chi}' \\ 1 \cdot \boldsymbol{\chi}' \end{pmatrix}$$

und mit (4.9) und (4.10)

$$\boldsymbol{\chi} = \pi(\boldsymbol{x}) = \boldsymbol{\epsilon}_{\mathrm{ind}(\boldsymbol{x})} = \boldsymbol{\epsilon}_{(2^n + \mathrm{ind}(\boldsymbol{x}'))} = \begin{pmatrix} 0 \cdot \boldsymbol{\chi}' \\ 1 \cdot \boldsymbol{\chi}' \end{pmatrix} \; .$$

Die Eins im Einheitsvektor steht also wie erwartet an der $(2^n + \mathrm{ind}(\boldsymbol{x}'))$-ten Stelle, ist also um 2^n nach „unten“ verschoben.

Da jedem $\boldsymbol{x} \in \mathbb{B}^n$ genau ein $\boldsymbol{\epsilon}_i \in \mathbb{B}^{2^n}$ zugeordnet wird, ist π injektiv. Da es zu jedem $\boldsymbol{\epsilon}_i \in \mathbb{B}^{2^n}$ ein $\boldsymbol{x}$ mit $\boldsymbol{\epsilon}_i = \pi(\boldsymbol{x})$ gibt, ist π auch surjektiv. Somit ist π bijektiv. □

Folgerung 2. Weil π bijektiv ist, existiert π^{-1}. Wir können also mit Hilfe von π und π^{-1} zwischen der Komponenten- und Produktdarstellung hin und her wechseln.

Damit ist es äquivalent, ob wir Leitungen im Komponentenraum bündeln und dann in den Produktraum übergehen, oder ob wir erst in den Produktraum übergehen und die Leitungen dort bündeln:

Beispiel 4. Bündeln wir die beiden Leitungen der Zustände $x_0 = 0$ und $x_1 = 1$ im Komponentenraum, erhalten wir den Zustand $\boldsymbol{x} = 10$ und der zugeordnete Zustandsvektor ist

$$\boldsymbol{\chi} = \pi(\boldsymbol{x}) = \pi(10) = \boldsymbol{\epsilon}_2 = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} \; .$$

Gehen wir dagegen zunächst in den Produktraum und bilden wir dort das Leitungsbündel mit Hilfe des Kronecker-Produkts, erhalten wir

$$\boldsymbol{\chi} = \boldsymbol{\chi}_1 \otimes \boldsymbol{\chi}_0 = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \otimes \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} \; .$$

4.4.1.2 Matrixdarstellung einer Hyperpermutation

Wie sieht nun im Produktraum die Transformation eines Leitungsbündels aus? Ein Bündel der Breite n wird im Produktraum durch Einheitsvektoren der Dimension 2^n repräsentiert. Die einer Hyperpermutation zugeordnete lineare Abbildung wird also Einheitsvektoren der Dimension 2^n aufeinander abbilden. Damit kommen wir zu

Satz 3. *Sei* h_v *mit* $\boldsymbol{y}_i = \mathrm{h}_v(\boldsymbol{x}_i)$ *eine Hyperpermutation eines Knoten* v *der Dimension* $d = \dim(v)$. *Dann ist die im Produktraum zugeordnete lineare Abbildung* $\boldsymbol{H}_{\mathrm{h}_v} : \mathbb{B}^{2^d} \to \mathbb{B}^{2^d}$ *mit* $\pi(\boldsymbol{y}_i) = \boldsymbol{H}_{\mathrm{h}_v} \cdot \pi(\boldsymbol{x}_i)$ *gegeben durch:*

$$\boldsymbol{H}_{\mathrm{h}_v} = \big(\pi(\boldsymbol{y}_0), \pi(\boldsymbol{y}_1), \ldots, \pi(\boldsymbol{y}_{d-1})\big) \qquad \textit{mit} \qquad \begin{aligned} \boldsymbol{y}_i &= \mathrm{h}_v(\boldsymbol{x}_i) \\ \boldsymbol{x}_i &= \mathrm{ind}_{2^d}^{-1}(i) \end{aligned}$$

In Worten: In der i*-ten Spalte der Matrix* $\boldsymbol{H}_{\mathrm{h}_v}$ *steht der zum Zustand* $\boldsymbol{y}_i$ *gehörende Zustandsvektor. Der Zustand* $\boldsymbol{y}_i$ *ist der mit* h_v *transformierte Zustand* $\boldsymbol{x}_i$.

Beweis. Voraussetzung ist, daß im Komponentenraum $\boldsymbol{y}_i = \mathrm{h}_v(\boldsymbol{x}_i)$ ist. Wir müssen also zeigen, daß im Produktraum $\pi(\boldsymbol{y}_i) = \boldsymbol{H}_{\mathrm{h}_v} \cdot \pi(\boldsymbol{x}_i)$ gilt:

$$\begin{aligned} \boldsymbol{H}_{\mathrm{h}_v} \cdot \pi(\boldsymbol{x}_i) &= \boldsymbol{H}_{\mathrm{h}_v} \cdot \pi(\mathrm{ind}_{2^d}^{-1}(i)) = \big(\pi(\boldsymbol{y}_0), \pi(\boldsymbol{y}_1), \ldots, \pi(\boldsymbol{y}_{d-1})\big) \cdot \boldsymbol{\epsilon}_i \\ &= \pi(\boldsymbol{y}_i) \end{aligned}$$

□

Folgerung 3. Eine Hyperpermutation ist damit völlig äquivalent sowohl im Komponentenraum als auch im Produktraum darstellbar:

$$\boldsymbol{y} = \mathrm{h}(\boldsymbol{x}) \qquad \leftrightarrow \qquad \boldsymbol{\kappa} = \boldsymbol{H}_{\mathrm{h}} \cdot \boldsymbol{\chi}$$

Da die $\pi(\boldsymbol{y}_i)$ Einheitsvektoren sind, nimmt eine Hyperpermutation h_v im Produktraum die Form einer Permutationsmatrix $\boldsymbol{H}_{\mathrm{h}_v}$ an, die einen Satz von Einheitsvektoren permutiert.

An dieser Stelle sehen wir noch einmal den Zusammenhang zwischen Hyperpermutation und Permutation: Die im Komponentenraum nichtlineare Abbildung Hyperpermutation läßt sich erst im wesentlich größeren Produktraum als Permutation und damit als lineare Abbildung darstellen.

Zur Veranschaulichung von Satz 3 dient

Beispiel 5. Gegeben sei $\boldsymbol{y} = \mathrm{h}(\boldsymbol{x})$ und die Hyperpermutation

$\mathrm{h}:$

$\boldsymbol{x}$	$\boldsymbol{y}$
00	01
01	10
10	11
11	00

Wir wollen die Matrix $\boldsymbol{H}_{\mathrm{h}}$ finden, so daß $\pi(\boldsymbol{y}_i) = \boldsymbol{H}_{\mathrm{h}} \cdot \pi(\boldsymbol{x}_i)$. Nach Satz 3 besteht $\boldsymbol{H}_{\mathrm{h}}$ aus den Zustandsvektoren $\pi(\boldsymbol{y}_0), \ldots, \pi(\boldsymbol{y}_3)$. Mit

$$\pi(00) = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}; \qquad \pi(01) = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}; \qquad \pi(10) = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}; \qquad \pi(11) = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

ergibt sich

$$\boldsymbol{H}_{\mathrm{h}} = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} .$$

Transformieren wir nun z. B. $\mathrm{h}(00) = 01$ im Produktraum, erhalten wir

$$\boldsymbol{\kappa} = \boldsymbol{H}_{\mathrm{h}} \cdot \boldsymbol{\chi} = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} ,$$

was wie erwartet dem Zustand 01 im Komponentenraum entspricht.

Nachdem wir Leitungsbündel kombinieren und die Hyperpermutation als Matrix darstellen können, ist es möglich, die gesamte Netzfunktion eines HPNs als Matrix zu schreiben. Da die Größe dieser Matrix aber überexponentiell mit der Netzwerkgröße wächst, hat sie praktisch keinerlei Relevanz wird nur im Anhang A.1 auf Seite 173 aufgeführt.

4.4.1.3 Der Fokus

Bis jetzt können wir im Produktraum Leitungen nur bündeln, aber wie realisieren wir den umgekehrten Weg von einem Leitungsbündel zur einzelnen Leitung bzw. zu einem Teilbündel? Im Komponentenraum bilden wir den Fokus, indem wir eine oder mehrere Komponenten x_k eines Zustands $\boldsymbol{x}$ selektieren.

Im Produktraum entspricht die Bildung des Fokus der Projektion des Zustandsvektors auf einen Unterraum mit niedrigerer Dimension. Diese Projektion realisieren wir mit einer Projektionsmatrix $\boldsymbol{\Phi}$, mit der wir den Zustandsvektor eines Fokus erhalten.

Definition 17. Sei $\boldsymbol{x}' = {\phi^{\mathrm{ko}}}_{i,j}(\boldsymbol{x})$ ein Fokus der Breite $m = j - i + 1$ eines Zustands $\boldsymbol{x}$ der Breite n. Dann realisieren wir mit der Abbildung

$$\phi^{\mathrm{pr}} : \begin{array}{l} \mathbb{B}^{2^n} \to \mathbb{B}^{2^m} \\ \boldsymbol{\chi} \mapsto \boldsymbol{\Phi} \cdot \boldsymbol{\chi} \end{array}$$

wobei

$$\boldsymbol{\Phi} = \boldsymbol{v}(n-1-j) \otimes \boldsymbol{E}(2^m) \otimes \boldsymbol{v}(i) \qquad \text{mit} \qquad \begin{array}{l} \boldsymbol{E}(d) = \text{Einheitsmatrix der Dimension } d \\ \boldsymbol{v}(d) = \underbrace{(1, \ldots, 1)}_{2^d \text{ Einsen}} \end{array}$$

den äquivalenten Fokus im Produktraum. Falls $\boldsymbol{v}(d) = \boldsymbol{v}(0)$ fällt die entsprechende Kronecker-Multiplikation weg.

Betrachten wir dazu

Beispiel 6. Wir möchten aus einem Zustandsvektor

$$\boldsymbol{\chi} = \boldsymbol{\chi}_1 \otimes \boldsymbol{\chi}_0 = \begin{pmatrix} \chi_{1,0} \\ \chi_{1,1} \end{pmatrix} \otimes \begin{pmatrix} \chi_{0,0} \\ \chi_{0,1} \end{pmatrix} = \begin{pmatrix} \chi_{1,0}\,\chi_{0,0} \\ \chi_{1,0}\,\chi_{0,1} \\ \chi_{1,1}\,\chi_{0,0} \\ \chi_{1,1}\,\chi_{0,1} \end{pmatrix}$$

die beiden Zustandsvektoren $\boldsymbol{\chi}_0$ und $\boldsymbol{\chi}_1$ wieder „extrahieren“. Im Komponentenraum wäre klar, daß wir die jeweilige Leitung mit einem 1-Bit breiten Fokus aus dem Zustand $\boldsymbol{x}$ erhalten: $x_0 = {\phi^{\mathrm{ko}}}_{0,0}(\boldsymbol{x})$ und $x_1 = {\phi^{\mathrm{ko}}}_{1,1}(\boldsymbol{x})$. Ganz analog erhalten wir im Produktraum mit Hilfe von ϕ^{pr} die beiden gesuchten Zustandsvektoren $\boldsymbol{\chi}_0$ und $\boldsymbol{\chi}_1$. Die beiden Projektionsmatrizen sind

$$\boldsymbol{\Phi}_0 = \boldsymbol{v}(1) \otimes \boldsymbol{E}(2) \otimes \boldsymbol{v}(0) = (1,1) \otimes \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix}$$

$$\boldsymbol{\Phi}_1 = \boldsymbol{v}(0) \otimes \boldsymbol{E}(2) \otimes \boldsymbol{v}(1) = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \otimes (1,1) = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix}.$$

Mit ihnen erhalten wir die einzelnen Zustandsvektoren:

$$\boldsymbol{\Phi}_0 \cdot \boldsymbol{\chi} = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \chi_{1,0}\,\chi_{0,0} \\ \chi_{1,0}\,\chi_{0,1} \\ \chi_{1,1}\,\chi_{0,0} \\ \chi_{1,1}\,\chi_{0,1} \end{pmatrix} = \begin{pmatrix} (\chi_{1,0} + \chi_{1,1})\,\chi_{0,0} \\ (\chi_{1,0} + \chi_{1,1})\,\chi_{0,1} \end{pmatrix} = \begin{pmatrix} \chi_{0,0} \\ \chi_{0,1} \end{pmatrix} = \boldsymbol{\chi}_0$$

$$\boldsymbol{\Phi}_1 \cdot \boldsymbol{\chi} = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} \chi_{1,0}\,\chi_{0,0} \\ \chi_{1,0}\,\chi_{0,1} \\ \chi_{1,1}\,\chi_{0,0} \\ \chi_{1,1}\,\chi_{0,1} \end{pmatrix} = \begin{pmatrix} \chi_{1,0}\,(\chi_{0,0} + \chi_{0,1}) \\ \chi_{1,1}\,(\chi_{0,0} + \chi_{0,1}) \end{pmatrix} = \begin{pmatrix} \chi_{1,0} \\ \chi_{1,1} \end{pmatrix} = \boldsymbol{\chi}_1$$

Wir nutzen aus, daß die Terme $(\chi_{1,0} + \chi_{1,1})$ und $(\chi_{0,0} + \chi_{0,1})$ immer zu 1 evaluieren, da jeweils zwei verschiedene Komponenten desselben Einheitsvektors addiert werden.

4.4.2 Gewichtsvektoren

Wir kommen jetzt zur ersten Anwendung des eben entwickelten Formalismus. Dabei geht es um die einfache Frage:„Wie oft kommt welches Bitmuster auf einem Leitungsbündel vor?“ Können wir diese Frage beantworten, läßt sich eine Häufigkeitsverteilung der Zustände eines Leitungsbündels erstellen, die die Voraussetzung für die spätere Optimierung der Knotentabellen ist.

Stellen wir uns ein Leitungsbündel der Breite n vor, das zu verschiedenen Zeitpunkten in verschiedenen Zuständen ist. Wenn wir zählen wollen, wie oft welcher Zustand vorkommt, brauchen wir 2^n Zähler, einen für jeden Zustand. Bei jedem Zeitpunkt erhöhen wir den zum aktuellen Zustand gehörenden Zähler um eins. Dies liefert uns

Definition 18. Gegeben sei ein Leitungsbündel $\boldsymbol{e}$ der Breite d und eine Folge von Zustandsvektoren dieses Bündels in der Produktdarstellung $\boldsymbol{X}^{\mathrm{p}} = (\boldsymbol{\chi}_0, \ldots, \boldsymbol{\chi}_{n-1})$. Der zu dieser Zustandsfolge gehörende *Gewichtsvektor* wird durch

$$\boldsymbol{w} = \sum_{i=0}^{n-1} \boldsymbol{\chi}_i = \sum_{i=0}^{2^{d-1}} w_i\,\boldsymbol{\epsilon}_i$$

definiert. Eine Komponente w_i des Gewichtsvektors gibt also an, wie oft der Zustandsvektor $\boldsymbol{\chi} = \boldsymbol{\epsilon}_i$ des Leitungsbündels $\boldsymbol{e}$ in der Folge $\boldsymbol{X}^{\mathrm{p}}$ vorkommt. Kommen die Zustandsvektoren

aus K verschiedenen Klassen, können wir den Gewichtsvektor $\boldsymbol{w}$ in die *klassenspezifischen Gewichtsvektoren* $\boldsymbol{w}_k$ aufteilen. Ein klassenspezifischer Gewichtsvektor gibt also an, wie oft ein Zustandsvektor aus einer Klasse k vorgekommen ist. Für die klassenspezifischen Gewichtsvektoren $\boldsymbol{w}_k$ gilt:

$$\boldsymbol{w} = \sum_{k=0}^{K-1} \boldsymbol{w}_k$$

In dieser Darstellung geht die zeitliche Information der Zustandsfolge verloren. Für die Optimierung ist dies kein Nachteil, da dort nur die Häufigkeiten der Zustände in die Bewertung eingehen. Trotzdem kann man mit einem HPN auch zeitliche Muster lernen (siehe Teil „Experimente“ ab Seite 131).

Beispiel 7. Angenommen, wir haben auf einem Leitungsbündel der Breite zwei die Zustandsfolge $\boldsymbol{X}^{\mathrm{k}} = (10, 00, 01, 00, 11, 00, 01)$ in der Komponentendarstellung beobachtet. Dann ist der zugehörige Gewichtsvektor

$$\boldsymbol{w} = \sum_{\boldsymbol{x} \in \boldsymbol{X}^{\mathrm{k}}} \pi(\boldsymbol{x}) = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} + \cdots + \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 3 \\ 2 \\ 1 \\ 1 \end{pmatrix} ,$$

der Zustand 00 ist also 3 Mal vorgekommen, Zustand 01 ist 2 Mal vorgekommen usw.

Wir können mit einem Gewichtsvektor also kompakt beschreiben, was ein Knoten an seinem Eingang bzw. Ausgang für Bitmuster „gesehen“ hat.

4.4.2.1 Transformation eines Gewichtsvektors

Bis jetzt haben wir mit einer Hyperpermutation immer nur einen einzelnen Zustand transformiert. Wie ändert sich der Gewichtsvektor einer Zustands*folge* durch die Transformation mit einer bestimmten Hyperpermutation h? Dies beantwortet

Satz 4. *Gegeben sei der Gewichtsvektor $\boldsymbol{w}$ einer Zustandsfolge. Transformieren wir die Zustandsfolge mit der Hyperpermutation* h*, so ist der Gewichtsvektor $\boldsymbol{w}'$ der transformierten Zustandsfolge gegeben durch*

$$\boldsymbol{w}' = \boldsymbol{H}_{\mathrm{h}} \cdot \boldsymbol{w} \ .$$

Beweis. Der transformierte Gewichtsvektor $\boldsymbol{w}'$ einer Zustandsfolge läßt sich im Produktraum als gewichtete Summe der transformierten Leitungsvektoren $\boldsymbol{\epsilon}'_i = \boldsymbol{H}_{\mathrm{h}} \cdot \boldsymbol{\epsilon}_i$ schreiben,

$$\begin{aligned} \boldsymbol{w}' &= \sum_{i=0}^{2^n-1} w_i \, \boldsymbol{\epsilon}'_i = \sum_{i=0}^{2^n-1} w_i \, \boldsymbol{H}_{\mathrm{h}} \cdot \boldsymbol{\epsilon}_i = \boldsymbol{H}_{\mathrm{h}} \cdot \left(\sum_{i=0}^{2^n-1} w_i \, \boldsymbol{\epsilon}_i \right) \\ &= \boldsymbol{H}_{\mathrm{h}} \cdot \boldsymbol{w} \end{aligned}$$

wobei n die Bündeldimension ist. □

Folgerung 4. Wir erhalten den Gewichtsvektor am Knotenausgang aus dem Gewichtsvektor am Knoteneingang durch die Multiplikation des Gewichtsvektors am Knoteneingang mit der Matrix $\boldsymbol{H}_{\mathrm{h}}$. Wir können daher die Auswirkung einer Änderung von $\boldsymbol{H}_{\mathrm{h}}$ berechnen, *ohne* erneut die Zustandsfolge auswerten zu müssen.

4.4.2.2 Kombination von Leitungsbündeln

Was passiert, wenn wir zwei Leitungsbündel $\boldsymbol{x}_0$ und $\boldsymbol{x}_1$ miteinander zu einem Leitungsbündel $\boldsymbol{x}$ kombinieren, auf denen wir Folgen von Bitstrings beobachtet haben? Können wir aus den zugeordneten Gewichtsvektoren $\boldsymbol{w}_0$ und $\boldsymbol{w}_1$ den Gewichtsvektor $\boldsymbol{w}$ des kombinierten Leitungsbündels $\boldsymbol{x}$ berechnen, vielleicht wieder mit Hilfe des Kronecker-Produkts? Die Antwort gibt

Satz 5. *Gegeben seien zwei Leitungsbündel $\boldsymbol{x}_0$, $\boldsymbol{x}_1$ und die zugeordneten Gewichtsvektoren $\boldsymbol{w}_0$, $\boldsymbol{w}_1$. Das Leitungsbündel $\boldsymbol{x} = (\boldsymbol{x}_0, \boldsymbol{x}_1)$ sei die Kombiniation von $\boldsymbol{x}_0$ und $\boldsymbol{x}_1$ mit dem zugeordneten Gewichtsvektor $\boldsymbol{w}$. Dann ist es im allgemeinen* nicht *möglich, $\boldsymbol{w}$ aus $\boldsymbol{w}_0$ und $\boldsymbol{w}_1$ zu berechnen.*

Beweis. Beweis durch Widerspruch. Seien

$$\begin{aligned}
\boldsymbol{X}_0 &= (\boldsymbol{\chi}_0(0), \boldsymbol{\chi}_0(1), \ldots, \boldsymbol{\chi}_0(l-1)) \\
\boldsymbol{X}_1 &= (\boldsymbol{\chi}_1(0), \boldsymbol{\chi}_1(1), \ldots, \boldsymbol{\chi}_1(l-1)) \\
\boldsymbol{X} &= (\boldsymbol{\chi}(0), \boldsymbol{\chi}(1), \ldots, \boldsymbol{\chi}(l-1)) = (\boldsymbol{\chi}_1(0) \otimes \boldsymbol{\chi}_0(0), \boldsymbol{\chi}_1(1) \otimes \boldsymbol{\chi}_0(1), \ldots, \boldsymbol{\chi}_1(l-1) \otimes \boldsymbol{\chi}_0(l-1))
\end{aligned}$$

die drei Zustandsfolgen, aus denen die Gewichtsvektoren $\boldsymbol{w}_0$, $\boldsymbol{w}_1$ und $\boldsymbol{w}$ entstanden sind, der Zeitschritt steht als Argument. Angenommen es gäbe eine injektive Abbildung f, so daß

$$\boldsymbol{w} = f(\boldsymbol{w}_0, \boldsymbol{w}_1) \; .$$

Nun vertauschen wir in der Folge $\boldsymbol{X}_0$ zwei *ungleiche* Zustandsvektoren $\boldsymbol{\chi}_0(i) \neq \boldsymbol{\chi}_0(j)$ miteinander und berechnen die drei neuen Gewichtsvektoren:

$$\begin{aligned}
\boldsymbol{w}_0' &= \sum_{k=0}^{l-1} \boldsymbol{\chi}_0(k) = \boldsymbol{w}_0 \\
\boldsymbol{w}_1' &= \boldsymbol{w}_1 \\
\boldsymbol{w}' &= \sum_{k=0}^{l-1} \boldsymbol{\chi}_0(k) \otimes \boldsymbol{\chi}_1(k) \\
&= \boldsymbol{w} - \boldsymbol{\chi}_1(i) \otimes \boldsymbol{\chi}_0(i) - \boldsymbol{\chi}_1(j) \otimes \boldsymbol{\chi}_0(j) + \boldsymbol{\chi}_1(i) \otimes \boldsymbol{\chi}_0(j) + \boldsymbol{\chi}_1(j) \otimes \boldsymbol{\chi}_0(i) \\
&= \boldsymbol{w} + (\boldsymbol{\chi}_1(i) - \boldsymbol{\chi}_1(j)) \otimes \underbrace{(\boldsymbol{\chi}_0(j) - \boldsymbol{\chi}_0(i))}_{\neq 0 \text{ nach Voraussetzung}}
\end{aligned}$$

Aber auch der erste Term im Produkt $(\boldsymbol{\chi}_1(i) - \boldsymbol{\chi}_1(j)) \otimes (\boldsymbol{\chi}_0(j) - \boldsymbol{\chi}_0(i))$ ist im allgemeinen ungleich null, so daß folgt:

$$\boldsymbol{w}' \neq \boldsymbol{w}$$

Dagegen ist wegen $\boldsymbol{w}_0' = \boldsymbol{w}_0$ und $\boldsymbol{w}_1' = \boldsymbol{w}_1$

$$f(\boldsymbol{w}_0', \boldsymbol{w}_1') = f(\boldsymbol{w}_0, \boldsymbol{w}_1) \qquad \Rightarrow \qquad \boldsymbol{w}' = \boldsymbol{w} \; ,$$

woraus der Widerspruch folgt. □

Anschaulich gesprochen ist die Begründung für Satz 5, daß durch die Bildung eines Gewichtsvektors die Zeitinformation der Zustandsfolge verloren geht. Daher ist aus den Gewichtsvektoren $\boldsymbol{w}_0$ und $\boldsymbol{w}_1$ nicht mehr zu entnehmen, welche Zustände auf den Bündeln $\boldsymbol{x}_0$ und $\boldsymbol{x}_1$ *gleichzeitig* erschienen sind.

Benötigen wir die Statistik einer Kombination von Leitungsbündeln, bleibt uns nichts anderes übrig, als aus der Zustandsfolge des Gesamtbündels den Gewichtsvektor auszuzählen.

4.4.2.3 Bildung des Fokus

Das „Gegenteil“ der Kombiniation von Leitungsbündeln ist die Bildung eines Fokus. In diesen Fall können wir den Gewichtsvektor eines Fokus aus dem Gewichtsvektor des ursprünglichen Leitungsbündels berechnen:

Satz 6. *Sei $\boldsymbol{\chi}$ ein Zustandsvektor mit dem dazugehörenden Gewichtsvektor $\boldsymbol{w}$. Sei außerdem $\boldsymbol{\chi}' = {\phi^{\mathrm{pr}}}_{i,j}(\boldsymbol{\chi})$ ein Fokus der Breite $m = j - i + 1$ auf $\boldsymbol{\chi}$. Dann ist der zu $\boldsymbol{\chi}'$ gehörende Gewichtsvektor $\boldsymbol{w}'$ durch*

$$\boldsymbol{w}' = \boldsymbol{\Phi} \cdot \boldsymbol{w}$$

gegeben, wobei $\boldsymbol{\Phi} = \boldsymbol{v}(n-1-j) \otimes \boldsymbol{E}(2^m) \otimes \boldsymbol{v}(i)$, wie in Definition 17 angegeben.

Beweis. Die zum Leitungsbündel bzw. Fokus gehörenden Zustandsfolgen seien

$$\begin{aligned} \boldsymbol{X} &= (\boldsymbol{\chi}(0), \boldsymbol{\chi}(1), \ldots, \boldsymbol{\chi}(l-1)) \qquad \text{bzw.} \\ \boldsymbol{X}' &= (\boldsymbol{\chi}'(0), \boldsymbol{\chi}'(1), \ldots, \boldsymbol{\chi}'(l-1)) \end{aligned}$$

Dann ist $\boldsymbol{w}'$ nach Definition 18 und mit $\boldsymbol{\chi}'_i = \boldsymbol{\Phi} \cdot \boldsymbol{\chi}_i$ (Definition 17):

$$\boldsymbol{w}' = \sum_{i=0}^{l-1} \boldsymbol{\chi}'_i = \sum_{i=0}^{l-1} \boldsymbol{\Phi} \cdot \boldsymbol{\chi}_i = \boldsymbol{\Phi} \sum_{i=0}^{l-1} \boldsymbol{\chi}_i = \boldsymbol{\Phi} \cdot \boldsymbol{w}$$

□

Ist also der Gewichtsvektor $\boldsymbol{w}$ eines Leitungsbündels bekannt, läßt sich der Gewichtsvektor eines beliebigen Fokus des Leitungsbündels aus $\boldsymbol{w}$ berechnen.

Beispiel 8. Angenommen, wir haben auf einem Leitungsbündel der Breite zwei mit dem Zustand $\boldsymbol{x}$ die Zustandsfolge $\boldsymbol{X}^{\mathrm{k}} = (10, 00, 01, 00, 11, 00, 01)$ beobachtet, woraus sich der Gewichtsvektor $\boldsymbol{w} = (3, 2, 1, 1)^{\mathrm{T}}$ ergibt. Wir berechnen die Gewichtsvektoren $\boldsymbol{w}_0$ und $\boldsymbol{w}_1$ der Leitungen $x_0 = {\phi^{\mathrm{ko}}}_{0,0}(\boldsymbol{x})$ und $x_1 = {\phi^{\mathrm{ko}}}_{1,1}(\boldsymbol{x})$ nach Satz 6 zu

$$\boldsymbol{w}_0 = \boldsymbol{\Phi}_0 \cdot \boldsymbol{w} = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 3 \\ 2 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 4 \\ 3 \end{pmatrix}$$

$$\boldsymbol{w}_1 = \boldsymbol{\Phi}_1 \cdot \boldsymbol{w} = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} 3 \\ 2 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 5 \\ 2 \end{pmatrix}$$

Die Gewichtsvektoren $\boldsymbol{w}_0$ und $\boldsymbol{w}_1$ haben die erwarteten Werte: Die Leitung x_0 hat in der Zustandsfolge $\boldsymbol{X}^{\mathrm{k}}$ viermal den Zustand 0 und dreimal den Zustand 1, die Leitung x_1 dagegen hat fünfmal den Zustand 0 und zweimal den Zustand 1.

4.4.3 Die Dynamik eines rückgekoppelten Knotens

Formal haben wir die Dynamik eines Hyperpermutationsnetzwerks schon in Abschnitt 4.2.2 eingeführt. Nun zeigen wir, wie wir die Dynamik rückgekoppelter Netzwerkknoten wie in Abbildung 4.10 im Produkraum schreiben können.

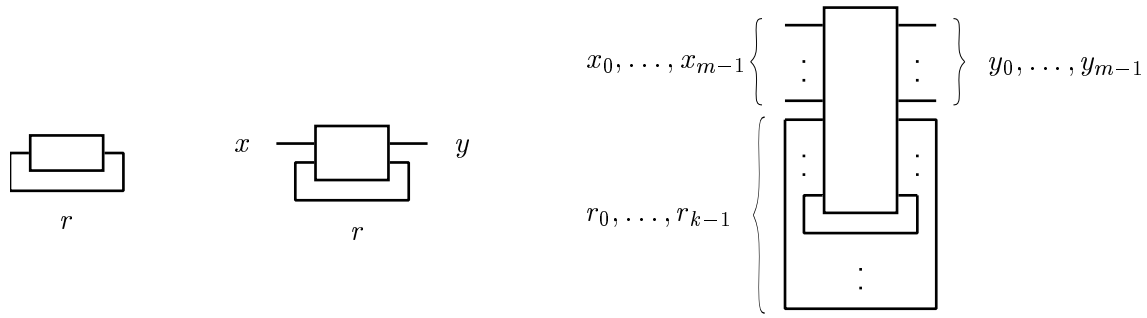


Abbildung 4.10: Rückgekoppelte Hyperpermutationsknoten. Links ein geschlossener Knoten mit einer rückgekoppelten Leitung, in der Mitte ein offener Knoten mit einer rückgekoppelten Leitung und einer Ein- und Ausgangsleitung, rechts ein allgemeiner Knoten mit k rückgekoppelten Leitungen und m Ein- und Ausgangsleitungen.

Geschlossener 1-Bit-Knoten

Ein geschlossener 1-Bit-Knoten wie in Abbildung 4.10 ist zugegebenermaßen recht unspektakulär. Da es keine aus dem System herausführenden Leitungen gibt, kann man mit einem geschlossenen Knoten keine „Daten verarbeiten“. Wir verdeutlichen an diesem Knoten die beiden möglichen Beschreibungsweisen, den Komponentenraum und den Produktraum. Die einzige im System vorkommende Leitung ist die Rückkopplungsleitung. Ihr Zustand zum Zeitpunkt t ist $r(t)$. Die Dynamik des Systems wird im Komponentenraum durch

$$r(t+1) = \mathrm{h}(r(t))$$

und im Produktraum durch

$$\boldsymbol{\chi}_\mathrm{r}(t+1) = \boldsymbol{H} \cdot \boldsymbol{\chi}_\mathrm{r}(t))$$

beschrieben, wobei $\boldsymbol{\chi}_\mathrm{r}$ für den Zustandvektor der rückgekoppelten Leitung steht. Bei einem 1-Bit-Knoten gibt es nur zwei mögliche Hyperpermutationen, nämlich die identische Abbildung und die Negation, im Produktraum sind sie:

$$\boldsymbol{H}_\mathrm{I}: \quad \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \qquad \boldsymbol{H}_\mathrm{N}: \quad \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

Die zeitliche Abfolge $r(0), r(1), r(2), \ldots$ des Zustands $r(t)$ der rückgekoppelten Leitung ist also bei $\boldsymbol{H}_\mathrm{I}$ und dem angenommenen Anfangszustand $r(0) = 0$

$$0, 0, 0, 0, \ldots \qquad \text{bzw.} \qquad \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \ldots$$

und bei $\boldsymbol{H}_\mathrm{N}$

$$0, 1, 0, 1, \ldots \qquad \text{bzw.} \qquad \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \ldots$$

Offener rückgekoppelter 2-Bit-Knoten

Ein offener rückgekoppelter 2-Bit-Knoten wie in Abbildung 4.10 ist schon interessanter als der eben beschriebene Fall. Bei diesem Knoten haben wir drei Leitungen:

1. Die Eingangsleitung $x(t)$.
2. Die Ausgangsleitung $y(t)$.
3. Die rückgekoppelte Leitung $x(t)$.

Über die beiden externen Leitungen x und y ist der Knoten mit der Außenwelt verbunden, die rückgekoppelte Leitung r ist eine interne Leitung. Das zwei Bit lange Eingangssymbol $\boldsymbol{x}$ des Knotens wird jetzt aus den beiden ein Bit langen Bitstrings der Leitungen x und r zusammengesetzt. Außerdem muß das zwei Bit lange Ausgangssymbol $\boldsymbol{y}$ des Knotens auf die beiden Leitungen y und r projiziert werden. Wir verwenden im folgenden die Darstellung im Produktraum und bezeichnen die zu den Leitungen x, y und r gehörenden Zustandsvektoren mit $\boldsymbol{\chi}_\mathrm{x}$, $\boldsymbol{\chi}_\mathrm{y}$ und $\boldsymbol{\chi}_\mathrm{r}$ und den Zustandsvektor des Knoteneinbzw. ausgangs mit $\boldsymbol{\chi}$ bzw. $\boldsymbol{\kappa}$.

In jedem Zeittakt führt der Knoten mit einer der $2^2! = 24$ möglichen Hyperpermutationen $\boldsymbol{H}$ die Operation

$$\boldsymbol{\kappa}(t+1) = \boldsymbol{H} \cdot \boldsymbol{\chi}(t) \tag{4.12}$$

aus. Mit Hilfe des Kronecker-Produktes können wir den Zustandsvektor am Knoteneingang $\boldsymbol{\chi}$ aus $\boldsymbol{\chi}_\mathrm{x}$ und $\boldsymbol{\chi}_\mathrm{r}$ zusammensetzen:

$$\boldsymbol{\chi} = \boldsymbol{\chi}_\mathrm{r} \otimes \boldsymbol{\chi}_\mathrm{x} \tag{4.13}$$

Mit den Projektionsmatrizen

$$\boldsymbol{\Phi}_0 = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix} \qquad \text{und} \qquad \boldsymbol{\Phi}_1 = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix}$$

projizieren wir das Ausgangssymbol $\boldsymbol{\kappa}$ auf $\boldsymbol{\chi}_\mathrm{y}$ und $\boldsymbol{\chi}_\mathrm{r}$:

$$\begin{aligned} \boldsymbol{\chi}_\mathrm{y} &= \boldsymbol{\Phi}_0 \cdot \boldsymbol{\kappa} \\ \boldsymbol{\chi}_\mathrm{r} &= \boldsymbol{\Phi}_1 \cdot \boldsymbol{\kappa} \end{aligned} \tag{4.14}$$

Setzen wir Gleichung (4.13) und (4.14) in Gleichung (4.12) ein, läßt sich die Dynamik des Systems folgendermaßen schreiben:

$$\begin{aligned} \boldsymbol{\chi}_\mathrm{y}(t+1) &= \boldsymbol{\Phi}_0 \cdot \boldsymbol{H} \cdot (\boldsymbol{\chi}_\mathrm{r}(t) \otimes \boldsymbol{\chi}_\mathrm{x}(t)) \\ \boldsymbol{\chi}_\mathrm{r}(t+1) &= \boldsymbol{\Phi}_1 \cdot \boldsymbol{H} \cdot (\boldsymbol{\chi}_\mathrm{r}(t) \otimes \boldsymbol{\chi}_\mathrm{x}(t)) \end{aligned}$$

Da nun zu jedem Zeitpunkt t die Eingangsleitung $\boldsymbol{\chi}_\mathrm{x}(t)$ möglicherweise einen neuen Wert hat, kann der Zustand des Systems nur soweit vorausgesagt werden, wie auch $\boldsymbol{\chi}_\mathrm{x}(t)$ bekannt ist. Wir können die Operation des Knotens in der Zeit auch herumdrehen: Kennen wir zu einem Zeitpunkt t die Folge $\boldsymbol{\chi}_\mathrm{y}(0), \ldots, \boldsymbol{\chi}_\mathrm{y}(t)$ und den Wert $\boldsymbol{\chi}_\mathrm{r}(t)$, so können wir die Folge $\boldsymbol{\chi}_\mathrm{x}(t-1), \ldots, \boldsymbol{\chi}_\mathrm{x}(0)$, also die vergangenen Eingangsdaten berechnen. Dies folgt aus der Umkehrbarkeit der Hyperpermutationen.

Allgemeiner rückgekoppelter n-Bit-Knoten

Ein allgemeiner rückgekoppelter Knoten wie in Abbildung 4.10 hat m Ein- bzw. Ausgangsleitungen $x_0, \ldots, x_{m-1}$ bzw. $y_0, \ldots, y_{m-1}$ und k rückgekoppelte Leitungen $r_0, \ldots, r_{k-1}$. Er ist also insgesamt $n = m + k$ Bits breit. Die rückgekoppelten Leitungen stellen einen k Bit großen

Speicher dar, der 2^k verschiedene Zustände annehmen kann. Wir können nun die Leitungen fortlaufend mittels neuer Variablen $x_0, \ldots, x_{2m+k-1}$ indizieren:

$$x_j = \begin{cases} x_j & : \text{ falls } j \in [0, \ldots, m-1] \\ r_{j-m} & : \text{ falls } j \in [m, \ldots, m+k-1] \\ y_{j-m-k} & : \text{ falls } j \in [m+k, \ldots, 2m+k-1] \end{cases}$$

In der Folge der x_j kommen also erst die Eingangsleitungen, dann die rückgekoppelten Leitungen und zum Schluß die Ausgangsleitungen. Wir ordnen wieder jeder Leitung x_i einen Zustandsvektor $\boldsymbol{\chi}_i$ zu. Der Zustandsvektor des Knotenein- bzw. -ausgangs ist $\boldsymbol{\chi}$ bzw. $\boldsymbol{\kappa}$. Nun läßt sich die Dynamik des Systems wieder kompakt schreiben, ganz analog zum 2-Bit-Knoten. Pro Takt wird

$$\boldsymbol{\kappa}(t+1) = \boldsymbol{H} \cdot \boldsymbol{\chi}(t) \tag{4.15}$$

ausgeführt. Das Eingangssymbol $\boldsymbol{\chi}$ ist:

$$\boldsymbol{\chi} = \bigotimes_{j=m+k-1}^{0} \boldsymbol{\chi}_j \tag{4.16}$$

Mit den Projektionsmatrizen $\boldsymbol{\Phi}_j$ projizieren wir das Ausgangssymbol $\boldsymbol{\kappa}$ auf die j-te Leitung

$$\boldsymbol{\chi}_j = \boldsymbol{\Phi}_j \cdot \boldsymbol{\kappa} \tag{4.17}$$

Mit Gleichung (4.15), (4.16) und (4.17), erhalten wir für die Dynamik des Systems mit $j \in [m, \ldots, 2m+k-1]$:

$$\boldsymbol{\chi}_j(t+1) = \boldsymbol{\Phi}_j \cdot \boldsymbol{H} \cdot \left(\bigotimes_{j=m+k-1}^{0} \boldsymbol{\chi}_j(t) \right) \tag{4.18}$$

Die Leitungen $\boldsymbol{\chi}_j(t)$ mit $j \in [0, \ldots, m-1]$ sind durch die Eingangsdaten festgelegt.

4.5 Vergleich mit vorhandenen mathematischen Modellen

Wir haben nun den Formalismus von Hyperpermutationsnetzwerken kennengelernt und werden jetzt zusammenfassen, welche Unterschiede bzw. Gemeinsamkeiten es bei HPNs und bei neuronalen Netzen, RAM-basierten neuronalen Netzen und Funktionen-Netzen gibt.

4.5.1 Kontinuierliche neuronale Netze

Wir können die Unterschiede zwischen neuronalen Netzen und Hyperpermutationsnetzwerken in zwei Klassen unterteilen: Die erste Klasse betrifft Unterschiede zwischen HPNs und neuronalen Netzen (kontinuierlichen wie RAM-basierten) im allgemeinen, die zweite Klasse von Unterschieden betrifft HPNs und kontinuierliche neuronale Netze im speziellen. Im Grunde sind die meisten der folgenden Punkte implizit in Kapitel 3 und 4 schon vorhanden, nur werden wir sie hier einmal zusammenfassen.

Unterschiede zwischen HPNs und neuronalen Netzen im allgemeinen:

- Neuronale Netze gehen von dem konnektionistischen Paradigma aus, was Folgen für die üblicherweise verwendenten Netzwerktopologien hat:

  - Jedes Neuron darf nur eine Ausgangsleitung haben. Ein HPN-Knoten hat im allgemeinen genausoviele Ausgangsleitungen wie Eingangsleitungen.
  - In einem neuronalen Netz können mehrere Neuronen denselben Ausgang eines Neurons als Eingang haben. Bei HPNs kann man Leitungen formal nur mit Hilfe von Blindbits duplizieren.

  Diese beiden Punkte wirken sich besonders bei RAM-basierten neuronalen Netzen aus, wie wir noch sehen werden.

- Die Netzfunktion eines neuronalen Netzes ist im allgemeinen irreversibel. Die Netzfunktion eines HPNs ist dagegen reversibel, wenn man die Ausgänge aller Knoten aus dem Netz herausführt. Die Datenreduktion mit einem HPN ähnelt daher der Datenreduktion mit der Hauptachsentransformation: Zunächst transformiert man die Daten verlustfrei in einen Raum, in dem die Daten für die Aufgabe besser zugänglich sind, und dann beachtet man nur die wichtigsten Komponenten.

- Die zu optimierende Funktion eines HPN ist nicht der mittlere quadratische Fehler, sondern die sogenannte *Information* (*nicht* die Shannon'sche), die wir im nächsten Kapitel einführen werden. Sie ist für die Optimierung eines solchen diskreten Netzes besser geeignet, als der mittlere quadratische Fehler, der bei diskreten Netzen nur in Ausnahmefällen eine sinnvolle Kostenfunktion ist.

Es ist bekannt, daß man mit kontinuierlichen neuronalen Netzen als universellen Approximatoren jedes beliebige RAM-basierte Netzwerk simulieren kann. Gurney (1992b) hat mit Hilfe der Σ–Π–Unit gezeigt, wie dies im einzelnen durchgeführt werden kann. Dies hat aber einen hohen Preis: Zum einen muß man im allgemeinen beliebig viele Neuronen zur Verfügung haben. Zum anderen simuliert man die auf dem Digitalcomputer sehr schnellen und einfachen Bitoperationen mit Skalarmultiplikationen von Vektoren mit Gleitkommazahlen. Dies ist also in der Praxis kein gangbarer Weg.

Es gibt auch spezielle Unterschiede zwischen HPNs und kontinuierlichen neuronalen Netzen. Diese Unterschiede bestehen teilweise auch zwischen RAM-basierten neuronalen Netzen und kontinuierlichen neuronalen Netzen.

- Ein kontinuierliches neuronales Netz ist typischerweise auf die arithmetischen Operationen „Addition“ und „Multiplikation“ im Kontinuum angewiesen. Je nach Neuronentyp kommt dann noch eine Schwellwertfunkion oder andere Nichtlinearitäten wie die logistische Funktion oder der Tangens Hyperbolicus hinzu. Daher muß eine Hardware, auf der ein kontinuierliches neuronales Netz simuliert wird, Gleitkommaarithmetik beherrschen. Bei einem HPN ist keine Gleitkommaarithmetik nötig.

- Das mathematische Modell eines HPNs besteht nur aus Knoten, die diskrete Lookup-Tabellen beinhalten und binären Leitungen, die diese Knoten miteinander verbinden. Daher läßt sich ein HPN im Gegensatz zu einem kontinuierlichen neuronalen Netz *direkt* in Digitalhardware realisieren. Man benötigt also nicht nur keine Gleitkommaarithmetik, man benötigt *überhaupt keine* arithmetikfähige Hardware, sondern nur „Leitungen und Speicher“.

Natürlich gibt es auch Gemeinsamkeiten zwischen neuronalen Netzen und HPNs. Unter anderem sind beide Ansätze statistische Verfahren zur Mustererkennung, die durch Training mit Beispielen eine Wahrnehmungsaufgabe erlernen sollen. In Bezug auf das Training müssen sich daher beide Ansätze mit demselben Problem kämpfen: Zum Training braucht man (gekennzeichnete) Trainingsdaten in ausreichender Menge und ebenso Testdaten in ausreichender

Menge, um statistisch signifikante Aussagen über die Leistungsfähigkeit eines Mustererkennungssystems treffen zu können.

4.5.1.1 Blindbits und das konstante Neuron

Wir haben bereits gesehen, wie wir mit formalen Blindbits die funktionale Kapazität eines Hyperpermutationsknoten erhöhen können. Bei einem neuronalen Netzes gibt es ein ähnliches Phänomen. Dort steht im Argument der Aktivierungsfunktion $f(x)$ normalerweise ein Skalarprodukt

$$x = \sum_i \alpha_i \, x_i$$

aus dem Gewichtsvektor $\boldsymbol{\alpha}$ und dem Eingangsvektor $\boldsymbol{x}$ des Neurons. Gilt für die Aktivierungsfunktion $f(0) = 0$, wie es z. B. bei der Identischen Funktion, den linearen Funktionen, $\sin(x)$, $\tanh(x)$ etc. der Fall ist, gilt für $\boldsymbol{x} = \mathbf{0}$: $f(\sum_{i=1}^{n} \alpha_i \, x_i) = 0$. Möchte man erreichen, daß trotz $\boldsymbol{x} = \mathbf{0}$ die Aktivierungsfunktion $f(x) \neq 0$ ist, kann man dem Neuronalen Netz ein Neuron hinzufügen, das einen konstanten Ausgangswert $c \neq 0$ hat. Hat ein anderes Neuron dieses „konstante Neuron“ als k-ten Eingang, ist der neue Ausgangswert

$$x = \alpha_k \cdot c + \sum_{\substack{i=0 \\ i \neq k}}^{n-1} \alpha_i \, x_i$$

und mit $\alpha_k \neq 0$ ist trotz $\boldsymbol{x} = \mathbf{0}$ die Forderung $f(x) = \alpha_k \cdot c \neq 0$ erfüllbar. Durch die Einführung des konstanten Neurons hat man also die Menge der Abbildungen, die das Neuron repräsentieren kann, vergrößert. Dies ähnelt der Einführung eines Blindbits an einen Hyperpermutationsknoten.

4.5.1.2 Unterteilung des Raumes

Um ein Gefühl dafür zu bekommen, auf welche Weise ein Schwellwert-Neuron und ein Hyperpermutationsknoten einen diskreten Eingaberaum partionieren, sehen wir uns als Beispiel den $\mathbb{B}^2$ an. Das Neuron und der Knoten sehen aus wie in Abbildung 4.11 gezeigt. Mit diesen Bauteilen läßt sich der $\mathbb{B}^2$ wie in Abbildung 4.12 partionieren. Wie wir sehen, hat ein HPN-Knoten gegenüber einem kontinuierlichen Schwellwert-Neuron zwei weitere nichtlineare Partionierungmöglichkeiten. Diese entsprechen gerade dem XOR der beiden Eingangsbits. Solange wir nur ein Ausgangsbit eines HPN-Knoten nutzen, gelten diese und die folgenden Überlegungen genauso für ein RAM-basiertes Neuron.

Abbildung 4.11: Neuron und HPN-Knoten mit zwei Bit am Eingang

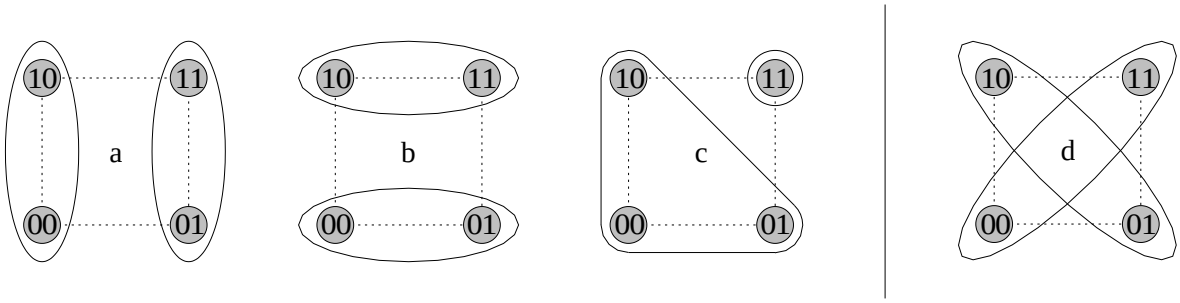


Abbildung 4.12: Partionierung des Einheitsquadrates $\mathbb{B}^2$ mit einem Neuron und einem HNP-Knoten. Die drei Partionierungen auf der linken Seite sind lineare Partionierungen den Einheitsquadrates. Von der Partition c gibt es noch drei weitere jeweils um 90° gedrehte Varianten. Um Partition c zu realisieren, braucht ein HPN-Knoten ein Blindbit. Partition d läßt sich mit einem kontinuierlichen Neuron nicht realsisieren, mit einem HPN-Knoten dagegen schon. Daher ist auch die Lösung des XOR-Problems mit einem einzigen HPN-Knoten möglich, wie wir noch sehen werden.

4.5.1.3 VC-Dimension eines Neurons und eines HPN-Knotens

Um die funktionale Kapazität eines Hyperpermutationsknoten und eines Schwellwert-Neurons zu quantifizieren, berechnen wir für beide die VC-Dimension. Die VC-Dimension ist folgendermaßen definiert (siehe auch Vapnik, 1995, Kapitel 3.5):

> **Die VC-Dimension einer Menge von Indikatorfunktionen** (Vapnik und Chervonenkis, 1968, 1971): Die VC-Dimension einer Menge von Indikatorfunktionen $Q(\boldsymbol{z}, \boldsymbol{\alpha})$, mit dem Parameter $\boldsymbol{\alpha}$, ist die größte Zahl h von Vektoren $z_1, \ldots, z_h$, die von den Funktionen $Q(\boldsymbol{z}, \boldsymbol{\alpha})$ auf alle 2^h Weisen in zwei Klassen eingeteilt werden können. Eine Indikatorfunktion teilt eine gegebene Menge von Vektoren in zwei Teilmengen: Die eine Teilmenge, auf der die Indikatorfunktion den Wert null annimmt und die andere Teilmenge, auf der die Indikatorfunktion den Wert eins annimmt.

Eine Indikatorfunktion ist z. B. ein Schwellwert-Neuron mit n Eingängen. Als Beispiel betrachten wir die Menge der *linearen Indikatorfunktionen* (Vapnik)

$$Q(\boldsymbol{z}, \boldsymbol{\alpha}) = \Theta\left(\sum_{i=0}^{n-1} \alpha_i \, z_i + \theta\right) ,$$

wobei $\Theta(x)$ die Sprungfunktion ist. Die VC-Dimension für $\boldsymbol{z} \in \mathbb{B}^n$ ist $h = n + 1$, da man $n + 1$ Vektoren mit einer $n - 1$-dimensionalen Hyperebene auf alle 2^h Weisen in zwei Klassen aufgeteilen kann (Vapnik, 1995). Hier ist die VC-Dimension gerade gleich der Zahl der freien Parameter.

Bei einem Hyperpermutationsknoten kommen wir folgendermaßen auf die VC-Dimension: Zunächst erhalten wir aus einem Knoten der Dimension n eine Indikatorfunktion, indem wir vom n Bit breiten Ausgang des Knoten nur ein Bit berücksichtigen, also auf einen eindimensionalen Unterraum projizieren. Nun gibt es 2^n mögliche Eingangsvektoren aus dem $\mathbb{B}^n$, auf denen das betrachtete Ausgangsbit $2^n/2 = 2^{n-1}$ mal den Wert null und ebenso oft den Wert eins annimmt. Beschränken wir uns am Eingang auf 2^{n-1} Vektoren, können wir jedem dieser Vektoren auf dem beobachteten Bit eine eins oder null zuordnen, unabhängig davon, welche Werte den anderen 2^{n-1} Vektoren zugeordnet wurden. Die Beschränkung auf 2^{n-1} Vektoren am

Eingang können wir z. B. dadurch erreichen, daß wir ein Eingangsbit immer konstant halten. Die VC-Dimension ist also $h = 2^{n-1}$, da wir gerade h Eingangsvektoren auf alle 2^h Weisen in zwei Klassen aufteilen können. Einen Vergleich der beiden VC-Dimensionen zeigt Abbildung 4.13.

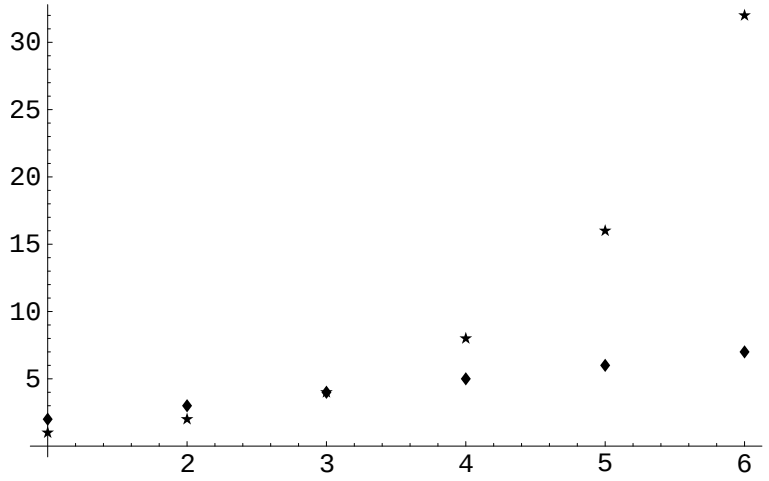


Abbildung 4.13: VC-Dimension eines Schwellwert-Neurons (Rauten) und eines HPN-Knotens (Sterne). Die Abszisse ist die Dimension n des Raums $\mathbb{B}^n$, die Ordinate die zugehörige VC-Dimension. Für $n = 3$ haben Schwellwert-Neuron und Hyperpermutationsknoten die VC-Dimension vier. Die Nichtlinearität des Hyperpermutationsknotens zeigt sich in der größeren VC-Dimension für $n > 3$.

Die funktionalen Mächtigkeit eines Neurons bzw. HPN-Knotens wächst mit der VC-Dimension. Ein Hyperpermutationsknoten hat im Gegensatz zu einem kontinuierlichen Neuron nicht die Einschränkung, daß er nur linear separierbare Partitionen im $\mathbb{B}^n$ realisieren kann. Wie drückt sich dieser Unterschied in der Zahl der realisierbaren Funktionen aus? Tabelle 4.4 zeigt die Zahl der realisierbaren Funktionen für ein Neuron und einen HPN-Knoten. Weitere Untersuchungen zur funktionalen Mächigkeit sind in Al-Alawi und Stonham (1992) zu finden.

4.5.2 RAM-basierte Neuronale Netze

Die Unterschiede zwischen RAM-basierten neuronalen Netzen und Hyperpermutationsnetzwerken sind naturgemäß deutlich geringer als zwischen analogen neuronalen Netzen und HPNs:

- Ein reines RAM-basiertes Neuron hat nur eine binäre Ausgangsleitung. Es kann daher nur die beiden verschiedenen Symbole 0 und 1 emittieren. Ein HPN-Knoten der Dimension d kann dagegen 2^d Symbole emittieren. Ist auf $k \in [1, d]$ Ausgangsbits des Knotens ein Fokus, so reduziert der Knoten die Zahl der möglichen Symbole von 2^d am Knoteneingang auf 2^k am Knotenausgang. Man kann einen HPN-Knoten der Dimension d mit d binären Neuronen simulieren. Die durch d binäre Neuronen vermittelte Abbildung $f : \mathbb{B}^d \to \mathbb{B}^d$ ist aber im Gegensatz zum HPN-Knoten nicht umkehrbar, da die Abbildungen der einzelnen binären Neuronen nicht umkehrbar sind. Wegen der Reversibilität des HPN sind die Systemgrößen, die wir noch definieren werden, Erhaltungsgrößen eines HPNs.

- Wie schon erwähnt, sind bei einem HPN die klassenspezifischen Rückschlußwahrscheinlichkeiten direkt zugänglich. Dies ist bei RAM-basierten neuronalen Netzen nicht ohne weiteres der Fall, worauf wir im nächsten Abschnitt genauer eingehen werden.

Eingangsbits n	Zahl der realisierbaren Funktionen: Ein Ausgangsbit eines HPN-Knotens	McCullough & Pitts – Neuron
1	2	4
2	4	14
3	16	104
4	256	1882
5	65536	$9.5 \cdot 10^4$
6	$4.3 \cdot 10^9$	$1.5 \cdot 10^7$
7	$1.9 \cdot 10^{19}$	$8.4 \cdot 10^9$
8	$3.4 \cdot 10^{38}$	$1.8 \cdot 10^{13}$

Tabelle 4.4: Zahl der realisierbare Funktionen $f : \mathbb{B}^n \to \mathbb{B}$ eines Neurons und eines HPN-Knotens, von dem nur ein einzelnes Ausgangsbit beachtet wird. Die funktionale Mächtigkeit des HPN-Knotens ist $2^{(2^{n-1})}$, da wir für 2^{n-1} Tabelleneinträge wählen können, ob wir auf das betrachtete Augangsbit eine 0 oder 1 schreiben. Die Zahlen für das analoge Neuron sind aus Muroga et al. (1970). Hat der HPN-Knoten ein zusätzliches Blindbit, ist seine funktionale Mächtigkeit 2^{2^n}, wodurch die Zahl der realisierbaren Funktionen in der Tabelle beim HPN-Knoten sozusagen um eine Zeile „nach oben rutschen“.

- Einige verwendete Neuronentypen wie z. B. der *Probabilistic Logic Node*, der *Probabilistic RAM-Node* oder das *Goal-Seeking Neuron* sind bedingt durch ihre kompliziertere Architektur nicht mehr direkt Hardware implementierbar. Diese Neuronen sind zwar immer noch leichter in Hardware zu implementieren als ein analoges Neuron. Relativ zum ursprünglichen RAM-Knoten muß man aber je nach Neuron mehr oder weniger Zusatzaufwand betreiben. Ein HPN-Knoten ist dagegen ein „reiner“ RAM-Knoten.

4.5.3 Das Problem der Rückschlußwahrscheinlichkeiten

Analoge neuronale Netze können z. B. bei der Eins-aus-N-Kodierung durch verschieden große Anteile im Ausgangsvektor Auskunft über die Sicherheit einer Entscheidung geben. Bei einem Ausgangsvektor $(0.5, 0.4)$ ist die Entscheidung für Klasse 0 unsicherer als bei einem Ausgangsvektor $(0.95, 0.1)$. Im Sinne eines statistischen Klassifikationsvorgangs hätten wir am liebsten eine Ausgabe, die uns direkt die klassenspezifischen Rückschlußwahrscheinlichkeiten mitteilt. Im allgemeinen ist bei analogen neuronalen Netzen der Ausgangsvektor kein Wahrscheinlichkeitsvektor. Es reicht nicht aus, ihn einfach auf die Komponentensumme Eins zu normieren. Unter gewissen Bedingungen kann man aber ein analoges neuronales Netz so trainieren, daß es einem die klassenspezifischen Rückschlußwahrscheinlichkeiten liefert.

RAM-basierte neuronale Netze geben einem normalerweise eine binäre Entscheidung für oder wider eine Klasse. Möchte man mehr als zwei Klassen unterscheiden, kombiniert man mehrere Netze so, daß jedes Teilnetz eine Klasse gegen alle anderen Klassen trennt. Da das Netz binäre Entscheidungen liefert, gibt es einem keine Auskunft darüber, wie *sicher* eine Entscheidung für eine Klasse ist. Natürlich möchte man aber von einem Klassifikator wissen, ob er sich seiner Sache sicher war, oder ob er nur „geraten“ hat.

Um bei RAM-basierten neuronalen Netzen dieses Problem zu umgehen, sind verschiedene Ansätze versucht worden. Beim Probabilistic RAM von Gorse und Taylor (1989) speichert der Knoten Wahrscheinlichkeiten, mit denen die Emission einer Eins oder Null am Knotenausgang bestimmt wird. Damit lassen sich die Rückschlußwahrscheinlichkeiten bis zum Ausgang des Netzes „herausführen“. Diese sind dort aber nicht direkt ablesbar, sondern ergeben sich aus den Mittelwerten der aus dem Netz kommenden Bitströme. Man muß also das Netz für jedes Muster

eine genügend große Zahl von Schritten laufen lassen, um aus der Zahl der beobachteten Nullen und Einsen die Mittelwerte und damit die Rückschlußwahrscheinlichkeit berechnen zu können.

Ein ähnlichen Weg geht das *Goal-Seeking Neuron (GSN)* (Filho et al., 1992), daß neben der Null und Eins auch noch das spezielle Symbol 'u' emittieren kann. Wird ein 'u' emittiert, so adressiert der folgende Knoten auf dieser Leitung nicht mit einer Null *oder* Eins, sondern mit beiden Möglichkeiten. Da nun ein Knoten nicht nur mit *einer* Adresse sondern mit einer Menge von Adressen adressiert wird, kann ein solcher Prozess nicht mehr so einfach in Hardware implementiert werden, wie der ursprüngliche RAM-basierte Knoten.

Es gibt auch Ansätze, die es erlauben, daß ein Neuron nicht nur zwei oder drei sondern k verschiedene Symbole auf einer Leitung emittieren kann. Howells et al. (1998) lassen ein Neuron ihres *Generalised Convergent Networks (GCN)*, soviele sogenannte *Basis-Symbole* emittieren, wie es Klassen gibt. Zusätzlich gibt es noch *zusammengesetzte Symbole*, und zwar für jede Kombination von Klassen eine. Damit ist die Zahl der zusammengesetzten Symbole gerade die Kardinalität der Potenzmenge der Menge der Klassen. Bei vier Klassen hätte man also schon $2^4 = 16$ verschiedene zusammengesetzte Symbole. Gurney (1992b,a, 1993) verwendet auch mehrere Symbole pro Adresse eines Neurons. Diese Symbole verlassen aber nicht direkt den Knoten, sondern werden ähnlich wie bei Gorse und Taylor in einen Bitstrom umgewandelt. Dieser Bitstrom wird dann nochmals durch eine nichtlineare Aktivierungfunktion in einen weiteren Bitstrom umgewandelt, der den Knoten dann verläßt.

Wie sieht die Situation bei einem Hyperpermutationsnetzwerk aus? Bei einem HPN sind am Ausgang des Netzwerks zunächst einmal genausoviele Bits vorhanden wie am Eingang, woher ja unter anderem die Reversibilität der Transformation kommt. Erst durch die Einschränkung des Blickfeldes auf wenige Ausgangsbits wird die Datenreduktion erreicht. Da wir in der Wahl der Fokusbits frei sind, können wir bestimmen, in wie viele Ausgangszustände die Eingangsmuster aufgeteilt werden: Bei l Fokusbits gibt es gerade $n = 2^l$ Ausgangszustände. Haben wir nun ein k-Klassen-Problem mit $k < n$, so gibt es mehrere Zustände, die die Zugehörigkeit eines Eingangsmusters zu dergleichen Klasse anzeigen. Wie wir noch sehen werden, lassen sich bei einem HPN die klassenspezifischen Rückschlußwahrscheinlichkeiten sehr einfach gewinnen: Wir prozessieren alle Trainingsmuster einmal und zählen die klassenspezifischen Häufigkeiten der Ausgangszustände. Diese klassenspezifischen Häufigkeiten sind aber gerade erwartungtreue Schätzungen der klassenspezifischen Rückschlußwahrscheinlichkeiten. Haben wir nun z. B. pro Klasse mehrere Ausgangszustände, so wird es „sichere" Zustände geben, in denen die Rückschlußwahrscheinlichkeit für eine Klasse stark dominiert, und Zustände, in denen die Rückschlußwahrscheinlichkeit für eine Klasse weniger stark dominiert. Daher läßt sich auch einfach eine Mindestrückschlußwahrscheinlichkeit festlegen, unterhalb der das HPN ein Eingangsmuster zurückweist, weil die Erkennungsicherheit für das Eingangsmuster nicht hoch genug ist. Im Training wird versucht werden, möglichst viele „reine" Zustände zu erzeugen, d. h. Zustände, die für eine Klasse eine Rückschlußwahrscheinlichkeit nahe Eins und für alle anderen Klassen eine Rückschlußwahrscheinlichkeit nahe Null aufweist.

4.5.4 Funktionen-Netze

Die bis jetzt vorgestellten Netzwerke gehören alle zur allgemeineren Klasse der Funktionen-Netze (Rojas, 1996). Es gibt noch eine weitere Architektur, die wie ein HPN zwar kein neuronales Netz, aber ein Funktionen-Netz ist. Hierbei handelt es sich um die Funktionen-Netze, die Toffoli (1980a,b); Fredkin und Toffoli (1982) in ihren Arbeiten vorgestellt haben. Sie wurden bis jetzt nicht erwähnt, weil sie in der Literatur noch nicht zur Mustererkennung eingesetzt wurden.

Toffoli hat seine Überlegungen unter dem Stichwort *Reversible Computing* zusammengefaßt. Bei der Konstruktion seiner reversible-computing Netze kam es ihm darauf an, daß das Netzwerk eine Berechnung *reversibel* durchführt, und zwar unter anderem aus den folgenden zwei Gründen:

> In fact, one of the strongest motivations for the study of reversible computing comes from the desire to reduce heat dissipation in compution machinery, and thus achieve higher density and speed.
>
> The main reason why the concept of reversibility is so pervasive is the following. In every reversible system there are a number of quantities (functions of the system's state) which are conserved, i.e. which are constant along each of the system's trajectories (cf. the „integrals of motion" of theoretical mechanics).

In ihrer Architektur sind die reversible-computing Netze den Hyperpermutationsnetzwerken sehr ähnlich: Die von Toffoli verwendeten Netzwerkknoten haben auch genauso viele Ein- wie Ausgänge und die Knoten beinhalten eine Lookup-Tabelle. Es gibt auch das Konzept des *Blindbits*, das bei Toffoli nur einen anderen Namen trägt. Ein bekannter Knotentyp ist der sogenannte *Fredkin-Schalter*, den Abbildung 4.14 zeigt. Der Fredkin-Schalter hat die Eigenschaft, daß er seine eigene Inverse ist.

Abbildung 4.14: Der Fredkin-Schalter. Er besteht aus einem Knoten mit drei Ein- und Ausgangsbits. Der Wert des obersten Eingangsbits wird immer auf das oberste Ausgangsbit kopiert. Hat das oberste Eingangsbit den Wert 0 (linkes Bild), werden die beiden unteren Eingangsbits auf ihre entsprechenden Ausgangsbits kopiert. Hat das oberste Eingangsbit dagegen den Wert 1 (rechtes Bild), werden die beiden unteren Eingangsbits vor dem Kopieren auf die beiden unteren Ausgangsbits vertauscht.

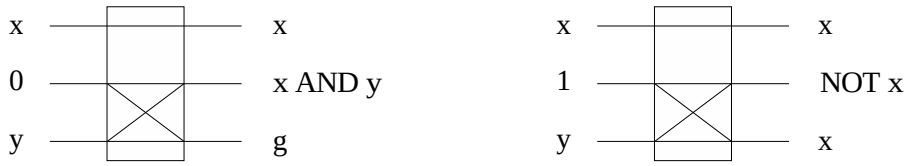


Abbildung 4.15: Der Fredkin-Schalter als logische Funktion. Legt man beim Fredkin-Schalter das mittlere Eingangsbit konstant auf null, hat das mittlere Ausgangsbit den Wert $x \text{AND} y$, und das obere Eingangsbit wird auf das obere Ausgangsbits kopiert. Legt man dagegen das mittlere Eingangsbit konstant auf eins, hat das mittlere Ausgangsbit hat den Wert $\text{AND} x$, und das obere Eingangsbit wird auf die beiden anderen Ausgangsbits kopiert.

Die Architektur von HPNs und reversible-computing Netzen ist zwar praktisch gleich, der „Betrieb" der Architektur sieht allerdings anders aus. Das eigentliche Ziel von Toffoli (1980b); Fredkin und Toffoli (1982) ist nämlich die sog. *Conservative Logic*, die sich dadurch auszeichnet, daß zu jedem Zeitpunkt eine konstante Zahl von Nullen und Einsen im System (Netzwerk) ist.

Diese Forderung ist aus physikalischer Sicht wie eine Art Teilchenzahlerhaltung zu verstehen. Fredkin und Toffoli benutzt die Forderung der konstanten Zahl von Nullen und Einsen, um darauf weitere Berechnungsmodelle, z. B. das *Billiardkugelmodell*, aufzubauen. Mit Hilfe des Billiardkugelmodells zeigen Fredkin und Toffoli dann, daß man mit einem perfekten klassischen Gas in einem speziell geformten Behälter logische Berechnungen durchführen kann.

Die Motivation zur Entwicklung der reversible-computing Netze war also eine ganz andere als bei der Entwicklung der HPNs. HPNs kennen die Einschränkung der konstanten Zahl von Nullen und Einsen nicht, da sie eine ganz andere Aufgabe als die reversible-computing Netze, nämlich die Mustererkennung, lösen sollen. In der Literatur sind keine Aussagen darüber zu finden, ob mit reversible-computing Netzen auch schon Mustererkennungsaufgaben angegangen worden sind. Ebenso wurde noch nicht versucht, reversible-computing Netze durch Lernen mit Beispielen auf bestimmte Problemlösungen zu trainieren.

5 Der Informationsbegriff

Um ein statistisches Mustererkennungssystem trainieren zu können, müssen wir die Performanz des Systems quantifizieren. Den hauptsächlich verwendeten Formalismus stellen wir hier vor.

5.1 Formalisierung der Begriffe 'Umwelt', 'Sensor' und 'Wahrnehmungs-System'

Wir werden die im Kapitel „Wahrnehmung“ informell eingeführten Begriffe 'Umwelt', 'Sensor' und 'Wahrnehmungs-System' formalisieren.

5.1.1 Umwelt

Die *Umwelt* ist die Umgebung, in die das *Mustererkennungssystem* eingebettet ist. Mit Hilfe seiner *Sensoren* nimmt das System Daten aus der Umgebung auf, die es nutzt, um Wissen über seine Umwelt zu erlangen. Hat das System auch Aktuatoren, kann es auf die Umwelt aktiv einwirken, so daß ein Regelkreis entsteht. Die Umwelt ist nun als Menge von Ereignissen anzusehen. In dieser Umwelt sind von vornherein keine Ereignisse ausgeschlossen, wir könnten also die Menge aller *möglichen* Ereignisse in der Umwelt gar nicht angeben.

Stellen wir uns als Beispiel vor, wir wollten anhand von Sprachsignalen verschiedene Sprecher identifizieren, und die Umwelt wäre ein Zimmer mit mehreren Personen. In diesem Zimmer würde es neben Schallsignalen auch noch Lichtsignale und andere physikalische Ereignisse geben. Die Menge aller dieser Ereignisse nennen wir im folgenden den Ereignisraum Ω.

$$\Omega = \{\omega \mid \text{„}\omega \text{ ist ein Ereignis aus der Umwelt“}\}$$

Ω enthält also alle Ereignisse der Umwelt. Von diesem Ω interessiert uns meistens nur eine Teilmenge $\Omega' \subseteq \Omega$. In dem eben genannten Beispiel wäre dies die Teilmenge der akustischen Signale.

5.1.2 Sensor

Der *Sensor* ist das Hilfsmittel, das dem *System* den Kontakt zur Außenwelt ermöglicht. Jeder Sensor ist nur für eine Teilmenge $\Omega'' \subseteq \Omega$ sensitiv, nämlich gerade für die Ereignisse ω'', für die er physikalisch ausgelegt ist. Im Idealfall ist $\Omega' = \Omega''$, denn dann ist der Sensor gerade für die Teilmenge aus Ω sensitiv, die uns auch interessiert.

Damit sich das System ein Bild seiner Umgebung machen kann, braucht es eine interne Repräsentation der Ereignisse in der Umwelt. Der Sensor hat nun die Aufgabe, ein Ereignis aus der Umwelt auf ein systeminternes Symbol abzubilden. Unser später verwendetes System, der Digitalcomputer, wird nur in der Lage sein, Bitstrings zu verarbeiten. Daher wählen wir als systeminternes Symbolalphabet Bitstrings b. Man kann sich zwar prinzipiell auch andere Symbolalphabete vorstellen, aber die einzigen Symbole, die ein Digitalcomputer wirklich „versteht“, sind Bitstrings. Die Bitstrings b kommen aus der Menge $\mathbb{B}^n$, wobei n noch unbestimmt ist.

Wir führen nun einen *formalen Sensor* S ein, der die geforderte Abbildung durchführt:

$$S: \quad \begin{array}{l} \Omega \to \mathbb{B}^n \\ \omega \mapsto S(\omega) = b \end{array}$$

In dem Sprecherbeispiel wäre der physikalische Sensor ein Mikrofon, das die Schallwellen in Strom- oder Spannungsschwankungen umwandelt. Das Mikrofon allein erfüllt aber noch nicht die Anforderungen, die wir an einen formalen Sensor stellen, da es die Umweltereignisse nicht auf Elemente der internen Repräsentation des Systems abbildet. Schalten wir aber hinter das Mikrofon einen Analog-Digital-Wandler (A-D-Wandler), dann erfüllt diese Einheit genau die Anforderungen an einen formalen Sensor. Sie bildet nämlich Umweltereignisse (Schallwellen) $\omega \in \Omega$ auf Bitstrings ab, die mit der Taktrate des A-D-Wandlers vom Sensor ins System übergehen. Diese Bitstrings sind nun genau die Symbole $b \in \mathbb{B}^n$, die der formale Sensor liefern soll und die unser System verarbeiten kann. Man kann die Bitstrings auch als Zahlen interpretieren, was allerdings oft nicht nötig bzw. nicht möglich ist.

Über den *formalen Sensor* können wir noch weitere Aussagen treffen:

1. Die durch den formalen Sensor realisierte Abbildung $S : \Omega \to \mathbb{B}^n$ ist im allgemeinen nicht surjektiv. Selbst, wenn man jedes Umweltereignis $\omega \in \Omega$ mit S nach $\mathbb{B}^n$ abbildet, deckt man auf diese Weise nicht unbedingt die gesamte Menge $\mathbb{B}^n$ ab. Bis auf Ausnahmen gilt also:

$$S(\Omega) \neq \mathbb{B}^n$$

2. Die durch den formalen Sensor realisierte Abbildung $S : \Omega \to \mathbb{B}^n$ ist im allgemeinen nicht injektiv. Je nach Art des Sensors werden mehr oder weniger Ereignisse ω durch den Sensor auf dasselbe Symbol $b \in \mathbb{B}^n$ abgebildet, es können auch unendlich viele ω auf dasselbe b abgebildet werden. Ein Beispiel für eine solche Abbildung wäre ein Sensor, der Töne oberhalb einer bestimmten Tonhöhe auf ein Symbol b_1 und Töne unterhalb einer bestimmten Tonhöhe auf ein anderes Symbol $b_2 \neq b_1$ abbildet. Daher gilt:[1]

$$\exists\, \omega_1, \omega_2 \in \Omega: \quad \omega_1 \neq \omega_2 \ \wedge\ S(\omega_1) = S(\omega_2)$$

5.1.3 Wahrnehmungs-System

Nachdem nun die Umwelt und der Sensor definiert sind, läßt sich das *Wahrnehmungs-System* relativ einfach spezifizieren: Das System nimmt einen Strom von Daten des formalen Sensors auf und benutzt diese Eingabe, um eine vorher festgelegte Aufgabe zu bearbeiten. Der Datenstrom kommt dadurch zustande, daß der Sensor zu bestimmten Zeitpunkten eine Messung durchführt und der D-A-Wandler dann einen Zahlenwert liefert. Das System ist also in die Zeit eingebettet. Es beginnt dort, wo die digitalen Daten hinter dem formalen Sensor anfallen und endet vor einem möglichen D-A-Wandler eines Aktuators (siehe Abbildung 2.1). In diesem Sinne ist das System also ein digitaler Datentransformator. Meistens wird es sich bei der Aufgabe um eine allgemeine Erkennungsaufgabe handeln. Dabei ist wichtig, daß das System durch Training mit Beispielen lernen soll, die Aufgabe zu lösen. Zu dem in Abbildung 2.1 auf Seite 8 angedeuteten Regelkreis kommt man, wenn das System neben der Sensorik auch eine Aktuatorik hat. Es bekommt mit der Taktrate des A-D–Wandlers Daten geliefert und kann über einen D-A–Wandler einen entsprechenden Aktuator steuern.

[1] Das Zeichen $\exists$ bedeutet „Es existiert ein“, also hier „Es existiert ein $\omega_1, \omega_2 \in \Omega$ ...“

5.2 Diversität, Information und Redundanz

Um die Güte des Wahrnehmungs-Systems bewerten zu können, brauchen wir Größen, die die Güte quantifizieren. Nun ist, wie wir später sehen werden, die Fehlerrate eines Mustererkennungssystems nicht immer eine gute Kostenfunktion. Oft werden auch nur Annäherungen an die Fehlerrate verwendet, wie z. B. bei neuronalen Netzen, bei denen man den mittleren quadratischen Fehler als Kostenfunktion verwendet. Ein digitales Mustererkennungssystem operiert auf Bitstrings $b \in \mathbb{B}^n$. Hier kommt das Konzept des *Unterschieds* ins Spiel. Bei einer Klassifikationsaufgabe soll das System die Eingangssymbole b aus verschiedenen Klassen *unterscheiden* können. Das System soll die Eingangsdaten so transformieren, *daß die für die Aufgabe benötigten Unterschiede erhalten bleiben* (siehe auch Oberländer (1999)). Wir erhalten die Eingangssymbole b, indem der Sensor zu bestimmen Zeiten $t_0, t_1, \ldots$ Ereignisse $\omega_i \in \Omega$ aus der Umwelt in die interne Repräsentation des Systems, also in Bitstrings $b \in \mathbb{B}^n$ abbildet. Damit wird eine Folge von Ereignissen in eine Folge von Bitstring abgebildet: $\boldsymbol{\omega} = (\omega_0, \omega_1, \ldots) \rightarrow \boldsymbol{b} = (b_0, b_1, \ldots)$ mit $b_i = S(\omega_i)$.

Am besten kann man sich die benötigten Begriffe klarmachen, wenn man die theoretischen Definitionen auf eine konkrete Menge von Symbolen anwendet. Die Begriffe *Diversität*, *Redundanz* und *Information* sind auf Paaren von Bitstrings, also auf den Elementen der Menge $\mathbb{B}^n \times \mathbb{B}^n$ definiert. Nehmen wir nun an, es sei die Ereignisfolge $\boldsymbol{\omega}$ = ('a','k','U','k','f') aufgetreten, die der Sensor gemäß der Zuordnung

$$\begin{array}{rcl} \text{'a'} & \mapsto & 00 \\ \text{'k'} & \mapsto & 01 \\ \text{'U'} & \mapsto & 10 \\ \text{'f'} & \mapsto & 11 \end{array}$$

in die Folge von Bitstrings $\boldsymbol{b} = (00, 01, 10, 01, 11)$ abgebildet hätte. Da wir gleich mit Mengen von Paaren operieren werden, benötigen wir jetzt den Begriff der *Multimenge*: Eine Multimenge ist eine Menge, in der Elemente mehrmals vorkommen dürfen. Dieser Begriff wurde zuerst von D. A. Knuth eingeführt. Die Multimenge der Ereignisse ist $\mathcal{M} := \{\omega \,|\, \omega \in \boldsymbol{\omega}\}$ = {'a','k','U','k','f'} und die Multimenge der internen Bitstrings $\mathcal{B} = \{b \,|\, b \in \boldsymbol{b}\} = \{00, 01, 10, 01, 11\}$. Die Multimenge der Paare einer Multimenge sind nun einfach $\mathcal{M}^2 := \mathcal{M} \times \mathcal{M}$ bzw. $\mathcal{B}^2 := \mathcal{B} \times \mathcal{B}$.

5.2.1 Grundbegriffe aus der Statistik

Da wir tiefer in die Welt der Statistik einsteigen werden, definieren wir nun die später benötigten Grundbegriffe aus der Statistik, so wie sie in Heise und Quattrocchi (1995) eingeführt werden.

Definition 19.

Stichprobenraum: Sei $\mathcal{Q} = \{s_0, s_1, \ldots, s_{n-1}\}$ eine Menge mit n *Elementarereignissen*. Den Elementarereignissen $s_i \in \mathcal{Q}$ sei jeweils eine reelle Zahl $p_i := p(s_i) \in [0, 1]$ mit $\sum_{i=0}^{n-1} p_i = 1$ zugeordnet, die die *Wahrscheinlichkeit* des Elementarereignisses s_i genannt wird. Den Vektor $\boldsymbol{p} = (p_0, p_1, \ldots, p_{n-1}) \in [0, 1]^n$ nennen wir *Wahrscheinlichkeitsverteilung* von $\mathcal{Q}$. Die Wahrscheinlichkeitsverteilung $p_i(\mathcal{Q}) = 1/n, i = 0, \ldots, n-1$ heißt *Gleichverteilung*. Das Paar $(\mathcal{Q}, \boldsymbol{p})$ heißt *endlicher Stichprobenraum*. Das *Wahrscheinlichkeitsmaß*

$$\begin{array}{rl} & \mathcal{P}(\mathcal{Q}) \rightarrow \mathbb{R} \\ \mathrm{P}: & m \mapsto \mathrm{P}(m) := \displaystyle\sum_{s \in m} p(s) \end{array}$$

ordnet jedem Ereignis $m \subseteq \mathcal{Q}$, also jedem Element m aus der Potenzmenge $\mathcal{P}(\mathcal{Q})$ von $\mathcal{Q}$, seine Wahrscheinlichkeit $\mathrm{P}(m)$ zu.

Verbundraum: Es seien $\mathcal{A}$ und $\mathcal{B}$ zwei nichtleere Mengen. Jedem Paar $ab \in \mathcal{A} \times \mathcal{B}$ von Elementarereignissen $a \in \mathcal{A}$ und $b \in \mathcal{B}$ sei eine Verbundwahrscheinlichkeit $p(ab) \in [0,1]$ mit $\sum_{a\in\mathcal{A},b\in\mathcal{B}} p(ab) = 1$ zugeordnet. Dann bildet das kartesische Produkt $\mathcal{AB}$ aller als Elementarereignisse betrachteten Paare $ab \in \mathcal{A} \times \mathcal{B}$ einen Stichprobenraum $(\mathcal{AB}, \boldsymbol{q})$, den *Verbundraum*. Auf den Mengen $\mathcal{A}$ und $\mathcal{B}$ wird mittels $p(a) := \sum_{b\in\mathcal{B}} p(ab)$ und $p(b) := \sum_{a\in\mathcal{A}} p(ab)$ jeweils eine Wahrscheinlichkeitsverteilung definiert.

Bedingte Wahrscheinlichkeit: Es sei $\mathcal{AB}$ ein Verbundraum und $a \in \mathcal{A}$ ein Elementarereignis mit $p(a) \neq 0$. Für jedes $b \in \mathcal{B}$ definieren wir seine *bedingte Wahrscheinlichkeit* als $p(b|a) := \frac{p(ab)}{p(a)}$ und für jedes $a \in \mathcal{A}$ als $p(a|b) := \frac{p(ab)}{p(b)}$. Nach der *Bayes'schen Formel* gilt für die bedingten Wahrscheinlichkeiten

$$p(a|b) = \frac{p(a)}{p(b)}\, p(b|a)$$

Zufallsgrößen: Es sei $(Q, \boldsymbol{p})$ ein endlicher Stichprobenraum. Eine Abbildung $v : \mathcal{Q} \to \mathbb{R}$ die jedem Elementarereignis $s \in \mathcal{Q}$ eine reelle Zahl $v(s)$ zuordnet, heißt *Zufallsgröße* oder *Zufallsvariable*. Der Erwartungswert von v ist

$$E(v) := \sum_{s\in\mathcal{Q}} p(s)\, v(s)$$

und ihre Varianz ist

$$V(v) := \sum_{s\in\mathcal{Q}} p(s)\ (v(s) - E(v))^2 = E\left((v - E(v))^2\right)\ .$$

Für den Erwartungswert schreiben wir auch $\langle x \rangle := E(x)$.

Quellen: Eine n-äre Quelle hat einen Nachrichtenvorrat $\mathcal{Q} = \{s_0, s_1, \ldots, s_{n-1}\}$, aus dem zu bestimmten Zeitpunkten $t_0, t_1, \ldots$ eine Nachricht s_i ausgewählt und emittiert wird. Eine solche Quelle ist die informationstheoretische Interpretation eines Stichprobenraums $(\mathcal{Q}, \boldsymbol{p})$. Zu jedem Zeitpunkt wird also aus dem Nachrichtenvorrat $\mathcal{Q}$ mit der Wahrscheinlichkeitsverteilung $\boldsymbol{p}$ eine Nachricht $s_i \in \mathcal{Q}$ ausgewählt.

Einige der Definitionen verdeutlichen wir uns in

Beispiel 9. Der endliche Stichprobenraum eines (fairen) Würfels ist

$$(\mathcal{Q}, \boldsymbol{p}) = \left(\{1,2,3,4,5,6\}, \left(\frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}\right)\right)\ .$$

Die Wahrscheinlichkeit für das Ereignis $m = \{1,2\}$ ist $\mathrm{P}(m) = 1/3$. Der Verbundraum zweier Würfel ist $(\mathcal{QQ}, \boldsymbol{p}')$. Die Abbildung $u : \mathcal{QQ} \to \mathbb{R}$, die jedem Element $(s^{(0)}, s^{(1)}) \in \mathcal{QQ}$ die Summe $u(s^{(0)}, s^{(1)}) = s^{(0)} + s^{(1)}$ zuordnet, ist eine Zufallsvariable. Wir können einen Würfel als Quelle betrachten, der die Symbole aus $\mathcal{Q}$ emittiert.

5.2.2 Diversität

Der Begriff der Diversität tauchte schon früh in der Literatur auf (Simpson, 1949), und zwar um die Verschiedenheit von Tierpopulationen zu quantifizieren. In der Physik steht für die Verschiedenheit einer (Teilchen-)Population u. a. die *Entropie*. Der Zusammenhang zwischen der im folgenden definierten Diversität und der Entropie wurde auch untersucht. Da es sich dabei aber nicht um ein zentralen Punkt dieser Arbeit handelt, ist die Untersuchung im Anhang A.2 auf Seite 175 zu finden.

Die *Diversität* gibt die *Verschiedenheit* der Elemente einer Multimenge an. Sie kommt in drei „Ausprägungen" vor, und zwar als *totale*, *kalte* und *heiße Diversität*.

Totale Diversität D_t**:** Die *totale Diversität* ist die gesamte Anzahl von Paaren $(b_k, b_l) \in \mathcal{B}^2 := \mathcal{B} \times \mathcal{B}$. [2] Weil $\mathcal{B}$ und $\mathcal{M}$ gleich viele Elemente haben (zu jedem Ereignis aus $\mathcal{M}$ gibt es eine Repräsentation aus $\mathcal{B}$), gibt die *totale Diversität* damit die prinzipiell mögliche Zahl paarweiser Unterschiede der Multimenge $\mathcal{M}$ an. Da der *formale Sensor* S unterschiedliche Elemente aus $\mathcal{M}$ auf gleiche Elemente aus $\mathcal{B}$ abbilden kann, sind nach dem Sensor S Unterschiede aus $\mathcal{M}$ in $\mathcal{B}$ möglicherweise nicht mehr sichtbar. Deshalb nennen wir Paare aus $\mathcal{B}$ synonym auch Unterschiede. Bei N Elementen ist $D_\mathrm{t} = N^2$.

Kalte Diversität D_k**:** Die *kalte Diversität* ist die Anzahl von Paaren (b_k, b_l), für die $b_k = b_l$ ist. Sie können sich zwar in $\mathcal{M}$ durchaus unterscheiden, sind aber innerhalb ihrer Repräsentation in $\mathcal{B}$ nicht mehr unterscheidbar und in diesem Sinne nicht mehr wirksam, daher die Bezeichnung *kalte* Diversität.

Heiße Diversität D_h**:** Die *heiße Diversität* ist die Anzahl von Paaren (b_k, b_l), für die $b_k \neq b_l$ ist. Hier werden die Unterschiede gezählt, die innerhalb des Systems wirklich sichtbar und damit wirksam sind, daher die Bezeichnung *heiße* Diversität.

Formal erhalten wir [3]

Definition 20.

$$
\begin{aligned}
D_\mathrm{t} &= |\mathcal{B}^2| \\
D_\mathrm{k} &= |\{(b_k, b_l) \in \mathcal{B}^2 \mid b_k = b_l\}| \\
D_\mathrm{h} &= |\{(b_k, b_l) \in \mathcal{B}^2 \mid b_k \neq b_l\}|
\end{aligned}
$$

Aus Definition (20) ergibt sich

Folgerung 5.

$$\boxed{D_\mathrm{t} = D_\mathrm{h} + D_\mathrm{k}}$$

In unserem Beispiel ist $|\mathcal{B}| = 5$ und $|\mathcal{B}^2| = 25$, d. h. es gibt 25 Paare aus $\mathcal{B}$ und $D_\mathrm{t} = 25$. Bildet man nun alle Paare (00, 00), (00, 01), (00, 10), ..., (10, 11), (11, 11) und zählt diejenigen, für die $b_k = b_l$ gilt, so erhalten wir die *kalte Diversität* $D_\mathrm{k} = 7$. Daraus ergibt sich $D_\mathrm{h} = D_\mathrm{t} - D_\mathrm{k} = 25 - 7 = 18$. Die *Diversitäten* geben also die „Verschiedenheit" einer Multimenge an, wobei die *heiße Diversität* entscheidend ist, da nur die von ihr gezählten paarweisen Unterschiede auch im System wirksam sind. Das Minimum der *heißen Diversität* ist null, wenn alle Elemente der Multimenge gleich sind und ihr Maximum ist $N(N-1)$, wenn alle N Elemente der Multimenge verschieden sind.

[2] Die Paare (b_k, b_l) sind im folgenden immer aus der Multimenge $\mathcal{B}^2$

[3] $|\mathcal{M}|$ bezeichnet die Anzahl von Elementen einer (Multi-)Menge $\mathcal{M}$

Eine detailliertere Einsicht in die Struktur einer Multimenge von Paaren bietet die Diversitätsmatrix $\boldsymbol{D}$. Sie gibt an, wie häufig ein Paar (b_k, b_l) vorkommt, und zwar wird die Häufigkeit jedes Paares (b_k, b_l) im Matrixelement $[\boldsymbol{D}]_{\mathrm{ind}(b_k),\mathrm{ind}(b_l)}$ eingetragen. Mit $\mathrm{ind}(b)$, der Indexfunktion aus Definition 8 auf Seite 35 und den i-ten Einheitsvektoren $\boldsymbol{\epsilon}_i$ erhalten wir:

$$\boldsymbol{D} = \sum_{b_1,b_2\in\mathcal{B}} \boldsymbol{\epsilon}_{\mathrm{ind}(b_1)} \cdot \boldsymbol{\epsilon}^{\mathrm{T}}_{\mathrm{ind}(b_2)} \tag{5.1}$$

Da jedes Paar (b_k, b_l) auch als Paar (b_l, b_k) auftritt, ist die Diversitätsmatrix symmetrisch. In Definition 18 hatten wir bereits den Gewichtsvektor einer Zustandsfolge eines Leitungsbündels definiert. Der Gewichtsvektor $\boldsymbol{w}(\mathcal{M})$ einer Multimenge $\mathcal{M}$ gibt ganz analog an, wie oft welches Element in einer Multimenge vorkommt. Mit dem Gewichtsvektor $\boldsymbol{w}$ der Dimension d läßt sich die Diversitätsmatrix über $\boldsymbol{w} \cdot \boldsymbol{w}^{\mathrm{T}}$ berechnen:

$$\boldsymbol{D} = \sum_{b_1,b_2\in\mathcal{B}} \boldsymbol{\epsilon}_{\mathrm{ind}(b_1)} \cdot \boldsymbol{\epsilon}^{\mathrm{T}}_{\mathrm{ind}(b_2)} = \sum_{i=0,j=0}^{d-1} w_i w_j\, \boldsymbol{\epsilon}_i \cdot \boldsymbol{\epsilon}_j^{\mathrm{T}} = \left(\sum_{i=0}^{d-1} w_i \boldsymbol{\epsilon}_i\right)\left(\sum_{j=0}^{d-1} w_j \boldsymbol{\epsilon}_j^{\mathrm{T}}\right) = \boldsymbol{w}(\mathcal{B}) \cdot \boldsymbol{w}^{\mathrm{T}}(\mathcal{B})$$

In unserem Beispiel ist der Gewichtsvektor $\boldsymbol{w}(\mathcal{B})$ der Multimenge $\mathcal{B}$

$$\boldsymbol{w}(\mathcal{B}) = \boldsymbol{w}(\{00, 01, 10, 01, 11\}) = \begin{pmatrix} 1 \\ 2 \\ 1 \\ 1 \end{pmatrix} \tag{5.2}$$

und die Diversitätsmatrix ist

$$\boldsymbol{D} = \boldsymbol{w}(\mathcal{B}) \cdot \boldsymbol{w}^{\mathrm{T}}(\mathcal{B}) = \begin{pmatrix} 1 \\ 2 \\ 1 \\ 1 \end{pmatrix} \cdot (1, 2, 1, 1) = \begin{pmatrix} 1 & 2 & 1 & 1 \\ 2 & 4 & 2 & 2 \\ 1 & 2 & 1 & 1 \\ 1 & 2 & 1 & 1 \end{pmatrix} .$$

Die Anzahl der Paare mit $b_k = b_l$ ist gleich der Summe der Hauptdiagonalelemente von $\boldsymbol{D}$ und die Anzahl der Paare mit $b_k \neq b_l$ gleich der Summe der Nicht-Hauptdiagonalelemente von $\boldsymbol{D}$. So kommt man für eine N-elementige Multimenge mit Elementen aus $\mathbb{B}^n$ zu den Beziehungen:[4]

$$\boxed{\begin{aligned} &D_{\mathrm{t}} = N^2 && \left(= \sum_{i=0,j=0}^{2^n-1} [\boldsymbol{D}]_{ij}\right) \\ &D_{\mathrm{k}} = \mathrm{Sp}(\boldsymbol{D}) = \boldsymbol{w}^{\mathrm{T}} \cdot \boldsymbol{w} && \\ &D_{\mathrm{h}} = D_{\mathrm{t}} - D_{\mathrm{k}} = N^2 - \mathrm{Sp}(\boldsymbol{D}) && \left(= \sum_{\substack{i=0,j=0 \\ i\neq j}}^{2^n-1} [\boldsymbol{D}]_{ij}\right) \end{aligned}} \tag{5.3}$$

[4] $\mathrm{Sp}(\boldsymbol{M})$ bezeichnet die Spur einer Matrix $\boldsymbol{M}$. Die Spur einer Matrix ist die Summe der Hauptdiagonalelemente: $\mathrm{Sp}(\boldsymbol{M}) = \sum_i m_{ii}$

5.2.2.1 Warum kein Abstandsmaß?

Man kann sich fragen, warum wir auf den Elementen $b \in \mathbb{B}^n$ kein Abstandsmaß $a(x, y)$ definieren. Es gäbe verschiedene Möglichkeiten:

1. Die Hamming-Distanz. Hier definiert man den Abstand zweier Bitstrings über die Anzahl der Bits, in denen sich zwei Bitstrings unterscheiden. Je mehr unterschiedliche Bits die beiden Bitstrings haben, desto größer ist ihr Abstand. Damit wären also 0111 und 1000 „weiter entfernt“ als 1010 und 0010.

2. Der Euklid'sche Abstand mit Dualcode. Hier würden wir jedem Bitstring mittels der Indexfunktion $\text{ind}(b)$ aus Definition 8 auf Seite 35 eine natürliche Zahl zuordnen und als Abstand die Differenz der beiden natürlichen Zahlen verwenden. Damit wären also 0111 (= 7) und 1000 (= 8) im Gegensatz zur Hamming-Distanz „näher zusammen“ als 1010 (= 10) und 0010 (= 2).

3. Der Euklid'sche Abstand mit Gray-Code. Man könnte die Bitstrings auch im Gray-Code $\text{G}(b)$ interpretieren und dann den Euklid'schen Abstand auf die Zahlen $\text{G}(b)$ anwenden. Dies gäbe wieder ein neues Abstandsmaß.

Jede Wahl eines Abstandsmaßes bedeutet gleichzeitig eine Interpretation bzw. Deutung der Daten in einer gewissen „Richtung“. Wie wir sehen, ist es prinzipiell nicht möglich, auf Bitstrings einen Abstand zu definieren, ohne dabei eine Bevorzugung in der einen oder anderen Richtung zu verursachen. Ein Problem ist, daß die Repräsentation in Bezug auf eine bestimmte Aufgabe u. U. gar kein vernünftiges Abstandsmaß zuläßt.

Nehmen wir als Beispiel die Multimenge $\mathcal{M}$. Die in ihr enthaltenen Buchstaben werden im ASCII-Code durch die in Tabelle 5.1 gezeigten Bitstrings repräsentiert. Mit der eben erwähnten zweiten Möglichkeit wäre also der Abstand $a(\text{'a'}, \text{'f'}) = |97 - 102| = 5$ und der Abstand $a(\text{'a'}, \text{'U'}) = |97 - 85| = 12$. Angenommen, wir möchten die Buchstaben nach Vokalen und Konsonanten zusammenfassen. Dabei sieht man sofort, daß für diese Aufgabe das Abstandsmaß ungeeignet ist, da 'a' und 'U' (beides Vokale) „weiter auseinander“ sind als 'a' und 'k'.

$m \in \mathcal{M}$	Bitstring	Dezimalzahl
'a'	01100001	97
'k'	01101011	107
'U'	01010101	85
'f'	01100110	102

Tabelle 5.1: Repräsentation der Elemente der Multimenge $\mathcal{M}$

Diese Schwierigkeiten einer unerwünschten Bevorzugung vermeiden wir, wenn wir uns auf die Gleichheitsrelation beschränken. Das einzige, was wir also von zwei Mengenelementen sagen werden, ist, ob sie gleich oder verschieden sind. Diese Beschränkung hat den großen Vorteil, daß die hier und im folgenden definierten Größen Erhaltungsgrößen eines HPNs sind und damit invariant unter unterschiedserhaltenden Transformationen (z. B. Hyperpermutationen) sind.

5.2.2.2 Relative Diversität

Bis jetzt ist die Diversität eine Größe, die uns in absoluten Zahlen die Verschiedenheit einer Multimenge angibt. Wir können die Diversität auch als relative Größe einführen, die uns dann Wahrscheinlichkeitsaussagen über eine Multimenge ermöglicht.

Sei S ein Sensor, der uns k Bit lange Bitstrings liefert, von denen wir N beobachtet haben. Diese Bitstrings fassen wir in der Multimenge $\mathcal{A} = (b_0, b_1, \ldots, b_{N-1})$ zusammen. Wir betrachten nun den Sensor als Quelle $(\mathcal{Q}, \boldsymbol{p})$ im informationstheoretischen Sinn, deren Nachrichtenvorrat $\mathcal{Q} = \{00\ldots00, 00\ldots01, \ldots, 11\ldots11\}$ aus 2^k Bitstrings bekannt ist. Die Emissionswahrscheinlichkeiten p_i der einzelnen Bitstrings sind unbekannt, wir wissen aber, daß die Bitstrings multinomialverteilt sind.

Durch Auszählen der Multimenge $\mathcal{A}$ erhalten wir den Gewichtsvektor $\boldsymbol{w}(\mathcal{A})$, der angibt, wie oft welcher Bitstring vorgekommen ist. Normieren wir den Gewichtsvektor $\boldsymbol{w}$ mit der Gesamtzahl der beobachteten Bitstrings, erhalten wir den Vektor der relativen Häufigkeiten $(w_0/N, w_1/N, \ldots, w_{N-1}/N)$. Nun wissen wir aber aus der Wahrscheinlichkeitstheorie, daß die relativen Häufigkeiten einer Multinomialverteilung gerade die erwartungstreuen Schätzungen der unbekannten Emissionswahrscheinlichkeiten p_i der Quelle $\mathcal{Q}$ sind. Wir schreiben daher für den Vektor der Emissionswahrscheinlichkeiten

$$\boldsymbol{p} = \frac{\boldsymbol{w}}{N}\,, \qquad \text{also} \qquad p_i = \frac{w_i}{N}\,. \tag{5.4}$$

Normieren wir nun die totale, kalte und heiße Diversität mit der totalen Diversität, erhalten wir mit Gleichung (5.3) auf Seite 70 und (5.4) die relativen Diversitäten:

$$\begin{aligned}
d_\mathrm{t} &= \frac{D_\mathrm{t}}{D_\mathrm{t}} = 1 \\
d_\mathrm{k} &= \frac{D_\mathrm{k}}{D_\mathrm{t}} = \frac{1}{N^2}\sum_{i=0}^{2^k-1} w_i^2 = \sum_{i=0}^{2^k-1}\left(\frac{w_i}{N}\right)^2 \\
&= \sum_{i=0}^{2^k-1} p_i^2 \\
d_\mathrm{h} &= \frac{D_\mathrm{h}}{D_\mathrm{t}} = \frac{D_\mathrm{t} - D_\mathrm{k}}{D_\mathrm{t}} = 1 - \frac{D_\mathrm{k}}{D_\mathrm{t}} \\
&= 1 - \sum_{i=0}^{2^k-1} p_i^2
\end{aligned}$$

Natürlich gilt auch für die relativen Diversitäten $d_\mathrm{t} = d_\mathrm{k} + d_\mathrm{h}$.

Die relativen Diversitäten lassen sich nun *unmittelbar interpretieren*: Greifen wir aus der Menge $\mathcal{A}$ der emittierten Daten der Quelle zufällig zwei Elemente heraus, so ist die

- *heiße* relative Diversität d_h die Wahrscheinlichkeit, daß die beiden Elemente *verschieden* sind und die
- *kalte* relative Diversität d_k die Wahrscheinlichkeit, daß die beiden Elemente *gleich* sind.

5.2.3 Information und Redundanz

Bisher war die *Bedeutung* der Elemente der Multimenge uninteressant. Es wurde nur gefragt, wieviele Elemente paarweise unterscheidbar sind und wieviele nicht. Nun soll auch die *Bedeutung* der Elemente eine Rolle spielen. Die Bedeutung eines Bitstrings macht nur im Kontext einer bestimmten Aufgabenstellung einen Sinn. Wir fragen also zusätzlich zur paarweisen Unterscheidbarkeit, welche der *möglichen* Unterscheidungen der Elemente für die Lösung der Aufgabe überhaupt *nötig* sind. Die abstrakte Bewertungsinstanz, die diese Frage beantwortet, nennen wir im folgenden *Lehrer*.

Um konkret zu werden, betrachten wir wieder die Multimenge $\mathcal{M}$. Die Beispielaufgabe soll nun lauten:

> „Teile die Elemente der Multimenge $\mathcal{M}$ in die Klasse 0 „Konsonanten“ und die Klasse 1 „Vokale“.“

Es gibt jetzt paarweise Unterschiede, z. B. zwischen 'a' und 'U', die bezogen auf diese Aufgabe keine Rolle mehr spielen, weil die Elemente des Paares ('a','U') in dieselbe Klasse „Vokale“ fallen. Durch den Kontext der Aufgabe wird die Multimenge $\mathcal{M}$ in zwei Teilmultimengen $\mathcal{M}_j$ geteilt:

$$\begin{aligned}\mathcal{M}_0 &= \{m \in \mathcal{M} \mid \text{„Das Element } m \text{ gehört zur Klasse 0“}\}\\ \mathcal{M}_1 &= \{m \in \mathcal{M} \mid \text{„Das Element } m \text{ gehört zur Klasse 1“}\}\end{aligned}$$

Entsprechend wird die Multimenge der Paare $(m_k, m_l) \in \mathcal{M}^2 := \mathcal{M} \times \mathcal{M}$ in zwei Teilmultimengen unterteilt, und zwar in die der redundanten Paare $\mathcal{N}_\mathrm{r}$ und die der informationstragenden Paare $\mathcal{N}_\mathrm{i}$. Ein Paar (m_k, m_l) ist nun genau dann informationstragend, wenn m_k und m_l zu unterschiedlichen Klassen gehören und redundant, wenn m_k und m_l zu der gleichen Klasse gehören:

$$\begin{aligned}\mathcal{N}_\mathrm{i} &= \{(m_k, m_l) \in \mathcal{M}^2 \mid m_k \in M_k \wedge m_l \notin M_k\}\\ \mathcal{N}_\mathrm{r} &= \{(m_k, m_l) \in \mathcal{M}^2 \mid m_k \in M_k \wedge m_l \in M_k\}\end{aligned}$$

Auch die Repräsentationsmultimenge $\mathcal{B}$ zerfällt durch die Aufgabe in zwei Teilmultimengen:

$$\begin{aligned}\mathcal{B}_0 &= S(\mathcal{M}_0)\\ \mathcal{B}_1 &= S(\mathcal{M}_1)\end{aligned}$$

Für unser Beispiel bedeutet dies, daß die Repräsentationsmultimenge $\mathcal{B}$ in $\mathcal{B}_0 = \{01, 01, 11\}$ und in $\mathcal{B}_1 = \{00, 10\}$ aufgeteilt wird. Ordnen wir jeder klassenspezifischen Repräsentationsmultimenge $\mathcal{B}_k$ entsprechend dem Vorgehen in Gleichung (5.2) auf Seite 70 einen Häufigkeitsvektor zu, erhalten wir die klassenspezifischen Häufigkeitsvektoren $\boldsymbol{w}_k$. Für die klassenspezifischen Häufigkeitsvektoren gilt $\boldsymbol{w} = \sum_{k=0}^{K-1} \boldsymbol{w}_k$, K ist die Anzahl der Klassen. Hier ist

$$\begin{aligned}\boldsymbol{w}_0(\mathcal{B}_0) = \boldsymbol{w}_0(\{01, 01, 11\}) &= \begin{pmatrix}0\\2\\0\\1\end{pmatrix}\\ \boldsymbol{w}_1(\mathcal{B}_1) = \boldsymbol{w}_1(\{00, 10\}) &= \begin{pmatrix}1\\0\\1\\0\end{pmatrix}\end{aligned} \tag{5.5}$$

und $\boldsymbol{w} = \boldsymbol{w}_0 + \boldsymbol{w}_1$. Auch die Paarmultimenge $\mathcal{B}^2$ unterteilen wir in die Multimenge der redundanten Paare $\mathcal{C}_\mathrm{r}$ und die Multimenge der informationstragenden Paare $\mathcal{C}_\mathrm{i}$:

$$\begin{aligned}\mathcal{C}_\mathrm{r} &= \{(S(m_k), S(m_l)) \mid (m_k, m_l) \in \mathcal{N}_\mathrm{r}\}\\ \mathcal{C}_\mathrm{i} &= \{(S(m_k), S(m_l)) \mid (m_k, m_l) \in \mathcal{N}_\mathrm{i}\}\end{aligned}$$

Der Begriff Redundanz meint im folgenden die Anzahl der Elemente der Multimenge der unwichtigen Unterschiede. Wir beginnen mit der Redundanz und nicht mit der Information, weil sich die Information einer Multimenge leicht berechnen läßt, wenn man die Redundanz schon ermittelt hat.

Redundanz

Analog zur Definition der *totalen*, *kalten* und *heißen Diversität* läßt sich nun die *totale*, *kalte* und *heiße Redundanz* definieren:

Totale Redundanz R_t**:** Die *totale Redundanz* ist die gesamte Anzahl von Paaren (b_k, b_l), die im Hinblick auf die gestellte Aufgabe *irrelevant* sind.

Kalte Redundanz R_k**:** Die *kalte Redundanz* ist die Anzahl von Paaren (b_k, b_l), für die $b_k = b_l$ ist *und* die im Kontext der Aufgabe *irrelevant* sind.

Heiße Redundanz R_h**:** Die *heiße Redundanz* ist die Anzahl von Paaren (b_k, b_l), für die $b_k \neq b_l$ ist *und* die im Kontext der Aufgabe *irrelevant* sind.

Oder formal:

Definition 21.

$$\begin{aligned} R_\mathrm{t} &= |\mathcal{C}_\mathrm{r}| \\ R_\mathrm{k} &= |\{(b_k, b_l) \in \mathcal{C}_\mathrm{r} \mid b_k = b_l\}| \\ R_\mathrm{h} &= |\{(b_k, b_l) \in \mathcal{C}_\mathrm{r} \mid b_k \neq b_l\}| \end{aligned} \tag{5.6}$$

Folgerung 6.

$$\boxed{R_\mathrm{t} = R_\mathrm{h} + R_\mathrm{k}}$$

Der Unterschied zwischen den Zeichen 'a' und 'U' sowie zwischen ihren Repräsentationen (siehe Seite 67) ist also redundant, weil sie beide zur Klasse „Vokale“ gehören. Die redundanten, „heißen“ Paare sind ('a','U') und zweimal ('k','f'). Entsprechend kommt ('U','a') und zweimal ('f','k') vor. Insgesamt sind dies sechs Paare. Für die Beispielmultimenge $\mathcal{B}$ ist also die *heiße Redundanz* $R_\mathrm{h} = 6$.

Im Gegensatz zur Diversitätsmatrix werden von der Redundanzmatrix nur die Paare gezählt, die im Sinne der Aufgabe redundant sind. Die Redundanzmatrix läßt sich elegant berechnen, wenn man statt des Häufigkeitsvektors $\boldsymbol{w}$ die klassenspezifischen Häufigkeitsvektoren $\boldsymbol{w}_k$ verwendet. Über die klassenspezifischen Redundanzmatrizen $\boldsymbol{R}_k = \boldsymbol{w}_k \cdot \boldsymbol{w}_k^\mathrm{T}$ läßt sich dann die klassenübergreifende Redundanzmatrix $\boldsymbol{R} = \sum_{k=0}^{K-1} \boldsymbol{R}_k$ berechnen:

$$\begin{aligned} \boldsymbol{R} &= \boldsymbol{R}_0 + \boldsymbol{R}_1 = \boldsymbol{w}_0 \cdot \boldsymbol{w}_0^\mathrm{T} + \boldsymbol{w}_1 \cdot \boldsymbol{w}_1^\mathrm{T} \\ &= \begin{pmatrix} 0 \\ 2 \\ 0 \\ 1 \end{pmatrix} \cdot (0,2,0,1) + \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \end{pmatrix} \cdot (1,0,1,0) \\ &= \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & 2 \\ 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 1 \end{pmatrix} + \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 4 & 0 & 2 \\ 1 & 0 & 1 & 0 \\ 0 & 2 & 0 & 1 \end{pmatrix} \end{aligned}$$

Mit den oben genannten Definitionen für die *totale*, *kalte* und *heiße Redundanz* kommt man bei einer Multimenge mit $b \in \mathbb{B}^n$ auf folgende Beziehungen:

$$\boxed{\begin{aligned} R_\mathrm{t} &= \sum_{i=0,j=0}^{2^n-1} \boldsymbol{R}_{ij} \\ R_\mathrm{k} &= \mathrm{Sp}(\boldsymbol{R}) \\ R_\mathrm{h} &= R_\mathrm{t} - R_\mathrm{k} = \sum_{i=0,j=0}^{2^n-1} \boldsymbol{R}_{ij} - \mathrm{Sp}(\boldsymbol{R}) = \sum_{\substack{i=0,j=0 \\ i\neq j}}^{2^n-1} \boldsymbol{R}_{ij} \end{aligned}} \tag{5.7}$$

Information

Nachdem nun die irrelevanten Unterschiede durch die *Redundanz* quantifiziert sind, müssen wir noch diejenigen Unterschiede behandeln, die im Rahmen der Aufgabenstellung relevant sind. Die Definitionen der *totalen*, *kalten* und *heißen Information* ergeben sich daher aus denen der *Redundanz*, indem wir jeweils das Wort „irrelevant“ durch „relevant“ ersetzen:

Totale Information I_t**:** Die *totale Information* ist die gesamte Anzahl von Paaren (b_k, b_l), die im Hinblick auf die gestellte Aufgabe *relevant* sind.

Kalte Information I_k**:** Die *kalte Information* ist die Anzahl von Paaren (b_k, b_l), für die $b_k = b_l$ ist *und* die im Kontext der Aufgabe *relevant* sind.

Heiße Information I_h**:** Die *heiße Information* ist die Anzahl von Paaren (b_k, b_l), für die $b_k \neq b_l$ ist *und* die im Kontext der Aufgabe *relevant* sind.

Formal:

Definition 22.

$$\begin{aligned} I_\mathrm{t} &= |\mathcal{C}_\mathrm{i}| \\ I_\mathrm{h} &= |\{(b_k, b_l) \in \mathcal{C}_\mathrm{i} \mid b_k \neq b_l\}| \\ I_\mathrm{k} &= |\{(b_k, b_l) \in \mathcal{C}_\mathrm{i} \mid b_k = b_l\}| \end{aligned} \tag{5.8}$$

Folgerung 7.

$$\boxed{I_\mathrm{t} = I_\mathrm{h} + I_\mathrm{k}}$$

Da der „Lehrer“ die Gesamtheit der Paare in zwei Teilmengen teilt, ergibt die Summe der Teilmengen wieder die Gesamtheit der Paare. Dementsprechend gilt für die Matrizen der *Diversität*, *Redundanz* und *Information*:

$$\boxed{\boldsymbol{D} = \boldsymbol{I} + \boldsymbol{R}} \tag{5.9}$$

Wir bekommen also die Informationsmatrix durch

$$\boldsymbol{I} = \boldsymbol{D} - \boldsymbol{R} = \begin{pmatrix} 1 & 2 & 1 & 1 \\ 2 & 4 & 2 & 2 \\ 1 & 2 & 1 & 1 \\ 1 & 2 & 1 & 1 \end{pmatrix} - \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 4 & 0 & 2 \\ 1 & 0 & 1 & 0 \\ 0 & 2 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 2 & 0 & 1 \\ 2 & 0 & 2 & 0 \\ 0 & 2 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix}$$

Der Unterschied zwischen den Zeichen 'a' und 'k' ist informationstragend, weil sie zu verschiedenen Klassen gehören. Wie bei der *Redundanz* kommt man auch bei der *Information* auf die drei Beziehungen

$$\begin{aligned} I_\mathrm{t} &= \sum_{i=0,j=0}^{2^n-1} \boldsymbol{I}_{ij} \\ I_\mathrm{k} &= \mathrm{Sp}(\boldsymbol{I}) \\ I_\mathrm{h} &= I_\mathrm{t} - I_\mathrm{k} = \sum_{i=0,j=0}^{2^n-1} \boldsymbol{I}_{ij} - \mathrm{Sp}(\boldsymbol{I}) = \sum_{\substack{i=0,j=0 \\ i\neq j}}^{2^n-1} \boldsymbol{I}_{ij} \end{aligned} \tag{5.10}$$

Weil die *kalte Information* die Anzahl von Paaren angibt, die zwar für die Aufgabe wichtig sind, aber innerhalb der Repräsentationsmultimenge $\mathcal{B}$ nicht unterschieden werden können, sollte die *kalte Information* möglichst klein sein. Das bedeutet für die Informationsmatrix $\boldsymbol{I}$ wegen $I_\mathrm{k} = \mathrm{Sp}(\boldsymbol{I})$, daß alle Hauptdiagonalelemente null sein sollten. Bei unserem einfachen Beispiel ist dies möglich, da die Klassen trennbar sind.

Die im Rahmen einer Aufgabe wichtigste Größe ist die *heiße Information*, denn sie gibt an, wieviele Paare für die Aufgabe wichtig sind *und* in der Repräsentationsmultimenge unterschieden werden können. Wenn wir später eine Datenreduktion vornehmen wollen, muß daher ein Ziel sein, die *heiße Information* möglichst groß zu halten. Für die Beispielmultimenge $\mathcal{B}$ ist die *heiße Information* $I_\mathrm{h} = 12$.

5.2.3.1 Relative Information und Redundanz

Auch die Information und Redundanz lassen sich als relative Größen definieren und interpretieren, wobei K die Anzahl der Klassen, n die Dimension der klassenspezifischen Gewichtsvektoren $\boldsymbol{w}_k$ und N die Zahl der beobachteten Bitstrings ist:

$$\begin{aligned} r_\mathrm{t} &= \frac{R_\mathrm{t}}{D_\mathrm{t}} \\ r_\mathrm{k} &= \frac{R_\mathrm{k}}{D_\mathrm{t}} = \frac{1}{N^2} \sum_{k=0}^{K-1} \boldsymbol{w}_k^\mathrm{T} \cdot \boldsymbol{w}_k = \sum_{k=0}^{K-1} \sum_{i=0}^{n-1} \left(\frac{w_{k,i}}{N}\right)^2 \\ &= \sum_{k=0}^{K-1} \sum_{i=0}^{n-1} p_{k,i}^2 \\ r_\mathrm{h} &= \frac{R_\mathrm{h}}{D_\mathrm{t}} = \frac{R_\mathrm{t} - R_\mathrm{k}}{D_\mathrm{t}} = r_\mathrm{t} - r_\mathrm{k} \\ i_\mathrm{t} &= \frac{I_\mathrm{t}}{D_\mathrm{t}} = d_\mathrm{t} - r_\mathrm{t} \\ i_\mathrm{k} &= \frac{I_\mathrm{k}}{D_\mathrm{t}} = d_\mathrm{k} - r_\mathrm{k} \\ i_\mathrm{h} &= \frac{I_\mathrm{h}}{D_\mathrm{t}} = d_\mathrm{h} - r_\mathrm{h} \end{aligned}$$

Auch diese relativen Größen lassen sich unmittelbar interpretieren: Greifen wir aus der Menge $\mathcal{A}$ der emittierten Daten der Quelle zufällig zwei Elemente heraus, haben die relativen Größen folgende Bedeutungen:

- Die *heiße* relative Redundanz r_h ist die Wahrscheinlichkeit, daß die beiden Elemente *verschieden* sind und dieser Unterschied *irrelevant* für die Aufgabe ist.
- Die *kalte* relative Redundanz r_k ist die Wahrscheinlichkeit, daß die beiden Elemente *gleich* sind und dieser Unterschied *irrelevant* für die Aufgabe ist.
- Die *heiße* relative Information i_h ist die Wahrscheinlichkeit, daß die beiden Elemente *verschieden* sind und dieser Unterschied *relevant* für die Aufgabe ist.
- Die *kalte* relative Information i_k ist die Wahrscheinlichkeit, daß die beiden Elemente *gleich* sind und dieser Unterschied *relevant* für die Aufgabe ist.

Die hier eingeführte Information hat im Gegensatz zur Shannon'schen Information also eine Bedeutung, die durch die gestellte Aufgabe definiert ist.

5.2.4 Beziehungen zwischen Diversität, Information und Redundanz

Die im vorigen Abschnitt genannten Beziehungen führen wir hier nochmal zusammen auf. Über die Definitionen von *Diversität*, *Redundanz* und *Information* kommt man auf

$$\begin{aligned} D_\mathrm{t} &= D_\mathrm{h} + D_\mathrm{k} \\ R_\mathrm{t} &= R_\mathrm{h} + R_\mathrm{k} \\ I_\mathrm{t} &= I_\mathrm{h} + I_\mathrm{k} \end{aligned} \tag{5.11}$$

und die Matrixgleichung

$$\boldsymbol{D} = \boldsymbol{I} + \boldsymbol{R}\,. \tag{5.12}$$

Verbinden wir die Gleichungen (5.11) und (5.12), kommen wir außerdem auf die drei Beziehungen

$$\begin{aligned} D_\mathrm{t} &= I_\mathrm{t} + R_\mathrm{t} \\ D_\mathrm{h} &= I_\mathrm{h} + R_\mathrm{h} \\ D_\mathrm{k} &= I_\mathrm{k} + R_\mathrm{k}\,. \end{aligned} \tag{5.13}$$

Die Beziehungen zwischen den neun Größen aus Gleichung (5.11) und (5.13) kann man sich auch in folgendem rechteckigen Schema angeordnet vorstellen, das von links nach rechts und von oben nach unten zu lesen ist:

$$\boxed{\begin{array}{ccccc} D_\mathrm{t} & = & I_\mathrm{t} & + & R_\mathrm{t} \\ \| & & \| & & \| \\ D_\mathrm{h} & = & I_\mathrm{h} & + & R_\mathrm{h} \\ + & & + & & + \\ D_\mathrm{k} & = & I_\mathrm{k} & + & R_\mathrm{k} \end{array}} \tag{5.14}$$

Diese Beziehungen gelten auch für die entsprechenden relativen Größen. Für unser Beispiel ergibt sich daraus:

$$\begin{array}{ccccc} 25 & = & 12 & + & 13 \\ \| & & \| & & \| \\ 18 & = & 12 & + & 6 \\ + & & + & & + \\ 7 & = & 0 & + & 7 \end{array}$$

Hier ist $I_\mathrm{k} = 0$ und $I_\mathrm{h} = I_\mathrm{t}$, d. h. alle in den Ereignissen enthaltene *Information* ist auch in Repräsentationsmultimenge vorhanden. Im Sinne der *Information* ist die Repräsentation also optimal, weil der formale Sensor S keine Information vernichtet. Die heiße Redundanz ist aber $R_\mathrm{h} = 6 > 0$, d. h. es gibt noch heiße (unterscheidbare) Paare in der Repräsentationsmultimenge, die eigentlich gar nicht unterschieden werden müßten. Bei einer optimalen Abbildung sollte daher nicht nur $I_\mathrm{h} = I_\mathrm{t}$, sondern auch $R_\mathrm{h} = 0$ sein. In der Praxis wird man dies nur in den einfachsten Fällen erreichen können.

5.3 Ein Beispiel zur Berechnung der statistischen Größen

5.3.1 Trennung von „Vokalen und Konsonanten“

Um zu verstehen, wie wir mit den Größen Diversität, Information und Redundanz die Optimalität einer Knotentabelle bewerten können, betrachten wir nun das Beispiel aus Abschnitt 5.2.3 auf Seite 72, die Multimenge $\mathcal{M} = \{\text{'a', 'k', 'U', 'k', 'f'}\}$ und ihre Repräsentationsmultimenge $\mathcal{B} = \{00, 01, 10, 01, 11\}$. Die Multimengen $\mathcal{M}$ und $\mathcal{B}$ sind absichtlich so unrealistisch klein gehalten, damit man im folgenden die Rechnungen unmittelbar nachvollziehen kann. Die Aufgabe ist:

> „Teile die Elemente der Multimenge $\mathcal{M}$ in die Klasse 0 „Konsonanten“ und die Klasse 1 „Vokale“.“

Damit wird die Multimenge $\mathcal{B}$ in die Klasse $\mathcal{B}_0 = \{01, 01, 11\}$ und die Klasse $\mathcal{B}_1 = \{00, 10\}$ aufgeteilt. Wir werden am Eingang des Knotens die Eingangssymbole $\boldsymbol{x} \in \mathcal{B} \subseteq \mathbb{B}^2$ einspeisen, und am Ausgang des Knotens sollen die Ausgangssymbole $\boldsymbol{y} \in \mathbb{B}^2$ so herauskommen, daß die gestellte Aufgabe gelöst werden kann. Um zwei Klassen auseinanderzuhalten, müssen wir alle möglichen Fälle voneinander trennen können. Dazu reicht ein Bit aus, das beim Auftreten der einen Klasse den Wert 0 und bei der anderen Klasse den Wert 1 hat oder umgekehrt. Wir werden also von den beiden Ausgangsleitungen des Knotens nur die obere beobachten und setzen den Fokus daher auf $y = {\phi^{\mathrm{ko}}}_{0,0}(\boldsymbol{y}) \in \mathbb{B}$.

Durch diese Projektion von zwei auf ein Bit verringert sich die Zahl der verschiedenen Symbole von vier auf zwei. Dadurch gehen notwendigerweise Unterscheidungsmöglichkeiten verloren. Das Ziel ist nun, mit Hilfe der Hyperpermutation die Daten *so* zu transformieren, daß die in den Daten steckende Information trotz der Projektion möglichst erhalten bleibt. Die heiße Information soll also maximal werden.

5.3.2 Berechnung der Ein- und Ausgangsstatistik

Die Berechnung der Statistik am Ein- und Ausgang des Knoten geht folgendermaßen: Wir zählen für jedes $\boldsymbol{x} \in \mathcal{B}$ und $y \in \mathbb{B}$, wie oft es pro Klasse vorkommt, und bilden so die klassenspezifischen Häufigkeitsvektoren, wie in Gleichung (5.5) auf Seite 73. Die klassenspezifischen Häufigkeitsvektoren sind:

$$\boldsymbol{w}_0(\mathcal{B}_0) = \begin{pmatrix} 0 \\ 2 \\ 0 \\ 1 \end{pmatrix} \qquad \boldsymbol{w}_1(\mathcal{B}_1) = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

Die totale Diversität, Information und Redundanz der Multimenge $\mathcal{B}$ sind

$$D_\mathrm{t} = 25; \qquad I_\mathrm{t} = 12; \qquad R_\mathrm{t} = 13\,. \tag{5.15}$$

Da jede vom Knoten durchgeführte Hyperpermutation unterschiedserhaltend ist, bekommen wir die gleichen Zahlen wie in (5.15), wenn wir am Ausgang des Knotens den Zustand *beider* Leitungen als Ausgangssymbol benutzen.

Suboptimale Hyperpermutation

Nun betrachten wir nur die obere Leitung und nehmen als Beispiel die willkürlich gewählte Hyperpermutation

$$\mathrm{h}_1: \quad \begin{array}{|c||c|c|c|c|} \hline \boldsymbol{x} & 00 & 01 & 10 & 11 \\ \hline \boldsymbol{y} & 10 & 11 & 01 & 00 \\ \hline \end{array} \quad .$$

Um die Projektion auf die obere (0.) Leitung durchzuführen, gehen wir zum Produktraum über, in der die Hyperpermutation h_1 die Form

$$\boldsymbol{H}_1 = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

und der Fokus ${\phi^{\mathrm{pr}}}_{0,0}$ die Form

$$\boldsymbol{\Phi}_0 = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix}$$

annimmt. Die projizierten Häufigkeitsvektoren $\boldsymbol{w}'_0$ und $\boldsymbol{w}'_1$ für die obere Ausgangsleitung erhalten wir, wenn wir erst $\boldsymbol{H}_1$ und dann ${\phi^{\mathrm{pr}}}_{0,0}$ auf die klassenspezifischen Häufigkeitsvektoren $\boldsymbol{w}_0$ und $\boldsymbol{w}_1$ am Eingang anwenden:

$$\boldsymbol{w}'_0 = \boldsymbol{\Phi}_0 \cdot \boldsymbol{H}_1 \cdot \boldsymbol{w}_0 = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 2 \\ 0 \\ 1 \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0 \\ 0 \\ 2 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$$

$$\boldsymbol{w}'_1 = \boldsymbol{\Phi}_0 \cdot \boldsymbol{H}_1 \cdot \boldsymbol{w}_1 = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Nachdem wir nun die klassenspezifischen Häufigkeitsvektoren haben, kennen wir auch den klassenübergreifenden Häufigkeitsvektor $\boldsymbol{w}'$:

$$\boldsymbol{w}' = \boldsymbol{w}'_0 + \boldsymbol{w}'_1 = \begin{pmatrix} 1 \\ 2 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 2 \\ 3 \end{pmatrix}$$

Nun können wir wie in Abschnitt 5.2.3 über die Matrizen $\boldsymbol{D}$, $\boldsymbol{I}$, und $\boldsymbol{R}$ die heiße Information berechnen und so feststellen, ob wir mit dieser Hyperpermutation eine optimale gewählt haben. Es gibt ja nicht *die* optimale Tabelle, sondern mehrere. Das liegt daran, daß wir nur die obere der beiden Ausgangsleitungen betrachten. Deshalb sind die Hyperpermutationen, die sich bei gleichem Eingangssymbol $\boldsymbol{x}$ in der betrachteten Ausgangsleitung gleichen, gleichwertig.

Die Informationsmatrix $\boldsymbol{I}$ ergibt sich unter Verwendung der Gleichung (5.9) auf Seite 75:

$$\begin{aligned}\boldsymbol{I} = \boldsymbol{D} - \boldsymbol{R} &= \boldsymbol{w}' \cdot (\boldsymbol{w}')^{\mathrm{t}} - (\boldsymbol{w}_0 \cdot \boldsymbol{w}_0^{\mathrm{T}} + \boldsymbol{w}_1 \cdot \boldsymbol{w}_1^{\mathrm{T}}) \\ &= \begin{pmatrix}2\\3\end{pmatrix} \cdot (2,3) - \begin{pmatrix}1\\2\end{pmatrix} \cdot (1,2) - \begin{pmatrix}1\\1\end{pmatrix} \cdot (1,1) \\ &= \begin{pmatrix}4 & 6\\6 & 9\end{pmatrix} - \begin{pmatrix}1 & 2\\2 & 4\end{pmatrix} - \begin{pmatrix}1 & 1\\1 & 1\end{pmatrix} \\ &= \begin{pmatrix}2 & 3\\3 & 4\end{pmatrix}\end{aligned} \tag{5.16}$$

Damit bekommen wir die heiße Information I_{h}:

$$I_{\mathrm{h}} = I_{\mathrm{t}} - \mathrm{Sp}(\boldsymbol{I}) = 12 - 6 = 6$$

Bei dieser Projektion ist die gewählte Hyperpermutation h_1 nicht optimal, da $I_{\mathrm{k}} = \mathrm{Sp}(\boldsymbol{I}) > 0$ und daher $I_{\mathrm{h}} < I_{\mathrm{t}}$. Es gibt also für die Aufgabe gerade I_{k} wichtige Unterschiede, die nach der Transformation und Projektion nicht mehr sichtbar sind. Wenn wir die Elemente von $\mathcal{B}$ als Eingangssymbole nehmen, können wir die Ausgangssymbole des Knotens wie in Tabelle 5.2 aufschreiben. Die nicht mehr unterscheidbaren Paare sind das Paar ('a', 'f'), zweimal das Paar ('k', 'U') und jeweils die Paare mit vertauschten Elementen, macht zusammen 6 Paare. Mit h_1 erhalten wir also nur $I_{\mathrm{h}}/I_{\mathrm{t}} = 0.5 = 50\%$ der gesamten Information, die ursprünglich in den Daten war.

$\omega \in \mathcal{M}$	$\boldsymbol{x} \in \mathcal{B}$	$\boldsymbol{y} \in \mathbb{B}^2$	${\phi^{\mathrm{ko}}}_{0,0}(\boldsymbol{y}) \in \mathbb{B}$	Klasse
'a'	00	10	0	Vokal
'k'	01	11	1	Konsonant
'U'	10	01	1	Vokal
'k'	01	11	1	Konsonant
'f'	11	00	0	Konsonant

Tabelle 5.2: Suboptimale Klassifikation der Multimenge $\mathcal{M}$ mit der Hyperpermutation h_1

Optimale Hyperpermutation

Im Gegensatz dazu ist eine mögliche optimale Hyperpermutation h_2

h_2 :

$\boldsymbol{x}$	00	01	10	11
$\boldsymbol{y}$	11	10	01	00

.

Um dies zu sehen, führen wir die gleiche Rechnung wie eben durch. Wir erhalten für die klassenspezifischen Häufigkeitsvektoren

$$\boldsymbol{w}_0' = \boldsymbol{\Phi}_0 \cdot \boldsymbol{H}_2 \cdot \boldsymbol{w}_0 = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 2 \\ 0 \\ 1 \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0 \\ 2 \\ 0 \end{pmatrix} = \begin{pmatrix} 3 \\ 0 \end{pmatrix}$$

$$\boldsymbol{w}_1' = \boldsymbol{\Phi}_0 \cdot \boldsymbol{H}_2 \cdot \boldsymbol{w}_1 = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 2 \end{pmatrix}$$

und für den klassenübergreifenden Häufigkeitsvektor

$$\boldsymbol{w}' = \begin{pmatrix} 3 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 2 \end{pmatrix} = \begin{pmatrix} 3 \\ 2 \end{pmatrix} .$$

Die Informationsmatrix ist

$$\begin{aligned} \boldsymbol{I} = \boldsymbol{D} - \boldsymbol{R} &= \boldsymbol{w}' \cdot (\boldsymbol{w}')^{\mathrm{t}} - (\boldsymbol{w}_0' \cdot (\boldsymbol{w}_0')^{\mathrm{t}} + \boldsymbol{w}_1' \cdot (\boldsymbol{w}_1')^{\mathrm{t}}) \\ &= \begin{pmatrix} 3 \\ 2 \end{pmatrix} \cdot (3,2) - \begin{pmatrix} 3 \\ 0 \end{pmatrix} \cdot (3,0) - \begin{pmatrix} 0 \\ 2 \end{pmatrix} \cdot (0,2) \\ &= \begin{pmatrix} 9 & 6 \\ 6 & 4 \end{pmatrix} - \begin{pmatrix} 9 & 0 \\ 0 & 0 \end{pmatrix} - \begin{pmatrix} 0 & 0 \\ 0 & 4 \end{pmatrix} \\ &= \begin{pmatrix} 0 & 6 \\ 6 & 0 \end{pmatrix} \end{aligned} \tag{5.17}$$

und die heiße Information

$$I_{\mathrm{h}} = I_{\mathrm{t}} - \mathrm{Sp}(\boldsymbol{I}) = 12 - 0 = 12 .$$

Bei der Hyperpermutation h_2 ist $I_{\mathrm{k}} = 0$ und $I_{\mathrm{h}} = I_{\mathrm{t}}$. Wir erhalten also alle Information, die in den Daten überhaupt enthalten ist, mit dem oberen Ausgangsbit des Knotens. Wir können jetzt diese Hyperpermutation zur Lösung unserer Aufgabe nutzen, indem wir den Wert des oberen Ausgangsbits als Klassenindex verwenden, so wie in Tabelle 5.3 gezeigt wird. Der Knoten trennt also die Eingangsdaten in die gewünschten zwei Klassen „Vokale“ und „Konsonanten“, wobei die Klasse „Vokale“ den Index 1 und die Klasse „Konsonanten“ den Index 0 bekommt.

Der Normalfall in der Praxis ist, daß die Klassen nicht vollständig separierbar sind und daß bei der Transformation der Daten Information verloren geht. Das Ziel ist dann, eine Tabelle zu finden, die eben möglichst viel Information erhält, d. h. die I_{k} ($= \mathrm{Sp}(\boldsymbol{I})$) möglichst klein bzw. I_{h} möglichst groß werden läßt.

$\omega \in \mathcal{M}$	$\boldsymbol{x} \in \mathcal{B}$	$\boldsymbol{y} \in \mathbb{B}^2$	${\phi^{\mathrm{ko}}}_{0,0}(\boldsymbol{y}) \in \mathbb{B}$	Klasse
'a'	00	11	1	Vokal
'k'	01	10	0	Konsonant
'U'	10	01	1	Vokal
'k'	01	10	0	Konsonant
'f'	11	00	0	Konsonant

Tabelle 5.3: Optimale Klassifikation der Multimenge $\mathcal{M}$ mit der Hyperpermutation h_2

Teil II

Optimierung

6 Optimierung durch Lernen

In den vorangegangenen Abschnitten haben wir mit den Hyperpermutationsnetzwerken ein Berechnungsmodell und mit der Information ein Gütemaß für Hyperpermutationsnetzwerke vorgestellt. Nun wollen wir vor allem rückgekoppelte HPNs so mit Beispielen trainieren, daß sie eine bestimmte Mustererkennungsaufgabe möglichst gut erfüllen.

6.1 Ziel der Optimierung

Das Ziel der Optimierung scheint zunächst klar zu sein: Eine gestellte Wahrnehmungs- bzw. Mustererkennungsaufgabe soll gelöst werden. Wie wir noch sehen werden, ist bei genauerem Hinsehen das Ziel einer Mustererkennungsaufgabe nicht immer eindeutig. Die beiden Hauptanwendungen eines Mustererkennungssystems sind die Gruppierung (engl. *Clustering*) bzw. die Klassfikation unbekannter Daten. Sollen unbekannte Daten nur gruppiert werden, verwendet man zum Training *unüberwachtes Lernen*. Sollen die unbekannten Daten dagegen in verschiedene Klassen eingeteilt werden, wird *überwachtes Lernen* verwendet. Beim überwachten Lernen gibt es eine Instanz, den sog. *Lehrer*, die entscheidet, welche Muster zu einer Klasse gehören und welche nicht. Beim unüberwachten Lernen gibt es eine solche Instanz nicht.

6.1.1 Unüberwachtes Lernen

Beim unüberwachten Lernen hat man als Datenmaterial nur die Eingangsmuster ohne Klasseninformation. Ein Beispiel für solche Daten sind Gensequenzen, deren Bedeutung noch unbekannt ist. Ein Mustererkennungssystem, das ähnliche Eingangsmuster gruppieren soll, wird nun unüberwacht trainiert. Zusätzlich zu den Eingangsmustern gibt es noch ein Ähnlichkeits- bzw. Abstandsmaß, also eine Metrik, welche die Ähnlichkeit zweier Eingangsmuster bestimmt. Da es im n-dimensionalen Raum der Eingangsmuster beliebig viele Metriken gibt, muß man sich für eine Metrik entscheiden, die am besten für das Problem geeignet erscheint. Man könnte unüberwachtes Lernen auch mit „ähnliche Eingaben ergeben ähnliche Ausgaben“ beschreiben, da das Mustererkennungssystem für zwei im Sinne des Ähnlichkeitsmaßes ähnliche Eingangsmuster auch ähnliche Ausgaben erzeugen soll.

Ein bekanntes Beispiel für ein neuronales Netz, das unüberwacht trainiert wird, ist die *Kohonen-Feature-Map* (Kohonen, 1984). Dieses neuronale Netz kann dazu genutzt werden, in einem hochdimensionalen Raum der Eingangsmuster Gruppen und Beziehungen zwischen verschiedenen Gruppen der Eingangsmuster zu erkennen sowie unbekannte Eingangsmuster einer der schon vorhandenen Gruppen zuzuordnen. Häufig wird als Metrik die Euklidische Norm verwendet.

6.1.2 Überwachtes Lernen

Überwachtes Lernen zeichnet sich im Gegensatz zum unüberwachten Lernen dadurch aus, daß jedes Eingangsmuster zu einer bestimmten Klasse gehört. Sind die Eingangsmuster beispielsweise Bilder von Ziffern, so gehören alle Bilder, die dieselbe Ziffer repräsentieren, zur selben

Klasse. Ein Mustererkennungssystem, soll nun lernen, die Eingangsmuster aus unterschiedlichen Klassen voneinander zu unterscheiden. Zum Training verwendet man einen Satz von Eingangsmustern mit bekannten Klassenzugehörigkeiten, die sogenannten Trainingsdaten. Mit diesen Trainingsdaten wird solange trainiert, bis das Mustererkennungssystem die Daten den verschiedenen Klassen gut genug zuordnen kann. Was aber „gut genug“ ist, hängt sehr von dem jeweiligen Problem ab.

6.1.3 Maximierung der Zahl der relevanten Unterschiede

Wie erreichen wir nun unser Optimierungsziel? Ein HPN ist ein „Rechenwerk“, das, wie ein Digitalcomputer, Bitstrings auf Bitstrings abbildet. Ohne eine weitere Metrik einzuführen, läßt sich von zwei Bitstrings nur sagen, ob sie gleich oder ungleich sind. Genau dies machen wir uns zu nutze: Wir fordern, daß zwei Eingangsbitstrings aus unterschiedlichen Klassen am Netzwerkausgang auf unterschiedliche Bitstrings abgebildet werden. Bei einem gegebenen Trainingsdatensatz läuft das darauf hinaus, die Zahl der relevanten Unterschiede am Netzwerkausgang zu maximieren. Diese Größe ist aber gerade die *heiße Information*, die wir in Abschnitt 5.2.3 ausführlich eingeführt haben.

Wie wir später noch sehen werden, können wir ein HPN, wenn wir es *global* optimieren, auch direkt auf eine Fehlerrate bzw. eine Kostenmatrix hin optimieren.

6.1.4 Was ist das optimale Suboptimum?

Angenommen, wir haben es mit einem n-Klassen-Problem zu tun, bei dem die Trainingsdaten linear separierbar sind. Dann gibt es verschiedene Klassifikatoren, die die Trainingsdaten perfekt trennen können, die also auf den Trainingsdaten keine Fehler machen. In einem solchen Fall ist klar, das das Optimum dann erreicht ist, wenn beim Training die Fehlerrate bzw. Kostenfunktion des Klassifikators Null ist. Was aber ist, wenn die Fehlerrate auf den Trainingsdaten nicht Null wird, so wie es in der Praxis fast immer der Fall ist? Dann müssen wir uns entscheiden, welches der Suboptima mit einer Fehlerrate größer Null das „beste“ Suboptimum ist.

Ist die Fehlerrate eines Klassifikators auf den Trainingsdaten nicht Null, haben wir meistens verschiedene Konfigurationen, die ähnliche Fehlerraten liefern. Ein wichtiger Punkt ist aber in der Fehlerrate gar nicht enthalten: Wie sicher war sich der Klassifikatior bei seinen Entscheidungen. Betreibt man einen Klassifikator ohne Rückweisung, ist im Sinne der Fehlerrate eine Entscheidung mit 99% Rückschlußwahrscheinlichkeit für eine bestimmte Klasse genauso gut wie eine Entscheidung mit 51% Rückschlußwahrscheinlichkeit für eine Klasse. Im ersten Fall war sich der Klassifikator ziemlich sicher, im zweiten Fall hat er dagegen eigentlich nur geraten.

Nun ist in der Praxis oft von Bedeutung, daß ein Klassifikator für möglichst viele Daten möglichst sichere Entscheidungen trifft. Er wird je nach Anwendung einem Klassifikator vorgezogen, der zwar niedrigere Fehlerrate, aber eine geringere Entscheidungssicherheit hat. Verwenden wir nun die heiße Information als Kostenfunktion, wird diesem Punkt automatisch Rechnung getragen. Denn die heiße Information maximiert die Zahl der relevanten Unterschiede, wodurch neben der Verringerung der Fehlerrate auch die Vergrößerung der klassenspezifischen Rückschlußwahrscheinlichkeiten erreicht wird.

7 Ein Knoten ohne Rückkopplungen

In diesem Kapitel zeigen wir, wie man einen einzelnen Knoten ohne Rückkopplungen optimiert. Genau wie bei neuronalen Netzen gibt es nicht *die* einzig richtige Kostenfunktion, sondern wir haben verschiedene Möglichkeiten.

7.1 Die Kostenfunktion im Diskreten

Wie wir ein HPN als Klassifikator „betreiben“ können, haben wir schon in Abschnitt 4.1.3 auf Seite 30 gesehen: Ein unbekanntes Eingangsmuster wird vom HPN auf ein Ausgangsmuster abgebildet. Für jedes Ausgangsmuster wissen wir aus dem Training die klassenspezifischen Rückschlußwahrscheinlichkeiten, so daß wir uns dann für die Klasse mit der höchsten Rückschlußwahrscheinlichkeit entscheiden.

Wir gehen im folgenden vom überwachten Training aus: Zu jedem Muster des Trainingsdatensatzes ist die zugehörige Klasse bekannt. Wie können also zu jeder Klasse die Häufigkeiten der vorkommenden Bitmuster angeben. Wir stellen zwei mögliche Kostenfunktionen vor, und zwar die *heiße Information* als Kostenfunktion und eine über die sogenannte *Vertauschungsmatrix* definierte Kostenfunktion.

7.1.1 Die heiße Information

Gegeben sei nun ein HPN-Knoten der Dimension d und ein Trainingsdatensatz $\mathcal{T}$ mit N Elementen und K Klassen. Wir gehen nun einmal durch die Trainingsdaten und protokollieren, wie oft ein Bitmuster am Knoteneingang in jeder Klasse vorkommt. So erhalten wir die klassenspezifischen Häufigkeitsvektoren $\boldsymbol{w}_k$ und den Gesamthäufigkeitsvektor $\boldsymbol{w} = \sum_{k=0}^{K-1} \boldsymbol{w}_k$. Die Komponente j des Häufigkeitsvektors w_i bedeutet, daß in der Klasse i das Bitmuster $\mathrm{ind}_d^{-1}(j)$ gerade w_{ij}-mal vorgekommen ist ($\mathrm{ind}_d(j)$ aus Definition 8 auf Seite 35). Mit $d = 4$ und $j = 3$ wäre beispielsweise $\mathrm{ind}_4^{-1}(3) = 0011$.

Sind nun die klassenspezifischen Häufigkeitsvektoren gegeben, läßt sich mit Gleichung (5.10) auf Seite 76 die Informationsmatrix $\boldsymbol{I}$ berechnen:

$$\boldsymbol{I} = \boldsymbol{D} - \boldsymbol{R} = \boldsymbol{w}\boldsymbol{w}^{\mathrm{T}} - \sum_{k=0}^{K-1} \boldsymbol{w}_k \boldsymbol{w}_k^{\mathrm{T}}$$

Wie ändert sich die Informationsmatrix $\boldsymbol{I}$ durch die Transformation mit einem HPN-Knoten? Im Produktraum wird die durch den Knoten vermittelte Hyperpermutation durch die Matrix $\boldsymbol{H}$ und die anschließende Projektion auf den Fokus mit durch die Matrix $\boldsymbol{\Phi}$ beschrieben. Wird ein Gewichtsvektor mittels $\boldsymbol{w}' = \boldsymbol{\Phi}\boldsymbol{H}\boldsymbol{w}$ transformiert, ist seine Dimension kleiner als die von $\boldsymbol{w}$, wodurch die Datenreduktion erreicht wird. Wir fassen im folgenden die Projektionsmatrix und die Hyperpermutationsmatrix zu einer Matrix $\bar{\boldsymbol{H}} = \boldsymbol{\Phi} \cdot \boldsymbol{H}$ zusammen. Für die transformierte

Informationsmatrix $\boldsymbol{I}'$ am Knotenausgang gilt:

$$\boldsymbol{I}' = \boldsymbol{w}'\boldsymbol{w}'^{\mathrm{T}} - \sum_{k=0}^{K-1} \boldsymbol{w}'_k \boldsymbol{w}'^{\mathrm{T}}_k = \bar{\boldsymbol{H}}\boldsymbol{w}\boldsymbol{w}^{\mathrm{T}}\bar{\boldsymbol{H}}^{\mathrm{T}} - \sum_k \bar{\boldsymbol{H}}\boldsymbol{w}_k\boldsymbol{w}_k^{\mathrm{T}}\bar{\boldsymbol{H}}^{\mathrm{T}}$$
$$= \bar{\boldsymbol{H}}\left(\boldsymbol{w}\boldsymbol{w}^{\mathrm{T}} - \sum_k \boldsymbol{w}_k\boldsymbol{w}_k^{\mathrm{T}}\right)\bar{\boldsymbol{H}}^{\mathrm{T}} = \bar{\boldsymbol{H}}\,\boldsymbol{I}\,\bar{\boldsymbol{H}}^{\mathrm{T}}$$

Um am Knotenausgang die heiße Information zu *maximieren*, können wir ebenso die kalte Information *minimieren*, da $I_{\mathrm{h}} = I_{\mathrm{t}} - I_{\mathrm{k}}$ und I_{t} konstant ist. Wir maximieren dann nicht mehr die relevanten unterscheidbaren Paare, sondern wir minimieren die Zahl der relevanten ununterscheidbaren Paare, was äquivalent ist. Die zu minimierende Kostenfunktion ist also die kalte Information am Ausgang des Knotens:

$$\boxed{I'_{\mathrm{k}} = \mathrm{Sp}\left(\boldsymbol{I}'\right) = \mathrm{Sp}\left(\bar{\boldsymbol{H}}\,\boldsymbol{I}\,\bar{\boldsymbol{H}}^{\mathrm{T}}\right) = \boldsymbol{w}'^{\mathrm{T}} \cdot \boldsymbol{w}' - \sum_{k=0}^{K-1} \boldsymbol{w}'^{\mathrm{T}}_k \cdot \boldsymbol{w}'_k} \tag{7.1}$$

7.1.2 Die Vertauschungsmatrix

Bei der eben vorgestellten Kostenfunktion werden alle Klassen gleich behandelt. Es gibt aber in der Praxis durchaus Probleme, bei denen Verwechslungen zwischen den Klassen unterschiedlich bewertet werden. Wir müssen also nicht nur festhalten, wie viele Fehler der Klassifikator macht, sondern auch, *welche* Fehler gemacht wurden. Dazu führen wie die sogenannte *Vertauschungsmatrix* $\boldsymbol{V}$ ein.

Definition 23. Bei einem K-Klassen Problem gibt die $K \times K$ *Vertauschungsmatrix* $\boldsymbol{V}$ eines Klassifikators an, wie oft welche Fehlklassifikation auf einem Datensatz vorgekommen ist. Ein Matrixelement v_{ij} ist gleich der Zahl der Klassifikationen zugunsten der Klasse j, wenn die tatsächliche Klasse i war.

Folgerung 8. Im Sinne der Fehlerrate ist eine *diagonale* Vertauschungsmatrix optimal, da Ist- und Sollklasse für alle Daten übereinstimmen.

Bei einem HPN H ist die Vertauschungsmatrix $\boldsymbol{V} = \boldsymbol{V}(H)$ direkt von den Tabellen in den Netzwerkknoten abhängig, da sich für jede Tabellenbelegung eine andere Ausgangsstatistik und damit andere klassenspezifische Rückschlußwahrscheinlichkeiten für die Trainingsdaten ergeben.

7.1.3 Kombinatorische Optimierung

Die zu minimierenden Kostenfunktionen hängen direkt von den Tabellenbelegungen ab. Da in den Tabellen diskrete Bitmuster stehen, ist die Kostenfunktion weder stetig noch differenzierbar. Wir haben es mit einem kombinatorischen Optimierungsproblem zu tun. Oft ist es nicht leicht einzuschätzen, ob ein kombinatorisches Optimierungsproblem schwierig oder einfach ist.

Die kombinatorische Optimierung rückkopplungsfreier HPN wurde schon in einer Studie an Derigs (1995) untersucht. Es konnte gezeigt werden, daß das vorgestellte Optimierungsproblem NP-vollständig ist. Derigs untersuchte verschiedene Verfahren zur Optimierung solcher Probleme, unter anderem Graph-Matching, Graph-Partionierung und Dynamische Programmierung.

Es zeigte sich, daß diese Verfahren, bei größeren HPN-Knoten nicht in endlicher Zeit zu einer vernünftigen Lösung kommen. Da der Suchraum überexponentiell mit der Knotengröße wächst, sind auch schon relativ kleine Knoten mit wenigen Bit auf diese Weise praktisch nicht

mehr optimierbar. Ein für diese Arbeit entwickelter Branch- and Bound Algorithmus zur Bestimmung der Knotentabelle mit der kleinsten kalten Information bestätigte die Ergebnisse von (Derigs, 1995). Knoten mit bis zu vier Eingangsbit ließen sich innerhalb einer Stunde optimieren, Knoten mit bis zu sechs Eingangsbit innerhalb eines Tages. Aber bei einem Knoten mit acht Eingangsbit war auch nach mehreren Tagen kaum ein Fortschritt erkennbar, so daß die Optimierung abgebrochen wurde.

Es gibt nun eine Vielzahl von Algorithmen, die mit ausgefeilten Heuristiken versuchen, in endlicher Zeit möglichst dicht an das gesuchte Optimum heranzukommen. Diese haben oft den Nachteil, daß sie algorithmisch recht komplex sind, wie z. B. der Lin-Kernighan-Algorithmus (Lin und Kernighan, 1973). Wie wir im nächsten Abschnitt sehen werden, gibt es auch eine andere Möglichkeit, HPN-Knoten zu optimieren. Zu dem vor allem in der Physik oft genutzen *Simulated Annealing* werden wir in Abschnitt 8.4.3 bei der Optimierung von HPNs mit Rückkopplungen kommen.

7.2 Stochastische Knoten

Bis jetzt waren die in den Knoten realisierten Funktionen diskrete Abbildungen. Eine bekannte Möglichkeit, die Schwierigkeiten der kombinatorischen Optimierung zu umgehen, ist den Parameterraum zu kontinuisieren (Grötschel und Padberg, 1999). Dadurch wird die Kostenfunktion oft stetig und differenzierbar, woduch man dem Bereich der kombinatorischen Optimierung entflieht und Optimierungsalgorithmen einsetzen kann, die die Stetigkeit bzw. Differenzierbarkeit der zu optimierenden Funktion voraussetzen. Diese Idee werden wir nun bei den HPN-Knoten umsetzen.

7.2.1 Erweiterung der linearen Funktionen im Produktraum

Um den Parameterraum eines HPN-Knotens zu kontinuisieren, setzen wir bei der Matrixdarstellung einer Hyperpermutation im Produktraum an. Dort ist eine Hyperpermutation h durch eine Permutationsmatrix $\boldsymbol{H}_\mathrm{h}$ realisiert, deren Zeilen- und Spaltensummen gleich eins und deren Elemente aus $\mathbb{B}$ sind.

Wir heben nun die letzte Einschränkung auf und erlauben, daß die Matrixelemente aus dem reellen Intervall $[0, 1]$ kommen dürfen, die Zeilen- und Spaltensummenbedingung gilt aber nach wir vor. Wir erlauben als Darstellung einer Hyperpermutation im Produktraum anstelle von Permutationsmatrizen nun doppelt stochastische Matrizen.

Wie ist nun eine doppelt stochastische Matrix im Produktraum zu deuten? Dazu rufen wir uns die Aussage von Satz 3 auf Seite 47 in Erinnerung: In der i-ten Spalte der Matrix $\boldsymbol{H}_\mathrm{h}$ steht der zum Zustand $\boldsymbol{y}_i = \mathrm{h}(\boldsymbol{x}_i)$ gehörende Zustandsvektor. Der Zustand $\boldsymbol{y}_i$ ist der mit h transformierte Zustand $\boldsymbol{x}_i$. Wir interpretieren nun einen Hyperpermutationsknoten der Dimension d als einen Satz von 2^d statistischen Quellen $(\mathcal{Q}, \boldsymbol{p}_i)$. Der Signalvorrat $\mathcal{Q} = \{\boldsymbol{\epsilon}_0, \ldots, \boldsymbol{\epsilon}_{2^d-1}\}$ aller Quellen ist identisch und entspricht den 2^d möglichen Ausgangsmustern des Hyperpermutationsknotens im Produktraum. Die Wahrscheinlichkeitsverteilungen $\boldsymbol{p}_i$ der Quellen sind durch die Spalten der doppelt stochastischen Matrix $\boldsymbol{H}_{\mathrm{h}^\mathrm{s}}$ gegeben, die die stochastische Hyperpermutation h^s realisiert. Die Ausgangssymbole $\boldsymbol{y}$ sind Zufallsvariablen geworden.

Die beiden „Betriebsweisen“ eines Knotens lassen sich also folgendermaßen beschreiben:

Deterministischer Knoten: Ein Eingangsmuster wird vom Knoten mit Hilfe der Funktion ind in einen Index i umgewandelt. Das an der entsprechenden Stelle der Knotentabelle stehende Ausgangsmuster wird vom Knoten ausgegeben.

Stochastischer Knoten: Ein Eingangsmuster wird vom Knoten wiederum in einen Index i gewandelt. Nun wird entsprechend der Wahrscheinlichkeitsverteilung in der i-ten Spalte der doppelt stochastischen Matrix $\boldsymbol{H}$ ein Ausgangsmuster ausgegeben. Die Elemente von $\boldsymbol{H}$ entsprechen also gerade den bedingten Wahrscheinlichkeiten $p(\boldsymbol{y}|\boldsymbol{x})$.

Der deterministische Fall ist im stochastischen Fall enthalten, dann ist nämlich die doppelt stochastische Matrix eine Permutationsmatrix.

Im Parameterraum ergibt sich folgendes Bild: In der Produktdarstellung wird eine Hyperpermutation H auf d Bit durch eine $2^d \times 2^d$ Permutationsmatrix $\boldsymbol{H}$ repräsentiert. Fassen wir die $2^d \cdot 2^d$ Elemente der Permutationsmatrix in einem Parametervektor zusammen, entsprechen die Permutationsmatrizen Punkten in einem $2^{2\,d}$ dimensionalen Parameterraum, wobei sich die Permutationsmatrizen gerade auf den Ecken des $2^{2\,d}$ dimensionalen Einheitswürfels befinden. Es sind nur diejenigen Ecken des Würfels erlaubt, die im Produktraum Permutationsmatrizen entsprechen, d. h. die Zeilen- und Spaltensummen müssen gleich eins und die Matrixelemente aus $\mathbb{B}$ sein. Durch die Erweiterung der Knotenmatrizen auf doppelt stochastische Matrizen erweitern wir die Menge der zulässigen Parametervektoren von den Ecken den Einheitswürfels auf das Innere des Einheitswürfels. Es sind allerdings nur diejenigen Bereiche innerhalb des Einheitswürfels erlaubt, deren zugeordnete Matrizen die Bedingungen einer doppelt stochastischen Matrix erfüllen, die Zeilen- und Spaltensummen müssen also gleich eins sein.

7.2.2 Transformation eines Gewichtsvektors mit einem stochastischen Knoten

Bei einem deterministischen Knoten ergibt sich nach Satz 3 auf Seite 47 der transformierte Zustandsvektor $\boldsymbol{\kappa} = \boldsymbol{H}_{\mathrm{h}} \cdot \boldsymbol{\chi}$ durch Matrixmultiplikation des Zustandsvektors am Knoteneingang mit der Hyperpermutationsmatrix $\boldsymbol{H}_{\mathrm{h}}$. Bei einem stochastischen Knoten können dagegen zu demselben Eingangszustand verschiedene Ausgangszustände entstehen, je nachdem, welches Ausgangssymbol im Inneren des Knotens zufällig ausgewählt wurde. Was uns daher interessiert ist der Erwartungswert $\langle \boldsymbol{\kappa} \rangle$ des Ausgangszustands.

Satz 7. *Sei $\boldsymbol{H}_{\mathrm{h}}$ mit $\boldsymbol{\kappa} = \boldsymbol{H}_{\mathrm{h^s}} \cdot \boldsymbol{\chi}$ eine stochastische Hyperpermutation eines Knoten v der Dimension $d = \dim(v)$. Dann ist der Erwartungswert $\langle \boldsymbol{\kappa} \rangle$ des Ausgangszustands bei gegebenem Eingangszustand $\boldsymbol{\epsilon}_j$ durch*

$$\langle \boldsymbol{\kappa} \rangle = \boldsymbol{H}_{\mathrm{h^s}} \cdot \boldsymbol{\epsilon}_j$$

gegeben.

Beweis. Der Erwartungswert einer Zufallsvariablen X ist durch $\langle X \rangle = \sum_i p(X = x_i)\, x_i$ definiert. Für unseren Fall bedeutet dies mit dem Eingangsvektor $\boldsymbol{\epsilon}_j$:

$$\begin{aligned} \langle \boldsymbol{\kappa} \rangle &= \sum_{i=0}^{2^d-1} p_j(\boldsymbol{\kappa} = \boldsymbol{\epsilon}_i)\, \boldsymbol{\epsilon}_i = \sum_{i=0}^{2^d-1} [\boldsymbol{H}_{\mathrm{h^s}}]_{i,j}\, \boldsymbol{\epsilon}_i \\ &= \boldsymbol{H}_{\mathrm{h^s}} \cdot \boldsymbol{\epsilon}_j \end{aligned}$$

Der zu dem Eingangsvektor $\boldsymbol{\epsilon}_j$ gehörende Erwartungswert des Ausgangsvektors ist also gerade die j-te Spalte der Matrix $\boldsymbol{H}_{\mathrm{h}}$. □

Wie sieht der Erwartungswert des Ausgangsvektors aus, wenn wir nicht nur einen Eingangsvektor, sondern einen kompletten Gewichtsvektor transformieren wollen? Bei einem deterministischen Knoten ergibt sich nach Satz 4 auf Seite 50 der transformierte Gewichtsvektor $\boldsymbol{w}'$

eines Ausgangsbündels eindeutig aus dem Gewichtsvektor $\boldsymbol{w}$ des Eingangsbündels und der Hyperpermutation des Knotens. Bei einem stochastischen Knoten können dagegen zu demselben Eingangsgewichtsvektor verschiedene Ausgangsgewichtsvektoren entstehen. Uns interessiert daher der Erwartungswert $\langle \boldsymbol{w}' \rangle$ des transformierten Gewichtsvektors.

Satz 8. *Gegeben sei der Gewichtsvektor $\boldsymbol{w}$ einer Zustandsfolge. Transformieren wir die Zustandsfolge mit der stochastischen Hyperpermutation $\boldsymbol{H}_{\mathrm{h^s}}$, so ist der Erwartungswert $\langle \boldsymbol{w}' \rangle$ des transformierten Gewichtsvektors gegeben durch*

$$\langle \boldsymbol{w}' \rangle = \boldsymbol{H}_{\mathrm{h^s}} \cdot \boldsymbol{w} \ .$$

Beweis. Der transformierte Gewichtsvektor $\boldsymbol{w}'$ einer Zustandsfolge der Dimension d läßt sich im Produktraum als gewichtete Summe der Erwartungswerte der transformierten Leitungsvektoren $\langle \boldsymbol{\epsilon}_i' \rangle = \boldsymbol{H}_{\mathrm{h^s}} \cdot \boldsymbol{\epsilon}_i$ schreiben,

$$\begin{aligned} \langle \boldsymbol{w}' \rangle &= \left\langle \sum_{i=0}^{2^d-1} w_i \, \boldsymbol{\epsilon}_i' \right\rangle = \sum_{i=0}^{2^d-1} w_i \, \langle \boldsymbol{\epsilon}_i' \rangle = \sum_{i=0}^{2^d-1} w_i \, \boldsymbol{H}_{\mathrm{h}} \cdot \boldsymbol{\epsilon}_i = \boldsymbol{H}_{\mathrm{h}} \cdot \left(\sum_{i=0}^{2^d-1} w_i \, \boldsymbol{\epsilon}_i \right) \\ &= \boldsymbol{H}_{\mathrm{h}} \cdot \boldsymbol{w} \ . \end{aligned}$$

□

Folgerung 9. Wir erhalten den Erwartungswert des Gewichtsvektors am Knotenausgang aus dem Gewichtsvektor am Knoteneingang durch die Multiplikation des Gewichtsvektors am Knoteneingang mit der Matrix $\boldsymbol{H}_{\mathrm{h^s}}$. Wir können daher die Auswirkung einer Änderung von $\boldsymbol{H}_{\mathrm{h^s}}$ berechnen, *ohne* erneut die Zustandsfolge auswerten zu müssen. Die Zeit zur Berechnung von $\langle \boldsymbol{w}' \rangle = \boldsymbol{H}_{\mathrm{h}} \cdot \boldsymbol{w}$ ist also unabhängig von der Größe des Trainingsdatensatzes.

7.2.3 Berechnung der statistischen Größen

Wie berechnen wir nun die statistischen Größen Diversität, Information und Redundanz bei einem stochastischen Knoten? Alle diese statistischen Größen lassen sich aus ihrer Matrixdarstellung gewinnen. Die Matrixdarstellungen sind:

$$\begin{aligned} \boldsymbol{D} &= \boldsymbol{w} \cdot \boldsymbol{w}^{\mathrm{T}} \\ \boldsymbol{R} &= \sum_k \boldsymbol{w}_k \cdot \boldsymbol{w}_k^{\mathrm{T}} \\ \boldsymbol{I} &= \boldsymbol{D} - \boldsymbol{R} \end{aligned}$$

Nun sind die Gewichtsvektoren $\boldsymbol{w}$ aus einer Sequenz von Zufallsvektoren entstanden. Diese Zufallsvektoren sind stochastisch unabhängig, da sie zu verschiedenen Zeiten generiert worden sind, pro Zeitschritt wurde ein neuer zufälliger Ausgangszustand des Knotens erzeugt. Daher gilt für den Erwartungswert $\langle \boldsymbol{w} \cdot \boldsymbol{w}^{\mathrm{T}} \rangle = \langle \boldsymbol{w} \rangle \cdot \langle \boldsymbol{w}^{\mathrm{T}} \rangle$ und somit für die statistischen Größen:

$$\begin{aligned} \langle \boldsymbol{D} \rangle &= \langle \boldsymbol{w} \rangle \cdot \langle \boldsymbol{w}^{\mathrm{T}} \rangle \\ \langle \boldsymbol{R} \rangle &= \sum_k \langle \boldsymbol{w}_k \rangle \cdot \langle \boldsymbol{w}_k^{\mathrm{T}} \rangle \\ \langle \boldsymbol{I} \rangle &= \langle \boldsymbol{D} \rangle - \langle \boldsymbol{R} \rangle \end{aligned}$$

7.2.4 Die Kostenfunktion im Reellen

Die reelle Kostenfunktion unterscheidet sich von der diskreten Kostenfunktion dadurch, daß nun ein Teil des Parameterraums zugelassen ist, der bei der diskreten Kostenfunktion verboten war, und zwar das Innere des Einheitswürfels des Parameterraums. Diese Erweiterung ermöglicht es, Verfahren der kontinuierlichen Optimierung auf unser Problem anzuwenden.

Hier tritt nun ein wichtiger Unterschied zwischen den beiden in Abschnitt 7.1 vorgestellen Kostenfunktionen zu Tage. Die kalte Information wird durch die Kontinuisierung der Parameter stetig differenzierbar, da sie eine quadratische Form der Parameter ist: $I_\mathrm{k} = \mathrm{Sp}\,(\boldsymbol{I}) = \boldsymbol{w}^\mathrm{T} \cdot \boldsymbol{w} - \sum_{k=0}^{K-1} \boldsymbol{w}_k^\mathrm{T} \cdot \boldsymbol{w}_k$. Die über eine Kostenmatrix definierte Kostenfunktion werten wir dagegen mit Hilfe der klassenspezifischen Rückschlußwahrscheinlichkeiten aus. Es wird jeweils zugunsten derjenigen Klasse entschieden, die die höchste Rückschlußwahrscheinlichkeit hatte. Dies ist aber eine Maximumentscheidung, die sich sprunghaft in Abhängigkeit der Parameter ändert. Eine so definierte Kostenfunktion ist nicht stetig bzw. differenzierbar. Für die kontinuierliche Optimierung ist eine solche Kostenfunktion uninteressant, wenn wir z. B. ein gradientenbasiertes Optimierungsverfahren einsetzen wollen, da sie trotz der Kontinuität der Parameter selbst nicht stetig differenzierbar wird.

Das Ziel einer kontinuierlichen Optimierung muß sein, eine *deterministische* Lösung zu finden, die die Kostenfunktion minimiert. Letztendlich soll das Hyperpermutationsnetzwerk in Hareware implementiert werden, denn dort wollen wir ja gerade die Vorteile eines RAM-basierten Netzwerks nutzen und ohne Arithmetik auskommen. Der Weg über stochastische Knoten ist also nur zur Optimierung „erlaubt“.

Eine stochastische Lösung läßt sich in eine deterministische Lösung umwandeln, indem man in jeder Spalte der doppelt stochastischen Matrix die vorhandene Wahrscheinlichkeitsverteilung durch diejenige Wahrscheinlichkeitsverteilung ersetzt, die das häufigste Ausgangssymbol mit der Wahrscheinlichkeit $p = 1$ emittiert. War die stochastische Lösung in der Nähe einer deterministischen Lösung, ändert sich die Kostenfunktion durch diese Maßnahme wegen ihrer Stetigkeit nur wenig.

7.2.5 Die Nettoinformation

Wir wollen nun zusätzlich erreichen, daß der Zustandsraum am Ausgang eines Knotens möglichst effizient genutzt wird. Dazu müssen wir neben der Maximierung der heißen Information auch noch die heiße Redundanz minimieren. Um sich das klarzumachen, betrachten wir in Abbildung 7.1 die beiden Statistiken zweier fiktiver Knoten, von denen drei Ausgangszustände genutzt werden.

Knoten A	Klasse 0	Klasse 1
Zustand 0	n_1	0
Zustand 1	0	n_2
Zustand 2	n_3	0

Knoten B	Klasse 0	Klasse 1
Zustand 0	$n_1 + n_3$	0
Zustand 1	0	n_2
Zustand 2	0	0

Abbildung 7.1: Statistiken zweier Drei-Zustands-Knoten. Die Gesamtzahl der Daten ist $n_1 + n_2 + n_3$. Knoten A verwendet Endzustand 2, Knoten B verwendet Endzustand 2 dagegen nicht.

Beide Knoten sind im Sinne der heißen Information gleich gut, wie die folgende kurze

Rechnung zeigt:

$$\begin{aligned} I_\mathrm{h}(\mathrm{A}) &= D_\mathrm{h}(\mathrm{A}) - R_\mathrm{h}(\mathrm{A}) = 2\,(n_1 n_2 + n_1 n_3 + n_2 n_3) - 2\,n_1 n_3 \\ &= 2\,(n_1 n_2 + n_2 n_3) = 2\,(n_1 + n_3)\,n_2 \\ I_\mathrm{h}(\mathrm{B}) &= D_\mathrm{h}(\mathrm{B}) - R_\mathrm{h}(\mathrm{B}) = 2\,(n_1 + n_3)\,n_2 - 0 \\ &= 2\,(n_1 + n_3)\,n_2 = I_\mathrm{h}(\mathrm{A}) \end{aligned} \tag{7.2}$$

Der Knoten B benutzt aber den Endzustand 2 nicht, er geht also mit dem zur Verfügung stehenden Platz im Zustandsraum effizienter um. Dies drückt sich quantitativ darin aus, daß die heiße Redundanz der beiden Knoten *nicht* gleich ist: $R_\mathrm{h}(\mathrm{A}) = 2\,n_1 n_3 > R_\mathrm{h}(\mathrm{B}) = 0$. Je größer bei gleicher heißer Information die heiße Redundanz ist, um so verschwenderischer geht der Knoten mit dem Platz im Zustandsraum um.

Um die effiziente Nutzung des Zustandsraums zu belohnen, wählen wir eine Gütefunktion $G(H)$, die die heiße Information I_h positiv und die heiße Redundanz R_h, gewichtet mit einem Faktor $k' \geq 0$, negativ bewertet:

$$\begin{aligned} G(H) &= I_\mathrm{h} - k' \cdot R_\mathrm{h} = D_\mathrm{h} - R_\mathrm{h} - k' \cdot R_\mathrm{h} \\ &= D_\mathrm{h} - \underbrace{(1 + k')}_{k} R_\mathrm{h} = D_\mathrm{h} - k \cdot R_\mathrm{h} \end{aligned} \tag{7.3}$$

Wir möchten, daß die Gütefunktion null ist, wenn die Rückschlußwahrscheinlichkeiten auf die verschiedenen Klassen in allen Zuständen gleich sind. Dann ändert sich nämlich durch die Kenntnis des Endzustandes nichts, und wir könnten uns genauso gut à priori für die Klasse mit der höchsten Rückschlußwahrscheinlichkeit entscheiden. In dieser Situation ist die Gütefunktion null, wenn wir für die Konstante k den Quotienten $k = D_\mathrm{t}/R_\mathrm{t}$ wählen (Beweis siehe Anhang A.3 auf Seite 183). Die erhaltene Gütefunktion nennen wir *Nettoinformation* I_n (Oberländer):

$$I_\mathrm{n} = D_\mathrm{h} - \frac{D_\mathrm{t}}{R_\mathrm{t}} R_\mathrm{h} \tag{7.4}$$

7.2.6 Gleichungs und Ungleichungsnebenbedingungen

Bei diskreten Knoten war es klar, daß wir es in den Knotentabellen mit Bitstrings und im Produktraum mit Permutationsmatrizen zu tun haben. Gehen wir nun zur kontinuierlichen Optimierung über, müssen wir berücksichtigen, daß unsere Parameter bestimmten Randbedingungen unterliegen. Die zu optimierenden Parameter sind die Elemente h_{ij} der Hyperpermutationsmatrix $\boldsymbol{H}$ im Produktraum. Die Spalten dieser Matrix sind Wahrscheinlichkeitsverteilungen, was zwei Sätze von Nebenbedingungen festlegt. Außerdem soll die Matrix bei einer Gleichverteilung am Knoteneingang keine Ausgangssymbole am Knotenausgang bevorzugen. Zusammen erhalten wir:

1. Da es sich bei einer Spalte von $\boldsymbol{H}$ um eine Wahrscheinlichkeitsverteilung handelt, müssen alle Spaltensummen gleich eins sein:

$$\sum_{i=0}^{2^d-1} \boldsymbol{H}_{ij} = 1\,; \qquad j \in [0, 2^d - 1]$$

2. Aus demselben Grund müssen alle Elemente von $\boldsymbol{H}$ die Ungleichungen

$$\boldsymbol{H}_{ij} \geq 0 \qquad \text{und} \qquad \boldsymbol{H}_{ij} \leq 1$$

erfüllen. Wie wir sehen, ergibt sich aus $\boldsymbol{H}_{ij} \geq 0$ und den Nebenbedingungen aus 1., daß $\boldsymbol{H}_{ij} \leq 1$. Diese Nebenbedingungen müssen also nicht explizit berücksichtigt werden.

3. Alle Ausgangsmuster sollen bei einer Gleichverteilung am Knoteneingang im Mittel am Knotenausgang gleich häufig vorkommen (wie bei einer Permutationsmatrix), was für die Zeilensummen

$$\sum_{j=0}^{2^d-1} \boldsymbol{H}_{ij} = 1 \; ; \qquad i \in [0, 2^d - 1]$$

bedeutet. Optimieren wir nicht nur $\boldsymbol{H}$, sondern gleich die $2^{d'} \times 2^d$-Matrix $\bar{\boldsymbol{H}} = \boldsymbol{\Phi} \cdot \boldsymbol{H}$, ändert sich diese Nebenbedingung durch die Projektion auf $2^{d'}$ Ausgangsbit:

$$\sum_{j=0}^{2^d-1} \boldsymbol{H}_{ij} = 2^{d-d'} \; ; \qquad i \in [0, 2^{d'} - 1]$$

Es gibt nun zwei Möglichkeiten, dieses Optimierungsproblem zu lösen. Man kann versuchen, die Lösungen analytisch zu bestimmen. Da dies in der Praxis meist nicht möglich ist, gehen wir auf die analytische Optimierung nur sehr kurz ein. Die andere Möglichkeit ist, numerische Verfahren zur Optimierung einzusetzen. Diese Verfahren nähern sich der optimalen Lösung meistens iterativ an.

7.2.7 Analytische kontinuierliche Optimierung

Nachdem die Kostenfunktion nun im Reellen definiert ist, kann das Optimum im Prinzip auf analytischem Wege gefunden werden (Übersicht in Papageorgiou, 1991). Wir skizzieren hier nur sehr kurz, wie man prinzipiell vorgeht. Für die Praxis ist in unserem Fall die analytische Optimierung allerdings nicht geeignet, weswegen weitere Ausführungen zur analytischen Optimierung mit Nebenbedingungen in den Anhang A.4 auf Seite 185 ausgelagert wurden.

Technisch gesprochen haben wir es hier mit einem sogenannten statischen Optimierungsproblem einer Funktion F mehrerer Variabler unter Gleichungs- und Ungleichungsnebenbedingungen zu tun. Notwenige Bedingung für ein Minimum ist, daß der Gradient der Funktion verschwindet. Wegen der Nebenbedingungen müssen wir aber nicht den Gradient von F selbst sondern von

$$L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = f(\boldsymbol{x}) + \boldsymbol{\lambda}^{\mathrm{T}} \, \boldsymbol{c}(\boldsymbol{x}) + \boldsymbol{\mu}^{\mathrm{T}} \, \boldsymbol{h}(\boldsymbol{x})$$

auswerten. Dabei ist $\boldsymbol{\lambda}$ der Vektor der *Lagrange-Multiplikatoren*, $\boldsymbol{c}(\boldsymbol{x})$ sind die Gleichungsnebenbedingungen in Form eines Vektors, $\boldsymbol{\mu}$ ist der Vektor der *Kuhn-Tucker-Multiplikatoren* und $\boldsymbol{h}(\boldsymbol{x})$ sind die Ungleichungsnebenbedingungen in Form eines Vektors. Nun müssen mit Hilfe der notwendigen Bedingungen mögliche Kandidaten für ein Minimum bestimmt werden und dann mit den hinreichenden Bedingungen die Kandidaten ausgesucht werden, die tatsächlich ein Minimum sind. Wie in Anhang A.4 beschrieben, ist dieser Prozeß für eine Funktion mit vielen Variablen und Nebenbedingungen sehr aufwendig und für unser Problem nicht praktikabel.

7.3 Numerische kontinuierliche Optimierung

Die numerische Minimierung bzw. Maximierung reeller Funktionen ist ein eigenständiges Forschungsgebiet mit etlichen Spezialgebieten (siehe Papageorgiou, 1991; Nemhauser et al., 1989).

Meistens versteht man unter „Minimierung einer Funktion“ das Finden eines lokalen Minimums der Funktion. Obwohl oft eigentlich das globale Minimum einer Funktion gesucht ist, versucht man meistens, das Optimierungsziel mit der Suche nach einem lokalen Minimum zu erreichen, da es bei komplexeren Optimierungsproblemen selten einen praktikablen globalen Minimierungsalgorithmus gibt.

Die hier vorgestellten Verfahren kombinieren jeweils eine Richtungswahl mit einer Linienminimierung. Das heißt, im Parameterraum wird erst eine Richtung ausgewählt, dann wird in der gewählten Richtung eine eindimensionale Minimierung durchgeführt:

1. Wähle einen Startpunkt $\boldsymbol{x}_0$ und setze $i = 0$.
2. Bestimme eine Suchrichtung (Abstiegsrichtung) $\boldsymbol{s}_i$.
3. Berechne die skalare Schrittweite α_i mittels einer Linienminimierungs, d. h. durch Lösen des Minimierungsproblems

$$\min_{\alpha_i > 0} f(\boldsymbol{x}_i + \alpha_i \, \boldsymbol{s}_i) \; .$$

   Setze dann $\boldsymbol{x}_{i+1} = \boldsymbol{x}_i + \alpha_i \, \boldsymbol{s}_i$.
4. Wenn ein Abbruchkriterium erfüllt ist, stoppe, sonst setze $i = i + 1$ und gehe nach 2.

Bei der numerischen Optimierung kontinuierlicher Kostenfunktionen unterscheidet man hauptsächlich zwei Varianten. Zum einen gibt es gradientenbasierte Verfahren, die bei jedem Minimierungsschritt zur Bestimmung der Abstiegsrichtung den Gradienten der n-dimensionalen Kostenfunktion benötigen. Zum anderen werden Verfahren verwendet, die während der Minimierung auf die Berechnung des Gradienten ganz verzichten (siehe Abschnitt 8.4 auf Seite 112).

7.3.1 Gradientenbasierte Verfahren

Hat das Minimierungsproblem keine Nebenbedingungen, ist die Bestimmung einer Suchrichtung relativ einfach: Die Suchrichtung $\boldsymbol{s}_i$ und der Gradient $\boldsymbol{g}_i := \boldsymbol{g}(\boldsymbol{x}_i)$ der Funktion müssen einen stumpfen Winkel bilden, damit $\boldsymbol{s}_i$ tatsächlich eine *Abstiegsrichtung* ist. Es muß also ${\boldsymbol{s}_i}^{\mathrm{T}} \boldsymbol{g}_i < 0$ gelten.

Zur tatsächlichen Wahl der Suchrichtung gibt es verschiedene Verfahren, oft wird der sogenannte *steilste Abstieg* oder auch *Gradientenabstieg* verwendet. Hier wird die Suchrichtung durch $\boldsymbol{s}_i = -\boldsymbol{g}_i$ festgelegt. Diese Wahl führt relativ schnell in die Nähe eines lokalen Minimums, konvergiert dann aber recht langsam. Des weiteren werden *Newton-Verfahren* und *Quasi-Newton-Verfahren* verwendet, die in der Nähe eines lokalen Minimums relativ schnell konvergieren, dafür nicht so schnell in die Nähe eines Minimums kommen.

Ein nicht zu vernachlässigender Punkt ist der für den Gradienten benötigte Rechenaufwand, der wegen $g_i \approx (f(\boldsymbol{x} + \Delta\boldsymbol{x}_i) - f(\boldsymbol{x} - \Delta\boldsymbol{x}_i))/2\delta_i$ bei n Dimensionen $2n$ Funktionsaufrufe erfordert. Kann man den Gradienten analytisch berechnen, kommt man teilweise auch mit weniger Rechenaufwand zum Ziel, für große n ist aber auch die analytische Gradientenberechnung aufwendig.

7.3.1.1 Der reduzierte Gradient

Kommen nun wie in unserem Fall Gleichungs- und Ungleichungsnebenbedingungen dazu, sind nicht mehr alle *möglichen* Suchrichtungen auch *zulässige* Suchrichtungen. Die Gleichungsbe-

dingungen verringern die Dimension des Suchraums, es ist nur noch ein Unterraum des n-dimensionalen Parameterraums zugelassen. Die Ungleichungsbedingungen schränken den zulässigen Bereich innerhalb des Unterraums weiter ein. Um diese Einschränkungen bei der Minimierung zu berücksichtigen, können wir sowohl bei der zu minimierenden Funktion selbst als auch bei der Wahl der zulässigen Suchrichtung ansetzen.

Im ersten Fall ändert man die Funktion durch die Addition eines sogenannten *Strafterms* so ab, daß die Funktion an nicht zulässigen Punkten des Suchraums extrem hohe Werte annimmt und dort kein Minimum gefunden wird. Der Strafterm bewertet die Abweichung eines Punktes im Parameterraum von den Randbedingungen. Die Methode der Straffunktion besticht durch ihre prinzipielle Einfachheit, allerdings hat sie in der numerischen Praxis auch ihre Schwierigkeiten. Zum Beispiel muß ein quadratischer Strafterm nahe des Minimums sehr stark gewichtet werden, was zur Folge haben kann, daß die Hesse-Matrix der Funktion mit Strafterm fast singulär wird. Wählt man dagegen als Strafterm den Absolutbetrag der Abweichung von den Nebenbedingungen, fällt das Problem der fast singulären Hesse-Matrix weg. Dafür ist aber die Funktion mit Strafterm nun auch im Minimum der Funktion nicht mehr differenzierbar (siehe Nemhauser et al., 1989, Seite 181 ff.).

Wir gehen hier den zweiten Weg und lassen nur noch Richtungen im Suchraum zu, die in den zulässigen Unterraum „hineinzeigen“. Die Gleichungsbedingungen werden dazu in den Gradienten integriert, der dann *reduzierter Gradient* genannt wird. Hat man eine zulässige Suchrichtung ausgewählt, muß man bei der anschließenden Linienminimierung nur noch die Einhaltung der Ungleichungsbedingungen berücksichtigen. Es gibt ausgefeilte Programmpakete, die die Minimierung einer Funktion und Gleichungs- und Ungleichungsnebenbedingungen beherrschen.

7.3.1.2 Der Gradient der Kostenfunktion

Um ein gradientenbasiertes Verfahren zu verwenden, ist es vorteilhaft, wenn der Gradient der Funktion nicht numerisch ermittelt werden muß, sondern analytisch zur Verfügung gestellt wird. Dazu berechnen wir den Gradient unserer Kostenfunktion, der kalten Information.

Um die Berechnung numerisch effizient zu gestalten, bauen wir die $r \times c$ – Hyperpermutationsmatrix $\boldsymbol{H}$ aus r Zeilenvektoren $\boldsymbol{s}_i^{\mathrm{T}}$ der Länge c auf:

$$\boldsymbol{H} = \begin{pmatrix} \boldsymbol{s}_0^{\mathrm{T}} \\ \boldsymbol{s}_1^{\mathrm{T}} \\ \vdots \\ \boldsymbol{s}_{r-1}^{\mathrm{T}} \end{pmatrix}$$

Dann läßt sich die kalte Information I_{k} als

$$I_{\mathrm{k}} = \mathrm{Sp}\left(\boldsymbol{H}\, \boldsymbol{I}_{\mathrm{k}}\, \boldsymbol{H}^{\mathrm{T}}\right) = \sum_i \boldsymbol{s}_i^{\mathrm{T}}\, \boldsymbol{I}_{\mathrm{k}}\, \boldsymbol{s}_i$$

schreiben. Nun gilt für eine Matrix $\boldsymbol{A}$ und einen Vektor $\boldsymbol{x}$

$$\frac{\partial\, \boldsymbol{x}^{\mathrm{T}} \boldsymbol{A} \boldsymbol{x}}{\partial \boldsymbol{x}} = \boldsymbol{A}\, \boldsymbol{x} + \boldsymbol{A}^{\mathrm{T}}\, \boldsymbol{x}\ .$$

Damit ist der Gradient von I_{k} bzgl. $\boldsymbol{s}_j$

$$\frac{\partial}{\partial \boldsymbol{s}_j} I_{\mathrm{k}} = \sum_i \frac{\partial}{\partial \boldsymbol{s}_j} \boldsymbol{s}_i^{\mathrm{T}}\, \boldsymbol{I}_{\mathrm{k}}\, \boldsymbol{s}_i = \sum_i 2\, \boldsymbol{I}_{\mathrm{k}}\, \boldsymbol{s}_j \delta_{ij} = 2\, \boldsymbol{I}_{\mathrm{k}}\, \boldsymbol{s}_j\ , \tag{7.5}$$

da $\boldsymbol{I}_\mathrm{k}$ symmetrisch ist. Bei der Berechnung von I_k können wir wegen (7.5) den Gradiententerm $\frac{\partial}{\partial \boldsymbol{s}_j} I_\mathrm{k} = 2\,\boldsymbol{I}_\mathrm{k}\,\boldsymbol{s}_j$ wiederverwenden, um so die Rechnung effizient durchzuführen.

$$I_\mathrm{k} = \sum_i \boldsymbol{s}_i^\mathrm{T}\,\boldsymbol{I}_\mathrm{k}\,\boldsymbol{s}_i = \frac{1}{2}\sum_i \boldsymbol{s}_i^\mathrm{T}\,\partial_{\boldsymbol{s}_i} I_\mathrm{k}\ .$$

7.3.1.3 Numerische Minimierung mit dem Programmpaket MINOS

Um die Leistungsfähigkeit eines klassischen gradientenbasierten Minimierungsverfahren für unser Minimierungsproblem zu testen, wurde das Programmpaket MINOS (Murtagh und Saunders, 1995) zu Hilfe genommen. MINOS stellt eine mächtige Fortran-Bibliothek zur Verfügung, um Minimierungsprobleme mit Gleichungs- und Ungleichungsnebenbedingungen zu lösen. Um die Minimierung durchzuführen, wird MINOS sowohl die Kostenfunktion und ihr Gradient als auch die Nebenbedingungen aus Abschnitt 7.2.6 als Fortran-Code übergeben. MINOS übernimmt dann vollständig die weitere Minimierung.

Wie gut ist MINOS in unserem speziellen Fall? Wie zu erwarten war, findet MINOS Minima der Kostenfunktion mit hoher Zuverlässigkeit und Genauigkeit. MINOS findet aber prinzipbedingt nur *lokale* Minima. Da die Kostenfunktion lokale Minima enthält, terminierte MINOS teilweise schon, bevor das globale Minimum erreicht wurde. Die Minimierung mit MINOS hatte auch den Nachteil, daß sie wegen der verwendeten aufwendigen Algorithmen lange dauert (Stunden). Wie wir gleich sehen werden, gibt es eine auf unser Optimierungsproblem zugeschnittene um Größenordnungen schnellere Alternative, die Minima genauso zuverlässig findet. Da außerdem bei rückgekoppelten HPN lokale Minima verstärkt auftreten, wurde die Minimierung mit gradientenbasierten Minimierungsverfahren und die Minimierung mit MINOS nicht weiter verfolgt.

7.3.2 Der PAP-Algorithmus

Der *PAP-Algorithmus* (PAP steht für „probabilistic augmenting path“) von Oberländer ist ein Verfahren, um das eben besprochene Minimierungsproblem für rückkopplungsfreie HPNs elegant und vor allem schnell zu lösen. Wir werden hier die Grundideen des PAP-Algorithmus vorstellen, auf denen die Erweiterung des Algorithmus auf rückgekoppelte HPN in Abschnitt 8.3 beruht. Eine detaillierte Beschreibung des PAP-Algorithmus befindet sich in Oberländer (1997).

Rekapitulieren wir nochmal kurz das zu lösende Optimierungsproblem: Wir wollen die Elemente einer $n \times m$-Matrix $\bar{\boldsymbol{H}}$ finden, die unsere Kostenfunktion minimieren. Dabei gelten die Nebenbedingungen aus Abschnitt 7.2.6. Die Hauptschwierigkeit ist, die Nebenbedingungen in ein allgemeines Minimierungsverfahren effizient einzubinden.

In Abschnitt 7.3 haben wir vorgestellt, welche Struktur ein typischer mehrdimensionaler Minimierungsalgorithmus, der Linienminimierungen einsetzt, hat. Der PAP-Algorithmus hat nun die gleiche Grundstruktur, er setzt aber an entscheidenden Stellen das Wissen über unser spezielles Minimierungsproblem ein. Wir gehen nun den PAP-Algorithmus Schritt für Schritt durch:

1. Wähle einen Startpunkt $\bar{\boldsymbol{H}}_0$ und setze $i = 0$:
   Ein Punkt im Parameterraum ist in unserem Fall eine $n \times m$-Matrix $\bar{\boldsymbol{H}}$, der Parameterraum hat damit $n \cdot m$ Dimensionen. Jede Matrix $\bar{\boldsymbol{H}}$, die die Nebenbedingungen aus Abschnitt 7.2.6 erfüllt, nennen wir *zulässig* und kann als Startpunkt gewählt werden. Um aber nicht schon zu Beginn der Optimierung irgendwelche stochastischen Matrizen vor anderen zu bevorzugen, ist es zweckmäßig, als Startmatrix die Matrix $[\bar{\boldsymbol{H}}_0]_{ij} = 1/n$ zu

wählen. Diese Matrix hat in jeder Spalte als Wahrscheinlichkeitsverteilung die Gleichverteilung.

2. Bestimme eine Suchrichtung (Abstiegsrichtung) $\boldsymbol{M}_i$:
   In unserem Fall wird die Suchrichtung nicht durch einen Vektor, sondern durch eine Matrix $\boldsymbol{M}_i$ bestimmt. Diese Matrix muß so beschaffen sein, daß die bei der nachfolgenden Linienminimierung ausgewerteten Punkte $\bar{\boldsymbol{H}}_i + \alpha_i \boldsymbol{M}_i$ zulässig sind. Da aber nach Voraussetzung $\bar{\boldsymbol{H}}_i$ zulässig ist, ist $\bar{\boldsymbol{H}}_i + \alpha_i \boldsymbol{M}_i$ genau dann zulässig, falls $\boldsymbol{M}_i$ die Zeilen- und Spaltensummen Null hat. Dann sind nämlich die Zeilen- und Spaltensummen von $\bar{\boldsymbol{H}}_i + \alpha_i \boldsymbol{M}_i$ und $\bar{\boldsymbol{H}}_i$ gleich. Eine einfache Klasse von Matrizen, die diese Bedingung erfüllt, sieht folgendermaßen aus:

$$\boldsymbol{M}_k = \begin{pmatrix} 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\ \vdots & & \vdots & & \vdots & & \vdots \\ \vdots & \cdots & 1 & \cdots & -1 & \cdots & \vdots \\ \vdots & & \vdots & & \vdots & & \vdots \\ \vdots & \cdots & -1 & \cdots & 1 & \cdots & \vdots \\ \vdots & & \vdots & & \vdots & & \vdots \\ 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \end{pmatrix}$$

   Alle bis auf vier Elemente dieser Matrizen sind Null. Die vier Elemente 1 bzw. -1 sind so angeordnet, daß die Zeilen- und Spaltensummen Null sind. Jeder dieser Matrizen kommt also für uns als Suchrichtung in Frage.

   Welche Matrix wählt man aber nun bei einer Iteration als nächste Suchrichtung? Wir könnten versuchen, ähnlich dem reduzierten Gradienten, diejenige Matrix $\boldsymbol{M}_i$ zu finden, die die größte Verringerung der Kostenfunktion verspricht. Ebenso könnten wir alle möglichen Matrizen $\boldsymbol{M}_k$ systematisch generieren. Wir machen es uns so einfach wie möglich und wählen bei jeder Iteration aus der Menge der möglichen Matrizen eine zufällig aus. Diese sogenannte *stochastische Richtungswahl* hat neben dem fast verschwindenden Rechenaufwand den Vorteil, daß sie prinzipiell in der Lage ist, lokalen Minima zu entkommen (siehe Abschnitt 8.4.2.1) und keine systematische Bevorzugung bestimmter Matrizen $\boldsymbol{M}_i$ verursacht.

3. Berechne die skalare Schrittweite α_i mittels einer Linienminimierung, d. h. durch Lösen des Minimierungsproblems

$$\min_{\alpha_i} f(\bar{\boldsymbol{H}}_i + \alpha_i \boldsymbol{M}_i) \ .$$

   Setze dann $\bar{\boldsymbol{H}}_{i+1} = \bar{\boldsymbol{H}}_i + \alpha_i \boldsymbol{M}_i$:
   Zunächst müssen wir berücksichtigen, daß kein Element von $\bar{\boldsymbol{H}}_{i+1}$ das Intervall $[0, 1]$ verlassen darf. Daraus resultiert für α, daß die α-Werte innerhalb eines zulässigen Intervalls $\alpha \in [\alpha_{\min}, \alpha_{\max}]$ liegen müssen.

   Um die Linienminimierung zu lösen müssen wir wissen, wie sich die Kostenfunktion f in Abhängigkeit von α ändert. Die transformierte kalte Information I'_{k} am Ausgang eines Knotens ist nach Gleichung (7.1) auf Seite 88

$$I'_{\mathrm{k}}(\boldsymbol{I}') = \mathrm{Sp}\left(\boldsymbol{I}'\right) = \mathrm{Sp}\left(\bar{\boldsymbol{H}}\, \boldsymbol{I}\, \bar{\boldsymbol{H}}^{\mathrm{T}}\right) \ .$$

Die Änderung $\Delta I'_{\mathrm{k}}(\alpha)$ der kalten Information in Abhängigkeit von α wird damit:

$$\begin{aligned}\Delta I'_{\mathrm{k}}(\alpha) &= I'_{\mathrm{k}}(\alpha) - I'_{\mathrm{k}}(0) \\ &= \mathrm{Sp}\left((\bar{\boldsymbol{H}}_i + \alpha_i \boldsymbol{M}_i)\, \boldsymbol{I}\, (\bar{\boldsymbol{H}}_i + \alpha_i \boldsymbol{M}_i)^{\mathrm{T}}\right) - \mathrm{Sp}\left(\bar{\boldsymbol{H}}_i \boldsymbol{I} \bar{\boldsymbol{H}}_i^{\mathrm{T}}\right) \\ &= \mathrm{Sp}\left(\alpha_i \boldsymbol{M}_i \boldsymbol{I}\, \alpha_i \boldsymbol{M}_i^{\mathrm{T}}\right) - \mathrm{Sp}\left(2\, \alpha_i \boldsymbol{M}_i \boldsymbol{I} \bar{\boldsymbol{H}}_i^{\mathrm{T}}\right) \\ &= \alpha_i^2\, \mathrm{Sp}\left(\boldsymbol{M}_i \boldsymbol{I} \boldsymbol{M}_i^{\mathrm{T}}\right) - 2\, \alpha_i\, \mathrm{Sp}\left(\boldsymbol{M}_i \boldsymbol{I} \bar{\boldsymbol{H}}_i^{\mathrm{T}}\right)\end{aligned}$$

$\Delta I'_{\mathrm{k}}(\alpha)$ ist also eine quadratische Funktion in α, wir können sie uns als Parabel vorstellen. Das Minimum der Parabel und damit das optimale α läßt sich nun leicht berechnen:

a) Die Parabel ist nach oben geöffnet: Der Scheitel und damit das Minimum der Parabel ist an der Stelle

$$\alpha_{\mathrm{opt}} = -\frac{\mathrm{Sp}\left(\boldsymbol{M}_i \boldsymbol{I} \bar{\boldsymbol{H}}_i^{\mathrm{T}}\right)}{\mathrm{Sp}\left(\boldsymbol{M}_i \boldsymbol{I} \boldsymbol{M}_i^{\mathrm{T}}\right)}\,.$$

Liegt der Scheitel der Parabel aber nicht im zulässigen Intervall, $\alpha_{\mathrm{opt}} \notin [\alpha_{\min}, \alpha_{\max}]$, gehen wir zu Punkt b).

b) Die Parabel ist nach unten geöffnet oder der Scheitel der Parabel liegt nicht im zulässigen Intervall: Wir berechnen $\Delta I'_{\mathrm{k}}(\alpha_{\min})$ und $\Delta I'_{\mathrm{k}}(\alpha_{\max})$ und wählen

$$\alpha_{\mathrm{opt}} = \min_{\alpha_{\min}, \alpha_{\max}} \Delta I'_{\mathrm{k}}(\alpha)\,.$$

4. Wenn ein Abbruchkriterium erfüllt ist, stoppe, sonst setze $i = i + 1$ und gehe nach 2.

Die Struktur des PAP-Algorithmus gleicht also einem typischen mehrdimensionalen Minimierungsalgorithmus, wobei aber bei der Richtungswahl und der Linienminimierung das konkrete Wissen über unser spezielles Minimierungsproblem eingeflossen ist.

Es wurden verschiedene Optimierungen mit zufälligen Eingangsstatistiken und verschiedenen Knotendimensionen durchgeführt. Dabei wurde der PAP-Algorithmus mit dem klassischen „General Purpose Solver“ MINOS verglichen. Bei sehr kleinen Knotendimensionen von zwei oder drei Bit brauchte MINOS zwar schon deutlich länger als der PAP-Algorithmus, konnte die Knoten aber immerhin in wenigen Minuten optimieren. Sobald man aber zu praxisrelevanten Knotengrößen von acht Bit überging, wurde der Abstand zwischen den beiden Verfahren sehr groß. Es zeigte sich, daß die Rechenzeit des PAP-Algorithmus pro 8-Bit-Knoten durchweg zwischen 10 und 20 Sekunden betrug, MINOS brauchte pro 8-Bit-Knoten dagegen jeweils mehrere Stunden. Dazu kommt, daß MINOS einem lokalen Minimum nicht entkommen kann und deshalb ab und zu in einem lokalen Minimum terminierte. Der PAP-Algorithmus hat durch die stochastische Richtungswahl kaum Probleme mit lokalen Minima und terminiert nur sehr selten in einem lokalen Minimum (seltener als in einem Prozent der Fälle). Wenn MINOS nicht in einem lokalen Minimum terminierte, unterschieden sich die Werte der beiden Algorithmen für die Kostenfunktion nur im Promillebereich.

8 Ein Knoten mit Rückkopplungen

Im Gegensatz zum vorherigen Kapitel sind an einem Knoten nun rückgekoppelte Leitungen erlaubt. Zunächst betrachten wir nur rückgekoppelte Leitungen an *einem* Knoten, d. h. ein Teil der Ausgangsleitungen eines Knotens gehen wieder zum Eingang des Knotens zurück.

8.1 Auswirkung der Rückkopplungen

Durch die Einführung von Rückkopplungen bekommt ein Netzwerk eine neue Qualität, die es ohne Rückkopplungen nicht hat. Dies gilt für Funktionennetze im allgemeinen, nicht nur für HPNs. Obwohl auch ein vorwärtsgerichtetes HPN ein dynamisches System ist, konnten wir es als eine *zeitunabhängige* Funktion betrachten, die einen Eingangsbitstring auf einen Ausgangsbitstring abbildet. Wegen der Rückkopplungen können wir auf die Zeit als Parameter nicht mehr verzichten, weshalb wir ein HPN wieder als „echtes“ dynamisches System betrachten. Die Abbildung der Eingangsmuster auf die Ausgangsmuster wird ein dynamischer Prozeß, so wie er in Definition 10 auf Seite 37 eingeführt wurde. Am Eingang eines rückgekoppelten HPN liegt jetzt eine zeitliche *Folge* von Bitstrings an und pro Arbeitstakt des HPN kommt ein Bitstring am Ausgang heraus. Veranschaulichen können wir uns die Arbeitsweise eines rückgekoppelten Knotens, wenn wir seine zeitliche Abwicklung betrachten, siehe Abbildung 8.1.

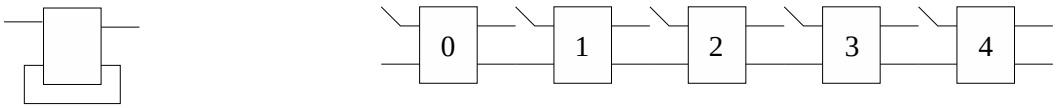


Abbildung 8.1: Zeitliche Abwicklung eines rückgekoppelten HPN-Knotens. Anstelle eines rückgekoppelten Knotens (links) können wir uns auch eine Folge von vorwärtsgerichteten Knoten ohne Rückkopplungen vorstellen (rechts). Am Knoten 0 liegen die Eingangsdaten des Zeitschritts 0 an und die Ausgangsdaten des Zeitschritt 1. Am Knoten 1 liegen die Eingangsdaten des Zeitschritts 1 an und die Ausgangsdaten des Zeitschritt 2 usw.

Die Aufgabe eines rückgekoppelten HPNs, auch RHPN abgekürzt, ist also, zeitliche Ströme von Daten zu verarbeiten. Dies können Meßwerte sein, die tätsächlich zeitlich gesehen nacheinander anfallen. Ebenso kann es aber auch sinnvoll sein, räumliche Muster in zeitliche Muster umzuwandeln und diese dann seriell zu verarbeiten. Dadurch läßt sich ein Netzwerk u. U. kompakter realisieren.

Ein Knoten, der im Zeitschritt t den Wert seiner Rückkopplungsleitungen setzt, kann diesen Wert im Zeitschritt $t+1$ an seinem Eingang wieder einlesen. Die Rückkopplungsleitungen eines Knotens stellen also einen Speicher dar, der gerade soviele Bit Speicherplatz hat wie der Knoten Rückkopplungsleitungen hat. Daher können wir dem Knoten einen zeitabhängigen Zustand $\boldsymbol{r}$ zuordnen, der bei r Rückkopplungsleitungen 2^r verschiedene Werte annehmen kann. Durch die Zahl der Rückkopplungen kann man die Größe des internen Speichers eines RHPN festlegen, und zwar unabhängig von der Länge der Eingangsbitstrings. Ist der interne Speicher für eine

gestellte Aufgabe zu klein, wird das RHPN diese Aufgabe nicht lösen können. Die Zahl der optimierenden Parameter läßt sich daher durch die Zahl der Rückkopplungen steuern.

8.1.1 Integration der Information in den Rückkopplungen

Wie können wir die Rückkopplungen eines Knotens nutzen, um bestimmte Aufgaben effektiver als ohne Rückkopplungen zu lösen? Nehmen wir dazu ein Standardproblem aus der Mustererkennung, das Erlernen der PARITY-Funktion. Das Netz soll unterscheiden können, ob die Zahl der Einsen in einem Bitstring gerade oder ungerade ist. Haben wir ein Bitstring der Länge n, brauchen wir ein vorwärtsgerichtetes Netzwerk mit mindestens n Eingängen, um jedes Bit des Eingangsbitstrings zu erfassen. Die Größe des Netzwerkeingangs wächst zwar nur linear mit n, bei großem n (z. B. $n = 1000$) wird das Netz trotzdem sehr groß.

Die Verhältnisse sind ganz anders, wenn das Netz einen Speicher hat und sich etwas merken kann. Im konkreten Fall des PARITY-Problems kann man den Bitstring nämlich auch seriell abarbeiten und muß sich nur merken, ob die Zahl der bisher im Bitstring gesehenen Einsen gerade oder ungerade war. Jedesmal, wenn wieder eine Eins vorbeikommt, muß man das „Merkbit“ invertieren. Diese Verarbeitung hat nicht nur den Vorteil, daß nun ein einzelner Knoten mit nur einer Rückkopplungsleitung das Problem lösen kann. Es gibt auch keine Längenbeschränkung für den Eingangsbitstring mehr. Die zu bearbeitenden Eingangsbitstrings können verschieden lang und n beliebig groß sein, ohne daß man den Knoten oder seinen Tabelleninhalt dafür ändern müßte. Wie man das XOR-Problem, daß ja das PARITY-Problem für $n = 2$ ist, trainiert, sehen wir in Abschnitt 10 auf Seite 131.

Der entscheidende Punkt ist also, daß die Information, die der Knoten bis zu einem bestimmten Zeitpunkt aus den Daten erhalten hat, in den Rückkopplungen steckt. Wir können uns die Rückkopplungen wie ein Sieb vorstellen, das die in den Daten steckende Information heraussiebt und zeitlich integriert.

8.1.2 Endliche Automaten

Ein einzelner HPN-Knoten mit Rückkopplungen läßt sich auch als endlicher Automat interpretieren. Ein rückgekoppelter Knoten der Dimension d mit r rückgekoppelten und $d - r$ Ein- bzw. Ausgangsleitungen läßt sich direkt auf einen endlichen Automaten abbilden. Dieser Automat hat ein Ein- bzw. Ausgabealphabet aus $\mathbb{B}^{d-r}$ und 2^r interne Zustände. Die Übergangs- und Ausgabefunktion des Automaten lassen sich direkt aus der Tabelle des rückgekoppelten Knotens konstruieren. Der Automat, der einem rückgekoppelten Knoten äquivalent ist, wird durch den Knoten implizit realisiert.

Ebenso läßt sich ein Netzwerk mehrerer rückgekoppelter Knoten als Automatennetz interpretieren, solange die Rückkopplungen auf die einzelnen Knoten beschränkt sind. Bei einem RHPN können wir Rückkopplungen aber auch zwischen beliebigen Knoten einfügen, so daß der gesamte Zustandsraum des Netzes nicht mehr in die Unterräume der einzelnen Knoten zerfällt. In diesem Fall würde man den Gesamtzustandsraum des Automaten als Graph beschreiben, der für mehr als 10000 Zustände unhandlich wird. Für die globale Optimierung eines RHPN macht dies keinen Unterschied, da wir mit dem gesamten Zustandsraum nicht explizit in Berührung kommen. Wollten wir dagegen den entprechenden Automaten explizit beschreiben, ist dies aufwendiger, da die Zahl der Zustände im gesamten Zustandsraum schnell wächst. Bei einem kleinen RHPN, z. B. mit sieben Knoten und zwei Rückkopplungen pro Knoten hat der gesamte Zustandsraum schon $2^{7 \cdot 2} = 16384$ Zustände.

8.2 Die Kostenfunktion

Durch die Einführung von Rückkopplungen ändert sich die Struktur der Kostenfunktion erheblich.

8.2.1 Die zeitliche Dynamik mit Rückkopplungen

Zur Erinnerung vergegenwärtigen wir uns noch einmal die Dynamik eines Knotens im Komponentenraum. Die Zeitentwicklungsgleichung eines HPN ist (aus Definition 9 auf Seite 36):

$$x_e(t+1) = [\mathrm{h}_v(\boldsymbol{z}(t))]_k \qquad \text{mit} \qquad \begin{aligned} & t \geq 0 \\ & e = \mathrm{out}(v,k) \\ & z_i(t) = x_{\mathrm{in}(v,i)}(t) = \sigma^{\mathrm{ko}}(\mathrm{in}(v,i),t) \\ & v \in V;\ k,i = 0,\ldots,\dim(v)-1 \end{aligned}$$

Für einen einzelnen Knoten, der gar nicht „weiß“, daß sein Eingang an seinem eigenen Ausgang angeschlossen ist, bedeutet dies:

1. Fasse die Eingangsleitungen $x_{\mathrm{in}(v,i)}(t)$ in dem Bitstring $\boldsymbol{z}(t)$ zusammen.
2. Schlage $\boldsymbol{z}(t)$ in der Knotentabelle h_v nach.
3. Schreibe das Ergebnis $\mathrm{h}_v(\boldsymbol{z}(t))$ auf die Ausgangsleitungen und gehe zu 1.

Die Operationen 1.–3. lassen sich sehr leicht auf einem Digitalcomputer implementieren:

1. Interpretiere den Bitstring $\boldsymbol{z}(t)$ als ganze positive Zahl und diese Zahl als Index.
2. Schlage im RAM an der durch den „Index“ definierten Stelle den Tabelleninhalt nach.
3. Interpretiere den Tabelleninhalt als Bitmuster und schreibe es auf die Ausgangsleitungen.

Stellen wir also die zeitliche Dynamik des RHPN im diskreten Komponentenraum dar, brauchen wir einen Optimierungsalgorithmus, z. B. Simulated Annealing, der mit einem diskreten Parameterraum umgehen kann.

Die Dynamik eines Knotens im Produktraum sieht dagegen nach Gleichung 4.18 auf Seite 55 so aus:

$$\boldsymbol{\chi}_j(t+1) = \boldsymbol{\Phi}_j \cdot \boldsymbol{H} \cdot \left(\bigotimes_{j=m+k-1}^{0} \boldsymbol{\chi}_j(t) \right) \tag{8.1}$$

Hier sind die Schritte 1.–3.:

1. Fasse die Leitungsvektoren $\boldsymbol{\chi}_j(t)$ der Eingangsleitungen durch Kronecker-Multiplikation zusammen.
2. Berechne den transformierten Zustandsvektor durch Matrixmultiplikation
3. Projiziere das Ergebnis mit einer Matrixmultiplikation auf die Ausgangsleitungen und gehe zu 1.

Die Schritte 1.–3. werden bei jedem Zeitschritt durchgeführt, wodurch die rückgekoppelten Leitungen immer wieder mit der Matrix $\boldsymbol{H}$ multipliziert werden. Bei einem geschlossenen rückgekoppelten Knoten ist der Zustandsvektor der rückgekoppelten Leitungen $\boldsymbol{\chi}_{\mathrm{r}}(t)$ zum Zeitschritt t gerade $\boldsymbol{\chi}_{\mathrm{r}}(t) = \boldsymbol{H}^t \cdot \boldsymbol{\chi}_{\mathrm{r}}(0)$. Die Komponenten von $\boldsymbol{\chi}_{\mathrm{r}}(t)$ sind Polynome aus den Elementen von $\boldsymbol{H}$, deren Grad mit jedem Zeitschritt um eins wächst. Dies und die Matrixmultiplikationen ist der „Preis“, den man zahlt, um die Optimierung eines RHPN mit kontinuierlichen Optimierungsmethoden durchführen zu können.

8.2.2 Berechnung der Verbundstatistik mehrerer Leitungsbündel

Ein Knoten mit Rückkopplungen hat an seinem Eingang zwei ankommende Leitungsbündel, nicht nur eins wie bisher. Nehmen wir nun an, wie müßten N Leitungsbündel miteinander kombinieren. Die Gewichtsvektoren der Leitungsbündel seinen $\boldsymbol{w}_i$. Normieren wir die Gewichtsvektoren mit der Zahl der jeweils aufgetretenen Muster n, erhalten wir die erwartungstreuen Schätzungen der Symbolwahrscheinlichkeiten:

$$\boldsymbol{p}_i = \boldsymbol{w}_i / n$$

$[\boldsymbol{p}_i]_j$ ist die Wahrscheinlichkeit, daß auf Bündel i das Muster j auftritt. Wie wir jetzt die Verbundwahrscheinlichkeit eines aus den N Bündeln zusammengesetzten neuen Bündels $\boldsymbol{x}$ berechnen, sagt uns

Satz 9. *Seien $\boldsymbol{p}_i$ die Symbolwahrscheinlichkeitsverteilungen von N statistisch unabhängigen Leitungsbündeln $\boldsymbol{x}_i$ und sei $\boldsymbol{x}$ das zusammengesetzte Bündel $\boldsymbol{x} = (\boldsymbol{x}_0, \ldots, \boldsymbol{x}_{N-1})$. Dann sind die Symbolwahrscheinlichkeiten des Bündels $\boldsymbol{x}$ gegeben durch*

$$\boldsymbol{p} = \bigotimes_{i=N-1}^{0} \boldsymbol{p}_i \,.$$

Beweis. Jedes Symbol auf dem Bündel $\boldsymbol{x}$ setzt sich aus einer Kombination der N Symbole der Bündel $\boldsymbol{x}_i$ zusammen. Nach Voraussetzung sind die einzelnen Wahrscheinlichkeitsverteilungen $\boldsymbol{p}_i$ statistisch unabhängig. Daher ist die Wahrscheinlichkeit, das Symbol $(\boldsymbol{y}_0, \ldots, \boldsymbol{y}_{N-1})$ auf dem Bündel $\boldsymbol{x}$ zu beobachten,

$$p(\boldsymbol{y}_0, \ldots, \boldsymbol{y}_{N-1}) = \prod_{i=N-1}^{0} p(\boldsymbol{x}_i = \boldsymbol{y}_i) = \prod_{i=N-1}^{0} [\boldsymbol{p}_i]_{\mathrm{ind}(\boldsymbol{y}_i)} \,.$$

Wir müssen aus jeder Wahrscheinlichkeitsverteilung $\boldsymbol{p}_i$ die zum Symbol $\boldsymbol{y}_i$ gehörende Komponente $\mathrm{ind}(\boldsymbol{y}_i)$ auswählen und erhalten so die richtige Wahrscheinlichkeit. Nun ist nach Definition 16 auf Seite 44 der Zustandsvektor $\boldsymbol{\chi}$ eines Leitungsbündels $\boldsymbol{\chi} = (\boldsymbol{\chi}_0, \ldots, \boldsymbol{\chi}_{n-1})$ im Produktraum durch

$$\boldsymbol{\chi} = \bigotimes_{i=N-1}^{0} \boldsymbol{\chi}_i$$

gegeben. Da die Wahrscheinlichkeit $[\boldsymbol{p}_i]_j$ zum Symbol i des Bündels i, also zu $[\boldsymbol{\chi}_i]_j$, gehört, erhalten wir genau dann die richtige Zuordnung, wenn

$$\boldsymbol{p} = \bigotimes_{i=N-1}^{0} \boldsymbol{p}_i \,.$$

□

8.2.3 Bewertungsmöglichkeiten der Trajektorien im Zustandsraum

Bei der Verarbeitung einer Bitstringfolge haben die rückgekoppelten Leitungen eines Knotens bei jedem Zeitschritt einen anderen Zustand. Eine solche Folge von Zuständen nennen wir *Trajektorie*, die wir folgendermaßen definieren:

Definition 24. Bei der Abarbeitung einer Bitstringfolge $\boldsymbol{x}(t)$ nehmen die rückgekoppelten Leitungen eines Knotens während der diskreten Zeitschritte $t = (0, 1, \ldots, n-1)$ gerade n Zustände $\boldsymbol{r}(t)$ an. Die Folge aller zu einer Bitstringfolge $\boldsymbol{x}(t)$ gehörenden Zustände $(\boldsymbol{r}(0), \boldsymbol{r}(1), \ldots, \boldsymbol{r}(n-1))$ nennen wir die *Trajektorie* zum Datenstring $\boldsymbol{x}(t)$.

Haben die Bitstringfolgen die Länge n, sind sie also nach n Zeitschritten vom Knoten verarbeitet worden, der dabei n Zustände angenommen hat. Nun gibt es bei einem rückgekoppelten Knoten mehrere Möglichkeiten, diese Zustände zu bewerten:

Bewertung des Haltezustandes: Die naheliegenste Möglichkeit ist, die kalte Information des Haltezustandes des Knotens zu minimieren. Das bedeutet, der Knoten soll möglichst selten für zwei Bitstringfolgen aus unterschiedlichen Klassen im selben Zustand *anhalten*. Möchte man nun den Knoten als Klassifikator betreiben, braucht man nur die verschiedenen Bitstringfolgen mit dem Knoten zu prozessieren. Aus dem Haltezustand, dem einzigen bewerteten Zustand der Trajektorie, bekommt man die klassenspezifischen Rückschlußwahrscheinlichkeiten, so wie im vorwärtsgerichteten Fall. Interessiert sich ein äußerer Beobachter nur für den Haltezustand eines Knotens, ist diese Bewertung die richtige, da sie dem Knoten bei allen Zeitschritten $t < n$ maximale Freiheit läßt, seine internen Zustände zu organisieren.

Bewertung der gesamten Trajektorie: Ist der Knoten mitten in einem Netzwerk und soll er lokal optimiert werden, reicht es nicht aus, nur den Haltezustand zu bewerten. Denn es ist denkbar, daß für $t < n$ die Rückschlußwahrscheinlichkeiten für alle Zustände für die verschiedenen Klassen gleich sind und erst mit $t = n$ die Rückschlußwahrscheinlichkeiten für die verschiedenen Klassen unterschiedlich sind. Ein Knoten im Netz soll aber schon möglichst früh seinen nachfolgenden Knoten Information zukommen lassen, und nicht erst im letzten Zeitschritt, denn dann ist es zu spät. Daher ist es in diesem Fall günstiger, die gesamte Trajektorie zu bewerten. Dazu wird die kalte Information der Zustände während der gesamten Trajektorie von $t = 0$ bis $t = n$ als Kostenfunktion genommen.

Bewertung von Trajektorienabschnitten: Man kann auch erlauben, daß der selbe Zustand von verschiedenen Klassen benutzt wird, aber nur zu unterschiedlichen Zeitschritten. Dann wäre zu jedem Zeitschritt die Identifikation der zu einem Zustand gehörenden Klasse möglich. Der Knoten hätte aber mehr Freiheit, seine internen Zustände zu organisieren, als wenn man die gesamte Trajektorie auf einmal bewertet. Um z. B. jeden Zeitschritt für sich zu bewerten, würden wir die kalte Information der Zustände pro Zeitschritt getrennt berechnen und erst zum Schluß über alle Zeitschritte summieren.

Die verschiedenen Bewertungsmöglichkeiten verdeutlicht Abbildung 8.2.3.

8.2.4 Rekursionsgleichungen der Systemgrößen

Die zeitliche Entwicklung der Systemgrößen läßt sich auch in Form von Rekursionsgleichungen schreiben. Dies ist dann nützlich, wenn wir wie z. B. im nächsten Abschnitt die Systemgrößen inkrementell berechnen wollen. Auf diese Weise lassen sich die Systemgrößen zum Zeitpunkt $t + 1$ aus den Systemgrößen zum Zeitpunkt t und dem Zustandsvektor des rückgekoppelten Knotens zum Zeitpunkt t berechnen.

Für die Rekursionsgleichungen benötigen wir zunächst eine Abbildung

$$\mathrm{I}: \begin{array}{c} \text{Zeitdefinitionsbereich} \to \text{Symbolwertebereich} \\ t \mapsto \mathrm{I}(t) \end{array},$$

Zeitschritt Zustand	0	1	2	3	4	5	6	7	8
z_0									x
z_1									x
z_0	x	x	x	x	x	x	x	x	x
z_1	x	x	x	x	x	x	x	x	x
z_0			x						
z_1			x						

Abbildung 8.2: Bewertungsmöglichkeiten von Trajektorien. Die Tabelle visualisiert eine Leitung mit zwei Zuständen z_0 und z_1. Die hypothetischen Eingangsdaten haben die Länge 9, so daß sich 9 Zeitschritte aus dem Intervall $t = [0, \ldots, 8]$ ergeben. Die drei im Text angesprochenen Möglichkeiten sind untereinander verglichen, wobei jeweils diejenigen Punkte der Trajektorien durch ein x gekennzeichnet sind, die in die Statistikberechnung einbezogen werden.
Im obersten Fall wird nur der Haltezustand der Leitung in die Berechnung der Statistik mit einbezogen. Im mittleren Fall werden alle Punkte der Trajektorien für die Statistik bewertet. Im unteren Fall wird die Statistik für jeden Zeitschritt getrennt ermittelt und erst hinterher akkumuliert. Man sieht also eine Momentaufnahme des Vorgangs für $t = 2$.

die das zum Zeitpunkt t gehörende Eingabesymbol liefert. Wir betrachten hier die Eingangsleitungen des Knotens nach „normalen“ und rückgekoppelten Leitungen getrennt. Sind die rückgekoppelten Bits am „oberen“ Ende des Knotens angebracht, ist die Transformationsmatrix $\boldsymbol{H}$ bei s verschiedenen Eingabesymbolen folgendermaßen in die quadratischen $\boldsymbol{H}_k$ zerlegbar:

$$\boldsymbol{H} = \left(\boldsymbol{H}_0, \boldsymbol{H}_1, \ldots, \boldsymbol{H}_{s-1}\right) \tag{8.2}$$

Die Rekursionsgleichungen für den Zustandsvektor $\boldsymbol{z}^t$ lauten dann:

$$\begin{aligned} \boldsymbol{z}^0 &= \boldsymbol{\epsilon}_0 \\ \boldsymbol{z}^{t+1} &= \boldsymbol{H}_{\mathrm{I}(t)} \cdot \boldsymbol{z}^t \end{aligned} \tag{8.3}$$

$\boldsymbol{H}_k$ ist jeweils eine der Matrizen, die von der Eingangsabbildung $\mathrm{I}(t)$ ausgewählt wird. Uns interessiert die Veränderung des Zustandsvektors, wenn wir die Elemente von $\boldsymbol{H}$ ändern. Dazu benötigen wir den Gradienten des Zustandsvektors. Wir benutzen ab jetzt die Summenkonvention $a_i b_i := \sum_i a_i b_i$ und $(a_i)^2 := a_i a_i$. In Tensor-Notation ergibt sich mit den Elementen h_{plm}

der Matrix $\boldsymbol{H}_p$:[1]

$$\begin{aligned} z_i^0 &= \boldsymbol{\epsilon}_0, i \\ z_i^{t+1} &= h_{\mathrm{I}(t)ij}\, z_j^t \\ \partial_{h_{plm}} z_i^{t+1} &= \frac{\partial z_i^{t+1}}{\partial h_{plm}} = \frac{\partial h_{\mathrm{I}(t)ij}\, z_j^t}{\partial h_{plm}} = \frac{\partial h_{\mathrm{I}(t)ij}}{\partial h_{plm}}\, z_j^t + h_{\mathrm{I}(t)ij}\, \frac{\partial z_j^t}{\partial h_{plm}} \\ &= \delta_{p,l,m}^{\mathrm{I}(t),i,j}\, z_j^t + h_{\mathrm{I}(t)ij}\, \partial_{h_{plm}} z_j^t \\ &= \delta_{p,l}^{\mathrm{I}(t),i}\, z_m^t + h_{\mathrm{I}(t)ij}\, \partial_{h_{plm}} z_j^t \end{aligned} \tag{8.4}$$

Wir untersuchen nun, wie sich die klassenspezifischen Gewichtsvektoren $\boldsymbol{w}_k$ von einem Zeitschritt zum nächsten ändern. Dazu benötigen wir die Abbildung K : Zeitdefinitionsbereich $\to$ Symboldefinitionsbereich, $t \mapsto K^t$, die die zum Zeitpunkt t gehörende Klasse liefert. Die klassenspezifischen Gewichtsvektoren sind gerade die Spalten der Gewichtsmatrix $\boldsymbol{W}$. Bei m Klassen ergibt sich:

$$\boldsymbol{W} = \left(\boldsymbol{w}_0, \boldsymbol{w}_1, \ldots, \boldsymbol{w}_{m-1}\right)$$

Ein Element w_{ik} von $\boldsymbol{W}$ gibt also die Häufigkeit des Symbols i in der Klasse k an. Da die Zeit nun als zusätzlicher Parameter hereinkommt, gibt es zu jedem Zeitschritt t eine Gewichtsmatrix $\boldsymbol{W}^t$ mit den Elementen w_{ik}^t. Wir definieren nun den Tensor $\boldsymbol{\eta}$, der nur Einsen enthält. In einer Dimension sieht er, als Vektor geschrieben, so aus:

$$\boldsymbol{\eta} := \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}$$

Damit läßt sich die Summation über einen Index eines Tensors durchführen. Wir erhalten so z. B. die Gesamthäufigkeiten der Symbole und die Gesamtzahl der beobachteten Ereignisse:

$$w_i^t := w_{ik}^t \eta_k \qquad N^t := w_{ik}^t \eta_{ik}$$

Die Rekursionsgleichungen für die klassenspezifischen Gewichtsvektoren $\boldsymbol{w}_k$ und ihre Gra-

[1] Man kann die Berechnungen auch in der Kronecker-Notation anstelle der Tensor-Notation durchführen. Dazu nehmen wir folgende Gleichungen zu Hilfe:

$$\frac{\partial \boldsymbol{X}}{\partial \boldsymbol{X}} = \sum_{r,s} \boldsymbol{E}_{rs} \otimes \boldsymbol{E}_{rs} =: \bar{U}$$

$$\frac{\partial (\boldsymbol{XY})}{\partial \boldsymbol{Z}} = \frac{\partial \boldsymbol{X}}{\partial \boldsymbol{Z}} (\boldsymbol{1}_q \otimes \boldsymbol{Y}) + (\boldsymbol{1}_p \otimes \boldsymbol{X}) \frac{\partial \boldsymbol{Y}}{\partial \boldsymbol{Z}}$$

Die Matrix $\boldsymbol{Z}$ hat die Dimension $p \times q$ und $\boldsymbol{1}_p$ bzw. $\boldsymbol{1}_q$ sind Einheitsmatrizen der Dimension $p \times p$ bzw. $q \times q$. Der Gradient des Zustandsvektors bezüglich der p-ten $r \times c$ – Transformationsmatrix ist:

$$\begin{aligned} \partial_{\boldsymbol{H}_p} \boldsymbol{z}^{t+1} &= \frac{\partial \boldsymbol{z}^{t+1}}{\partial \boldsymbol{H}_p} = \partial_{\boldsymbol{H}_p} \left(\boldsymbol{H}_{\mathrm{I}(t)} \cdot \boldsymbol{z}^t\right) = (\partial_{\boldsymbol{H}_p} \boldsymbol{H}_{\mathrm{I}(t)})(\boldsymbol{1}_c \otimes \boldsymbol{z}^t) + (\boldsymbol{1}_r \otimes \boldsymbol{H}_{\mathrm{I}(t)}) \partial_{\boldsymbol{H}_p} \boldsymbol{z}^t \\ &= \delta_p^{\mathrm{I}(t)} (\partial_{\boldsymbol{H}_p} \boldsymbol{H}_p)(\boldsymbol{1}_c \otimes \boldsymbol{z}^t) + (\boldsymbol{1}_r \otimes \boldsymbol{H}_{\mathrm{I}(t)}) \partial_{\boldsymbol{H}_p} \boldsymbol{z}^t \\ &= \delta_p^{\mathrm{I}(t)} \bar{U} (\boldsymbol{1}_c \otimes \boldsymbol{z}^t) + (\boldsymbol{1}_r \otimes \boldsymbol{H}_{\mathrm{I}(t)})\, \partial_{\boldsymbol{H}_p} \boldsymbol{z}^t \end{aligned}$$

dienten lauten in Tensor-Notation mit den Gewichtsmatrixelementen w_{ik}: [2]

$$\begin{aligned} w_{ik}^0 &= 0 \\ w_{ik}^{t+1} &= w_{ik}^t + \delta_k^{K^{t+1}} z_i^{t+1} \\ \partial_{h_{plm}} w_{ik}^{t+1} &= \partial_{h_{plm}} w_{ik}^t + \delta_k^{K^{t+1}} \partial_{h_{plm}} z_i^{t+1} \end{aligned}$$

Die Systemgrößen zum Zeitpunkt t und ihr Gradient lassen sich nun durch die w_{ik} ausdrücken (der Zeitindex t wird weggelassen):

$$\begin{aligned} D_\mathrm{t} &= N^2 = (w_{ik}\eta_{ik})^2 & \partial_{h_{plm}} D_\mathrm{t} &= 2\, w_{ik}\eta_{ik}\,\eta_{no}\,\partial_{h_{plm}} w_{no} \\ D_\mathrm{k} &= (w_i)^2 & \partial_{h_{plm}} D_\mathrm{k} &= 2\, w_i\,\partial_{h_{plm}} w_i \\ R_\mathrm{t} &= (w_{ik}\eta_i)^2 & \partial_{h_{plm}} R_\mathrm{t} &= 2\, w_{ik}\eta_i\,\eta_j\,\partial_{h_{plm}} w_{jk} \\ R_\mathrm{k} &= (w_{ik})^2 & \partial_{h_{plm}} R_\mathrm{k} &= 2\, w_{ik}\,\partial_{h_{plm}} w_{ik} \end{aligned}$$

Für die kalte Information und kalte Nettoinformation erhalten wir:

$$\begin{aligned} I_\mathrm{k} &= D_\mathrm{k} - R_\mathrm{k} \\ \partial_{h_{plm}} I_\mathrm{k} &= \partial_{h_{plm}} (D_\mathrm{k} - R_\mathrm{k}) = \partial_{h_{plm}} D_\mathrm{k} - \partial_{h_{plm}} R_\mathrm{k} \\ I_\mathrm{nk} &= D_\mathrm{k} - \frac{D_\mathrm{t}}{R_\mathrm{t}} R_\mathrm{k} \\ \partial_{h_{plm}} I_\mathrm{nk} &= \partial_{h_{plm}} \left(D_\mathrm{k} - \frac{D_\mathrm{t}}{R_\mathrm{t}} R_\mathrm{k} \right) = \partial_{h_{plm}} D_\mathrm{k} - \left(R_\mathrm{k}\,\partial_{h_{plm}} \frac{D_\mathrm{t}}{R_\mathrm{t}} + \frac{D_\mathrm{t}}{R_\mathrm{t}}\,\partial_{h_{plm}} R_\mathrm{k} \right) \\ &= \partial_{h_{plm}} D_\mathrm{k} - \left(R_\mathrm{k} \frac{R_\mathrm{t}\,\partial_{h_{plm}} D_\mathrm{t} - D_\mathrm{t}\,\partial_{h_{plm}} R_\mathrm{t}}{(R_\mathrm{t})^2} + \frac{D_\mathrm{t}}{R_\mathrm{t}}\,\partial_{h_{plm}} R_\mathrm{k} \right) \end{aligned}$$

Auch für die kalte Information läßt sich eine Rekursionsgleichung aufstellen. Diese kann z. B. dann nützlich sein, wenn man den Beitrag eines Zustands zur Information für verschiedene Zeiten verschieden gewichten möchte. Wir leiten nun mit (8.2.4) für diejenigen Größen Rekursionsgleichungen ab, die zur Berechnung der Rekursionsgleichung der kalten Information nötig sind. Nebenbei sieht man in dieser Darstellung auch, welchen Beitrag ein neu hinzukommender Zustandsvektor $\boldsymbol{z}$ zu den Systemgrößen liefert.

- Kalte Diversität D_k:

$$\begin{aligned} D_\mathrm{k}^0 &= 0 \\ D_\mathrm{k}^{t+1} - D_\mathrm{k}^t &= w_i^{t+1} w_i^{t+1} - w_i^t w_i^t = \left(w_i^t + z_i^{t+1}\right)\left(w_i^t + z_i^{t+1}\right) - w_i^t w_i^t = 2\, w_i^t z_i^{t+1} + z_i^{t+1} z_i^{t+1} \\ \rightarrow \quad D_\mathrm{k}^{t+1} &= D_\mathrm{k}^t + 2\, w_i^t z_i^{t+1} + z_i^{t+1} z_i^{t+1} \end{aligned}$$

[2] In Kronecker-Notation erhalten wir:

$$\begin{aligned} \boldsymbol{w}_k^0 &= 0 \\ \boldsymbol{w}_k^{t+1} &= \boldsymbol{w}_k^t + \delta_k^{K^{t+1}} \boldsymbol{z}^{t+1} \\ \partial_{\boldsymbol{H}_p} \boldsymbol{w}_k^{t+1} &= \partial_{\boldsymbol{H}_p} \boldsymbol{w}_k^t + \delta_k^{K^{t+1}} \partial_{\boldsymbol{H}_p} \boldsymbol{z}^{t+1} \end{aligned}$$

- Kalte Redundanz R_{k}:

$$
\begin{aligned}
R_{\mathrm{k}}^0 &= 0 \\
R_{\mathrm{k}}^{t+1} - R_{\mathrm{k}}^t &= w_{ik}^{t+1} w_{ik}^{t+1} - w_{ik}^t w_{ik}^t = \left(w_{ik}^t + \delta_k^{K^{t+1}} z_i^{t+1}\right)\left(w_{ik}^t + \delta_k^{K^{t+1}} z_i^{t+1}\right) - w_{ik}^t w_{ik}^t \\
&= 2 w_{ik}^t \delta_k^{K^{t+1}} z_i^{t+1} + \delta_k^{K^{t+1}} \delta_k^{K^{t+1}} z_i^{t+1} z_i^{t+1} = 2 w_{ik}^t \delta_k^{K^{t+1}} z_i^{t+1} + z_i^{t+1} z_i^{t+1} \\
\rightarrow \quad R_{\mathrm{k}}^{t+1} &= R_{\mathrm{k}}^t + 2 w_{iK^{t+1}}^t z_i^{t+1} + z_i^{t+1} z_i^{t+1}
\end{aligned}
$$

- Kalte Information I_{k}:

$$
\begin{aligned}
I_{\mathrm{k}}^0 &= 0 \\
I_{\mathrm{k}}^{t+1} &= D_{\mathrm{k}}^{t+1} - R_{\mathrm{k}}^{t+1} = D_{\mathrm{k}}^t + 2\, w_i^t z_i^{t+1} + \left(z_i^{t+1}\right)^2 - \left(R_{\mathrm{k}}^t + 2 w_{iK^{t+1}}^t z_i^{t+1} + \left(z_i^{t+1}\right)^2\right) \\
&= D_{\mathrm{k}}^t - R_{\mathrm{k}}^t + 2\, w_i^t z_i^{t+1} - 2 w_{iK^{t+1}}^t z_i^{t+1} \\
&= I_{\mathrm{k}}^t + 2(w_i^t - w_{iK^{t+1}}^t) z_i^{t+1}
\end{aligned}
$$

8.3 Erweiterung des PAP-Algorithmus auf Rückkopplungen

Da der PAP-Algorithmus bei rückkopplungsfreien Knoten sehr effizient ist, lag es nahe, ihn auf rückgekoppelte Knoten zu erweitern. Die Grundidee ist die gleiche wie beim PAP-Algorithmus für vorwärtsgerichtete HPN aus Abschnitt 7.3.2. Wir ändern die stochastische Matrix eines Knotens dadurch, daß wir innerhalb einer oder mehrerer Spalten der stochastischen Matrix „Wahrscheinlichkeitsmasse“ verschieben. Dadurch wird die Nebenbedingung gewahrt, daß die Spaltensumme gleich eins sein soll.

Wir zerlegen die stochastische Matrix $\boldsymbol{H}$ eines Knotens wie in (8.2) in die quadratischen $\boldsymbol{H}_j$ und betrachten den Zustandsvektor eines einzelnen Knotens. Dann gilt für den ungestörten Fall die Rekursionsgleichung (8.3). Nun würfeln wir n „Zweibeine“

$$
\boldsymbol{M}_k = \begin{pmatrix}
0 & \dots & \dots & \dots & 0 \\
 & \ddots & 1 & & \\
\vdots & & \ddots & & \vdots \\
 & & -1 & \ddots & \\
0 & \dots & \dots & \dots & 0
\end{pmatrix},
$$

wobei wir die Spalte der Matrix $\boldsymbol{M}_k$, in der die 1 und -1 steht, mit l_k bezeichnen. Die in Abhängigkeit von α erzeugte stochastische Matrix ist

$$
\boldsymbol{H}'(\alpha) = \boldsymbol{H} + \alpha \sum_k \boldsymbol{M}_k \tag{8.5}
$$

Wir setzen für den Zustandsvektor zur Zeit t eine Potenzreihe in α an:

$$
\boldsymbol{z}^t = \sum_{i=0}^{\infty} \alpha^i \, \boldsymbol{c}_i^t \tag{8.6}
$$

Nun benötigen wir noch eine Abbildung[3]

$$
\begin{aligned}
L(l,m): \quad & ([0,\dots,\text{Spaltenzahl } \boldsymbol{H} - 1], [0,\dots,n-1]) \rightarrow [0,\dots,\text{Spaltenzahl } \boldsymbol{H}_j - 1] \\
& (l,m) \mapsto \delta_{\lfloor l/\{\text{Spaltenzahl } \boldsymbol{H}_j\}\rfloor, m}
\end{aligned}
$$

[3] $z = \lfloor x \rfloor$ ist die größte ganze Zahl z mit $z \leq x$.

Diese Abbildung ist also genau dann gleich eins, wenn die Spalte l_k in der Matrix $\boldsymbol{H}_j$ enthalten ist. Für den Zustandsvektor ergibt sich mit (8.2), (8.5) und (8.6):

$$\begin{aligned}\boldsymbol{z}^{t+1} &= \boldsymbol{H}_{\mathrm{I}(t)} \cdot \boldsymbol{z}^t + \sum_k \left[L(l_k, \mathrm{I}(t))\, \alpha\, \boldsymbol{M}_k\right] \cdot \boldsymbol{z}^t = \boldsymbol{H}_{\mathrm{I}(t)} \sum_{i=0}^{\infty} \alpha^i\, \boldsymbol{c}_i^t + \sum_k \left[L(l_k, \mathrm{I}(t))\, \alpha\, \boldsymbol{M}_k\right] \cdot \sum_{i=0}^{\infty} \alpha^i\, \boldsymbol{c}_i^t \\ &= \sum_{i=0}^{\infty} \alpha^i\, \boldsymbol{H}_{\mathrm{I}(t)} \boldsymbol{c}_i^t + \sum_{i=0}^{\infty} \sum_k \alpha^{(i+1)}\, L(l_k, \mathrm{I}(t))\, \boldsymbol{M}_k\, \boldsymbol{c}_i^t\end{aligned}$$

Mit einer Indexverschiebung $i \mapsto i-1$ im rechten Summanden und $\boldsymbol{c}_{-1}^t := 0$:

$$\begin{aligned}&= \sum_{i=0}^{\infty} \alpha^i\, \boldsymbol{H}_{\mathrm{I}(t)} \boldsymbol{c}_i^t + \sum_{i=0}^{\infty} \sum_k \alpha^i\, L(l_k, \mathrm{I}(t))\, \boldsymbol{M}_k\, \boldsymbol{c}_{i-1}^t \\ &= \sum_{i=0}^{\infty} \alpha^i \left[\boldsymbol{H}_{\mathrm{I}(t)} \boldsymbol{c}_i^t + \left(\sum_k L(l_k, \mathrm{I}(t))\, \boldsymbol{M}_k\right) \boldsymbol{c}_{i-1}^t\right] \\ &\stackrel{!}{=} \sum_{i=0}^{\infty} \alpha^i\, \boldsymbol{c}_i^{t+1}\end{aligned} \tag{8.7}$$

Daraus folgt:

$$\boldsymbol{c}_i^{t+1} = \boldsymbol{H}_{\mathrm{I}(t)} \boldsymbol{c}_i^t + \left(\sum_k L(l_k, \mathrm{I}(t))\, \boldsymbol{M}_k\right) \boldsymbol{c}_{i-1}^t \tag{8.8}$$

Die Anfangsbedingungen $\boldsymbol{c}_i^0$ ergeben sich aus der Anfangsbedingung für den Zustandsvektor $\boldsymbol{z}^0 = \boldsymbol{\epsilon}_0$

$$\boldsymbol{z}^0 = \boldsymbol{\epsilon}_0 = \sum_{i=0}^{\infty} \alpha^i\, \boldsymbol{c}_i^0 \qquad \rightarrow \qquad \begin{cases} \boldsymbol{c}_0^0 = \boldsymbol{\epsilon}_0 \\ \boldsymbol{c}_{i\neq 0}^0 = 0 \end{cases}$$

Würfeln wir nur ein „Zweibein“ $\boldsymbol{M}$, ist $n = 1$ und (8.8) wird

$$\boldsymbol{c}_i^{t+1} = \boldsymbol{H}_{\mathrm{I}(t)} \boldsymbol{c}_i^t + L(l, \mathrm{I}(t))\, \boldsymbol{M} \boldsymbol{c}_{i-1}^t\ .$$

Von der Potenzreihe in Gleichung 8.6 berücksichtigen wir nun nur Terme bis zur zweiten Ordnung in α. Der Zustand für kleine α ist dann

$$\boldsymbol{z}^t = \boldsymbol{c}_0^t + \boldsymbol{c}_1^t\, \alpha + \boldsymbol{c}_2^t\, \alpha^2 + O(\alpha^3)$$

und die Rekursionsgleichungen lauten

$$\begin{aligned}\boldsymbol{c}_0^{t+1} &= \boldsymbol{H}_{\mathrm{I}(t)}\, \boldsymbol{c}_0^t \\ \boldsymbol{c}_1^{t+1} &= \boldsymbol{H}_{\mathrm{I}(t)}\, \boldsymbol{c}_1^t + L(l, \mathrm{I}(t))\, \boldsymbol{M} \boldsymbol{c}_0^t \\ \boldsymbol{c}_2^{t+1} &= \boldsymbol{H}_{\mathrm{I}(t)}\, \boldsymbol{c}_2^t + L(l, \mathrm{I}(t))\, \boldsymbol{M} \boldsymbol{c}_1^t\ .\end{aligned}$$

Die Rekursionsgleichung eines klassenspezifischen Gewichtsvektors zum jeweiligen Grad i in α ist mit (8.2.4) auf Seite 108

$$\boldsymbol{v}_i^{t+1} = \boldsymbol{v}_i^t + G^{t+1} \boldsymbol{\Phi} \boldsymbol{c}_i^{t+1} \qquad i \in [0, 1, 2]\ .$$

Dabei ist G^t die Gewichtungsfunktion die angibt, wie stark der Beitrag eines Zustands $\boldsymbol{z}^t$ zum Gesichtsvektor $\boldsymbol{v}_i^t$ sein soll. Der Gesamtgewichtsvektor setzt sich dann aus

$$\boldsymbol{w}^t = \boldsymbol{v}_0^t + \boldsymbol{v}_1^t\,\alpha + \boldsymbol{v}_2^t\,\alpha^2$$

zusammen.

Nun kommen wir zur kalten Nettoinformation, wobei der Zeitindex t weggelassen wird:

$$\begin{aligned}
I_{\mathrm{nk}} &= D_{\mathrm{k}} - r \cdot R_{\mathrm{k}} = (w_i)^2 - r \cdot (w_{ik})^2 \\
&= (v_{i,0} + v_{i,1}\,\alpha + v_{i,2}\,\alpha^2)^2 - r \cdot (v_{ik,0} + v_{ik,1}\,\alpha + v_{ik,2}\,\alpha^2)^2 \\
&= (v_{i,0})^2 + (v_{i,1})^2\alpha^2 + (v_{i,2})^2\alpha^4 + 2\,v_{i,0}v_{i,1}\alpha + 2\,v_{i,0}v_{i,2}\alpha^2 + 2\,v_{i,1}v_{i,2}\alpha^3 \\
&\quad - r\left[(v_{ik,0})^2 + (v_{ik,1})^2\alpha^2 + (v_{ik,2})^2\alpha^4 + 2\,v_{ik,0}v_{ik,1}\alpha + 2\,v_{ik,0}v_{ik,2}\alpha^2 + 2\,v_{ik,1}v_{ik,2}\alpha^3\right] \\
&= (v_{i,0})^2 - r(v_{ik,0})^2 + 2(v_{i,0}v_{i,1} - r\,v_{ik,0}v_{ik,1})\alpha \\
&\quad + \left[(v_{i,1})^2 + 2\,v_{i,0}v_{i,2} - r\big((v_{ik,1})^2 + 2\,v_{ik,0}v_{ik,2}\big)\right]\alpha^2 \\
&\quad + 2\left(v_{i,1}v_{i,2} - r v_{ik,1}v_{ik,2}\right)\alpha^3 + \left((v_{i,2})^2 - r(v_{ik,2})^2\right)\alpha^4 \\
&=: a_0 + a_1\alpha + a_2\alpha^2 + a_3\alpha^3 + a_4\alpha^4
\end{aligned}$$

Die Koeffizienten sind

$$\begin{aligned}
a_0 &= (v_{i,0})^2 - r(v_{ik,0})^2 \\
a_1 &= 2(v_{i,0}v_{i,1} - r\,v_{ik,0}v_{ik,1}) \\
a_2 &= (v_{i,1})^2 + 2\,v_{i,0}v_{i,2} - r\big((v_{ik,1})^2 + 2\,v_{ik,0}v_{ik,2}\big) \\
a_3 &= 2\left(v_{i,1}v_{i,2} - r v_{ik,1}v_{ik,2}\right) \\
a_4 &= (v_{i,2})^2 - r(v_{ik,2})^2
\end{aligned}$$

Wir berücksichtigen nun wieder nur Terme bis zur zweiten Ordnung in α, um eine in α lineare Ableitung zu bekommen:

$$I'_{\mathrm{nk}} = a_0 + a_1\alpha + a_2\alpha^2 + O(\alpha^3)$$

Wir erhalten nach α differenziert:

$$\partial_\alpha I'_{\mathrm{nk}} = a_1 + 2a_2\alpha$$

Damit verschwindet $\partial_\alpha I'_{\mathrm{nk}}$ genau dann, wenn

$$\alpha = -\frac{a_1}{2a_2}$$

ist.

Der Praxistest hat gezeigt, daß dieser Algorithmus zwar prinzipiell funktioniert, aber den gradientenfreien Verfahren mit Liniensuche aus Abschnitt 8.4.2 unterlegen ist. Dies hat zwei Gründe:

1. Die Beschreibung des Zustands in Gleichung 8.3 ist eine Näherung, die nur für kleine α gilt. Daher müssen wir den Bereich, in dem wir α auswerten, auf ein Intervall $[\alpha_{\min}, \alpha_{\max}]$ beschränken. Dies führt uns aber in ein Dilemma: Je kleiner das Intervall $[\alpha_{\min}, \alpha_{\max}]$ ist, um so genauer ist zwar die Approximation, um so länger läuft aber auch der Algorithmus.

2. Der Algorithmus benötig relativ viele Berechnungen, um die *näherungsweise* Position eines Minimums zu bestimmen. Die gradientenfreien Verfahren mit Liniensuche nutzen die einzelnen Funktionsaufrufe effizienter, da sie nicht mit einem genäherten $f(x)$ sondern mit $f(x)$ selbst arbeiten.

Obwohl der PAP-Algorithmus für vorwärtsgerichtete HPNs sehr gut funktioniert, ist seine Verallgemeinerung auf rückgekoppelte HPNs wegen des eben beschriebenen Dilemmas nicht gut nutzbar.

8.4 Optimierung mit direkter Suche

Die Optimierung mit *direkter Suche* bezeichnet Optimierungalgorithmen, die zur Optimierung ausschließlich den Wert $f(\boldsymbol{x})$ der zu optimierenden Funktion auswerten. Insbesondere benötigen diese Algorithmen weder einen Gradienten der Funktion noch muß die Funktion stetig sein. Einen guten Überblick gibt Powell (1998). Die direkte Suche ist auch dann nützlich, wenn wie in unserem Fall

- die Berechnung des Gradienten sehr aufwendig ist und
- die Zahl lokaler Minima groß ist, so daß ein Gradientenabstieg mit großer Wahrscheinlichkeit nur in ein lokales Minimum führen würde.

8.4.1 Algorithmen vom Downhill-Simplex Typ

Algorithmen vom *Downhill-Simplex* Typ werden „von Benutzern geliebt und von Optimierungs-Forschern ignoriert“ – ein Zitat aus Nemhauser et al. (1989). In der Praxis werden diese Algorithmen deshalb so gerne verwendet, weil die einfachen unter ihnen in 100 Zeilen Code zu implementieren sind und sie erstaunlich oft gute Ergebnisse liefern. Sie werden von Optimierungs-Forschern ignoriert, weil man für manche Algorithmen des Downhill-Simplex Typs Funktionen konstruieren kann, bei denen der Algorithmus entweder nicht terminiert oder schon terminiert, bevor er in einem lokalen Minimum angekommen ist. Wir gehen nun auf drei Varianten des Downhill-Simplex Algorithmus ein, die im Rahmen dieser Arbeit getestet wurden.

Die Grundidee der Simplex-Algorithmen ist, einen Simplex im n-dimensionalen Parameterraum solange „bergab“ wandern zu lassen, bis er in einem Minimum der Funktion angekommen ist. Ein Simplex im n-dimensionalen Raum besteht aus $n+1$ Punkten, ihren Verbindungslinien und den durch die Verbindungslinien definierten Polygonflächen. Die verschiedenen Algorithmen unterscheiden sich vor allem darin, mit welchen Operationen sie aus einem gegebenen Simplex einen neuen konstruieren.

8.4.1.1 Downhill-Simplex Algorithmus

Der bekannteste Downhill-Simplex Algorithmus ist der von Nelder und Mead (1965). Die Hauptschritte des Algorithmus sind (siehe Abbildung 8.3):

1. Zunächst zieht man durch den Punkt $\boldsymbol{x}_{\max}$ des Simplex mit dem größten Funktionswert $f_{\max}$ und dem Schwerpunkt des Polygons bestehend aus den restlichen Punkten eine gedachte Gerade. Auf dieser Geraden werden nun weitere Punkte gesucht.
2. Der Punkt $\boldsymbol{x}_{\max}$ wird auf die andere Seite des Polygons gespiegelt (a)
3. Ist am neuen Punkt $\boldsymbol{x}$ die Funktion $f(\boldsymbol{x}) < f_{\max}$, ersetzt der neue Punkt $\boldsymbol{x}$ den alten Punkt $\boldsymbol{x}_{\max}$ und man versucht, den Simplex weiter zu vergrößern (b).

4. Ist aber $f(\boldsymbol{x}) \geq f_{\max}$, wird versucht, den Simplex entlang der gedachten Geraden zu verkleinern (c). Ist auch an diesem Punkt $f(\boldsymbol{x}) \geq f_{\max}$, wird der gesamte Simplex kontrahiert (d). Dann geht es weiter mit 1.

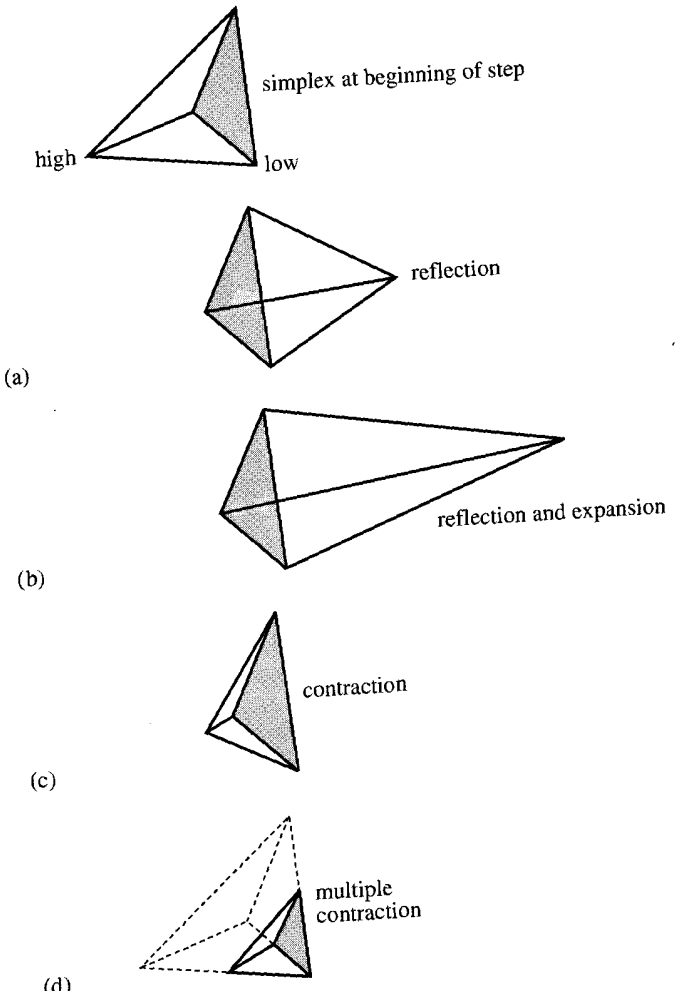


Abbildung 8.3: Schritte des Downhill-Simplex Algorithmus (aus Press et al., 1992)

Die Behandlung der Nebenbedingungen ist bei Algorithmus von Nelder und Mead (1965); Press et al. (1992) Algorithmus etwas problematisch. Es gibt zwei Möglichkeiten, den ursprünglichen Nelder und Mead Algorithmus so zu modifizieren, daß er Nebenbedingungen berücksichtigt:

1. Man führt einen Strafterm ein und setzt den Wert von $f(\boldsymbol{x})$ außerhalb des zulässigen Bereichs auf einen so hohen Wert, daß ein solcher Punkt niemals angenommen wird. Dies hat den Nachteil, daß der Algorithmus nur sehr langsam gegen ein Minimum in einer Ecke des zulässigen Bereichs konvergiert. In unserem Fall sind aber in den Ecken des zulässigen Bereichs gerade die potentiellen Kandidaten für Minima.

2. Die andere Möglichkeit ist, bei der Spiegelung des Punktes mit dem größten Funktionswert den neuen Punkt genau auf die Grenzfläche einer Nebenbedingung zu setzen, falls er diese

Nebenbedingung sonst verletzen würde. Der Nachteil hier ist, daß der Simplex die Tendenz hat, zu entarten.

8.4.1.2 Cobyla

Der Name des *Cobyla-Algorithmus* von Powell (1994) ist aus seiner Vorgehensweise abgeleitet: „Constraint Optimization by linear Approximation". Im Gegensatz zum Simplex-Algorithmus verwendet der Cobyla-Algorithmus ein Näherungsverfahren, um den Punkt zu bestimmen, an dem die Funktion $f(\boldsymbol{x})$ das nächste mal ausgewertet wird. Das Ziel dabei ist, mit möglichst wenigen Funktionsauswertungen ein Minimum der Funktion zu finden. Die wesentlichen Schritte von Cobyla sind:

1. Mit den $n+1$ bekannten Funktionswerten an den Ecken des Simplex wird eine lineare Approximation der Funktion $f(\boldsymbol{x})$ erstellt.

2. Mit Hilfe des gleichnamigen Simplex-Verfahrens aus der linearen Programmierung wird das lineare Minimierungsproblem mit Randbedingungen gelöst. An der Stelle $\boldsymbol{x}_0$, wo die lineare Approximiation der Funktion f ihr Minimum hat, wird der nächste tatsächliche Funktionswert $f_0 = f(\boldsymbol{x}_0)$ berechnet.

3. Ist am Punkt $\boldsymbol{x}_0$ der neue Funktionswert $f(\boldsymbol{x}_0) < f_{\max}$, wird der Punkt mit dem größten Funktionswert $\boldsymbol{x}_{\max}$ aus dem Simplex entfernt und der Punkt $\boldsymbol{x}_0$ in den Simplex aufgenommen. Dann geht es wieder zu Schritt 1.

Außerdem sorgt der Algorithmus dafür, daß der Simplex nicht entartet und er sucht neue Punkte nur in einer Region, in der die lineare Approximiation für glaubwürdig genug erachtet wird.

Die für unsere Optimierung geltenden Nebenbedingungen lassen sich auf natürliche Weise in Cobyla integrieren, da dieser Algorithmus gerade für den Fall der Optimierung mit Nebenbedingungen entwickelt wurde.

8.4.1.3 Verhalten der Downhill-Simplex Algorithmen in unserem Fall

Die numerischen Tests der Downhill-Simplex Algorithmen haben folgende Vorteile (+) und Nachteile (−) ergeben:

Downhill-Simplex Algorithmus:

+ Aufgrund der einfachen Grundidee des Algorithmus ist er leicht zu implementieren.
− Die Nebenbedingungen können nur schwierig berücksichtigt werden.
− Der Algorithmus benötigt sehr viele Funktionsauswertungen, bis er in einem Minimum terminiert.

Cobyla:

+ Der Algorithmus ist für die Optimierung mit Nebenbedingungen ausgelegt.
+ Der Algorithmus kommt bis zur Konvergenz in einem Minimum mit relativ wenigen Funktionsauswertungen aus.
− Die lineare Approximation ist bei unserer (stark) nichtlinearen Kostenfunktion keine gute Näherung.

Aspekte, die beide Algorithmen betreffen:

+ Die Algorithmen kommen ohne Gradienten aus.

– Die Algorithmen benötigen in n Dimensionen einen Simplex mit (mindestens) $n+1$ Punkten, um zu starten. Möchte man verschiedene Knoten eines Netzwerks abwechselnd optimieren, muß man die Funktion $f(\boldsymbol{x})$ vor jedem Start der Algorithmen erst $n+1$-mal auswerten, bevor man beginnen kann.

– Keiner der Algorithmen kann gut mit lokalen Minima umgehen. Durch ihre „Wanderung" durch den Parameterraum schneiden sie, was die Vermeidung von lokalen Minima angeht, aber besser ab als ein normaler Gradientenabstieg.

Wegen der aufgezeigten Nachteile wurden die beiden Alogrithmen nach einigen numerischen Tests nicht mehr weiter zum Training verwendet.

8.4.2 Gradientenfreie Verfahren mit Liniensuche

Die gradientenfreien Verfahren mit Liniensuche beruhen auf dem in Abschnitt 7.3 vorgestellten Algorithmus: In jedem Iterationsschritt des Algorithmus wählt man erst eine zulässige Suchrichtung $\boldsymbol{s}_i$ und führt anschließend auf der Suchrichtung eine Linienminimierung durch. Dann geht es zum nächsten Iterationsschritt.

Das entscheidende Merkmal gradientenfreier Verfahren ist, daß sie zur Bestimmung der Suchrichtung ganz oder fast ganz auf die Berechnung des Gradienten verzichten. Damit verzichtet man auf die genaue Kenntnis der Funktion in unmittelbarer Umgebung eines Punktes, spart dafür aber maximal $2n$ Berechnungen der Kostenfunktion, falls der Gradient numerisch ermittelt werden muß.

Ist man nicht an einem lokalen Minimum der Kostenfunktion, sondern am globalen Minimum innerhalb des zulässigen Bereichs interessiert, nützt einem die genaue Kenntnis der unmittelbaren Umgebung der Funktion in einem Punkt wenig. Denn ein Gradientenabstieg führt im allgemeinen nur zu einem lokalen Minimum. Anstatt daher die $2n$ Funktionsaufrufe für die Gradientenberechnung zu investieren, setzt man sie dazu ein, das globale Verhalten der Funktion im zulässigen Bereich zu erkunden.

8.4.2.1 Wahl der Suchrichtung

Wie bestimmen wir die Suchrichtung ohne die Kenntnis des Gradienten? In der Literatur existieren verschiedene Möglichkeiten, die alle ihre spezifischen Vor- und Nachteile haben.

Achsenparallele Suche: Bei der achsenparallelen Suche startet man mit der Linienminimierung in Richtung der ersten Koordinatenachse. Bei jeder Iteration wählt man die nächste Koordinatenachse als Suchrichtung, wobei man nach n Schritten wieder die erste Koordinatenachse wählt. Um zu entscheiden, ob man in positive oder negative Koordinatenrichtung suchen soll, braucht man nur eine Komponente des Gradienten mit maximal zwei Funktionsaufrufen zu approximieren. Bei komplizierten Gradienten ist dies von Vorteil. Allerdings kann die achsenparallelen Suche bei schlecht konditionierten Problemen sehr langsam konvergieren. Die Konvergenzgeschwindigkeit kann außerdem stark von der Reihenfolge abhängen, in der die Koordinatenrichtungen ausgewählt werden.

Konjugierte Richtungen: Die Idee der konjugierten Richtungen von Powell (siehe Press et al., 1992, S. 412) ist, die Wahl der nächsten Suchrichtung so vorzunehmen, daß möglichst die Minimierung des vorangegangenen Iterationsschritts nicht wieder zunichte gemacht wird. Powell zeigte, daß bei einem quadratischen Minimierungsproblem in n Dimensionen

diese Prozedur in n Schritten ein Minimum findet, sie also quadratisch zum Minimum konvergiert. Der Ansatz geht dabei davon aus, daß die zu minimierende Funktion in der Nähe des Minimums näherungsweise wie eine quadratische Form, also wie eine nach oben geöffnete mehrdimensionale Parabel aussieht.

Stochastische Richtungswahl: Bei einem hochdimensionalen Parameterraum bietet es sich an, die verschiedenen Koordinatenrichtungen nicht in einer systematischen Reihenfolge sondern zufällig auszuwählen. Weiterhin kann man auch zufällige Linearkombinationen der Koordinatenachsen als Suchrichtungen wählen. Auf diese Weise verhindert man eine systematische Bevorzugung bestimmter Raumrichtungen. Werden nun mehrere Minimierungsläufe vom selben Startpunkt $\boldsymbol{x}_0$ aus gestartet, läßt sich sofort sehen, ob die Minimierung in verschiedenen Minima endet.

Von diesen drei Verfahren hat die stochastische Richtungswahl (zumindest theoretisch) einen entscheidenden Vorteil: Man kann unter relativ schwachen Voraussetzungen beweisen, daß die stochastische Richtungswahl mit der Wahrscheinlichkeit eins das *globale* Minimum der Funktion findet, wenn die Minimierung beliebig lange läuft (Nemhauser et al., 1989, S. 645). Wir werden noch sehen, daß diese Art der Richtungswahl für unser Problem geeignet ist.

8.4.2.2 Einfluß der Nebenbedingungen

Wie schon in Abschnitt 7.2.6 auf Seite 93 beschrieben, gelten für die Elemente der Matrix $\boldsymbol{H}$ die Nebenbedingungen

$$\sum_{i=0}^{2^d-1} [\boldsymbol{H}]_{ij} = 1 \; ; \qquad j \in [0, 2^d - 1]$$

$$[\boldsymbol{H}]_{ij} \geq 0 \qquad \text{und} \qquad [\boldsymbol{H}]_{ij} \leq 1$$

$$\sum_{j=0}^{2^d-1} \boldsymbol{H}_{ij} = 2^{d-d'} \; ; \qquad i \in [0, 2^{d'} - 1]$$

wobei d' die Zahl der Ausgangsbits ist, auf die projiziert wird.

Bedingung 1. hat zur Folge, daß von einem zulässigen Punkt aus nur bestimmte Suchrichtungen erlaubt sind. Bedingung 2. legt fest, innerhalb welches Intervalls auf der Suchrichtung gesucht werden darf. Ist Bedingung 3. aktiv, werden sowohl die Suchrichtung als auch das zulässige Intervall weiter eingeschränkt. Da in unserem Fall die Suchregion konvex ist, läßt sich bei gegebener Suchrichtung genau ein abgeschlossenes Intervall bestimmen, in dem alle zulässigen Punkte auf dieser Suchrichtung liegen. Dies verdeutlicht Abbildung 8.4.

In numerischen Experimenten hat sich gezeigt, daß eine Minimierung mit dem Verfahren der konjugierten Richtungen bei unseren speziellen Nebenbedingungen nicht zum Ziel führt. Das Verfahren der konjugierten Richtungen ist darauf spezialisiert, möglichst schnell gegen ein Minimum zu konvergieren, das näherungsweise einer quadratischen Form (mit positiv semidefinierter Hesse-Matrix) entspricht. Unsere Kostenfunktion ist aber eine quadratische Form mit negativ semidefinierter bzw. indefiniter Hesse-Matrix, die außerdem auf den Einheitswürfel im Parameterraum beschränkt ist. Die meisten Minima liegen in den Ecken dieses Einheitswürfels und sind damit gerade nicht von der Form, die das Verfahren der konjugierten Richtungen benötigt.

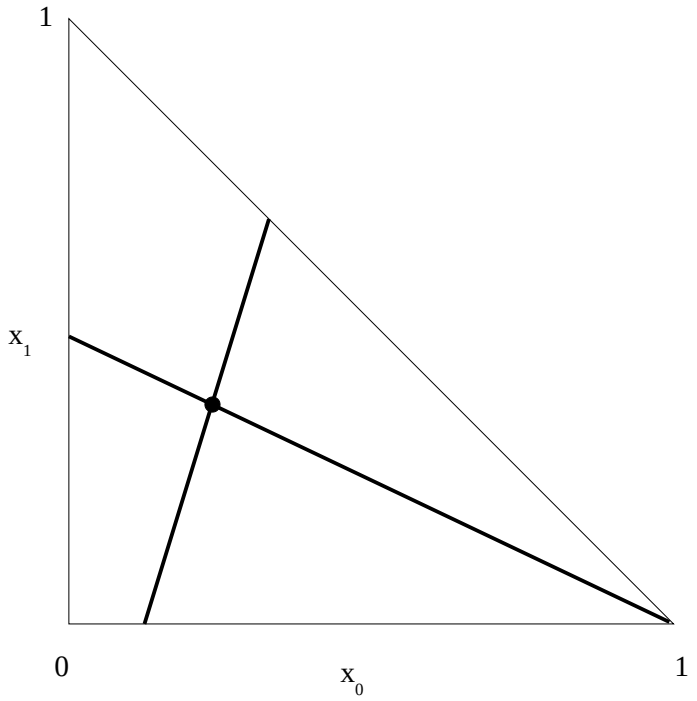


Abbildung 8.4: Behandlung der Nebenbedingungen. Als Beispiel ist hier angenommen, die Zahl der Parameter sei drei und die Nebenbedingungen seien $x_0+x_1+x_2 = 1$ und $x_0 \geq 0$, $x_1 \geq 0$, $x_2 \geq 0$. Gezeigt sind nur die beiden Parameter x_0 und x_1. Durch die Nebenbedingungen ergibt sich, daß sich der Punkt (x_0, x_1, x_2) innerhalb der x_0, x_1-Ebene nur innerhalb des konvexen Bereichs $x_0 \geq 0$, $x_1 \geq 0$ und $x_1 \leq 1 - x_0$ bewegen darf. Wir befinden uns nun an einem bestimmten Punkt innerhalb des konvexen Bereichs. Dann läßt sich zu jeder Geraden durch diesen Punkt berechnen, wo diese Gerade die Grenzen des konvexen Bereichs trifft. Die zulässigen Punkte liegen nun zwischen den beiden Punkten, an denen die Gerade die Grenzen des konvexen Bereichs durchstößt.

8.4.2.3 Wahl der Startparameter

Bei der Richtungsoptimierung mit Liniensuche gibt es verschiedene Möglichkeiten der Initialisierung der Knotenmatrix. Im Prinzip ist jeder Punkt im zulässigen Bereich des Parameterraums ein gültiger Startpunkt für die Optimierung. Es gibt aber günstige und weniger günstige Startpunkte.

Das Hauptaugenmerk muß darauf gerichtet sein, sich mit dem Startpunkt nicht gleich in oder in die Nähe eines lokales Minimums zu begeben. Auch wenn die stochastische Richtungswahl lokalen Minima recht gut wieder entkommen kann, ist es vorteilhaft, wenn der Startpunkt möglichst „neutral" ist, d. h. wenn er möglichst von allen lokalen Minima gleich weit entfernt ist.

Zufälliger Punkt: Ein zufällig gewählter Startpunkt kann naturgemäß ein guter oder ein schlechter Startpunkt sein.

Permutationsmatrix: Eine Permutationsmatrix als Startpunkt der kontinuierlichen Optimierung ist im allgemeinen keine gute Wahl. Dies hat den Grund, das viele Permutationsmatrizen im Sinne des Kostenfunktion lokale Minima sind. Und wie eben schon beschrieben ist es ungüstig, die Optimierung in einem solchen Minimum zu beginnen.

Gleichverteilung: Die beste Wahl für den Startpunkt ist eine gleichverteilte stochastische Hyperpermutationsmatrix. Ein Knoten mit einer gleichverteilten stochastische Hyperpermutationsmatrix emittiert für ein beliebiges Eingangsmuster alle Ausgangsmuster mit gleicher Wahrscheinlichkeit. Die Gleichverteilung als Startpunkt ist im Parameterraum von allen Ecken des zulässigen Bereichs und damit von potentiellen lokalen Minima gleich weit entfernt.

8.4.2.4 Die Linienoptimierung

Jeder Algorithmus zur Richtungsoptimierung besteht aus der Richtungswahl und der Linienoptimierung, (siehe Press et al., 1992, S. 394 ff.). Beim PAP-Algorithmus für vorwärtsgerichtete Netze haben wir in Abschnitt 7.3.2 auf Seite 97 gesehen, daß die Kostenfunktion, beschränkt auf eine Linie, eine quadratische Funktion war. Daher konnten wir das Minimum dieser Funktion direkt berechnen. Bei rückgekoppelten Netzen haben wir es bei der Liniensuche mit der Minimierung einer eindimensionalen Funktion zu tun, deren Grad wir im allgemeinen nicht kennen.

Bei der Linienminimierung im Rahmen einer stochastischen Richtungswahl muß die Linienminimierung nicht sehr genau sein. Es kommt nur darauf an, möglichst schnell in die Nähe eines Minimums der eindimensionalen Funktion zu kommen. Dabei soll nicht zu viel Rechenzeit verbraucht werden, da ja die stochastische Richtungswahl sehr viele Linienminimierungen erfordert.

Ein Standardverfahren, das Minimum einer eindimensionalen Funktion zu finden, die im Minimum nicht einer Parabel ähnelt, ist die *Goldene-Schnitt-Suche (Golden Section Search)* (siehe Press et al., 1992). Die Goldene-Schnitt-Suche startet mit dem Wissen, daß ein Minimum mit drei Punkten (a, b, c) eingegrenzt ist. Das heißt, es sind drei Punkte $a < b < c$ bekannt, so daß $f(b) < f(a)$ und $f(b) < f(c)$. Dann startet folgender Prozeß (Abbildung 8.5):

1. Man wählt einen neuen vierten Punkt x im *größeren* der beiden Intervalle $[a, b]$ und $[b, c]$, um die Zahl der Schritte im schlimmsten Fall zu minimieren.

2. Nehmen wir nun an, das größere Intervall sei $[b, c]$ und wir wählen einen neuen Punkt x im Intervall $[b, c]$, an dem $f(x)$ berechnet wird. Ist $f(b) < f(x)$, sind die drei neuen

begrenzenden Punkte (a, b, x), andernfalls sind die drei neuen Punkte (b, x, c). Der mittlere Punkt ist immer der Punkt mit dem zugehörigen kleinsten Funktionswert, der bis jetzt gefunden wurde. Dann geht es weiter mit Schritt 1., bis sich die beiden äußeren Punkte genügend genähert haben.

Um zu erreichen, daß das Verhältnis der Größen der beiden Intervalle bei jedem Schritt gleich bleibt, wählt man die Längen der beiden Intervalle bezogen auf die Länge des Intervalls $[a, c]$ im Verhältnis $(3 - \sqrt{5})/2 \approx 0.38$ zu $(\sqrt{5} - 1)/2 \approx 0.62$, was gerade dem Goldenen Schnitt entspricht. Da die Länge des Intervalls $[a, c]$ in jedem Schritt nur noch 0.62 mal so lang ist, wie im vorherigen Schritt, hat das Intervall $[a, c]$ schon nach 10 Schritten nur noch 0.8% der urprünglichen Länge. Läßt man den Algorithmus nur einen Schritt lang laufen, bekommt man den kleinsten der drei Punkte $f(a)$, $f(b)$ und $f(c)$ geliefert.

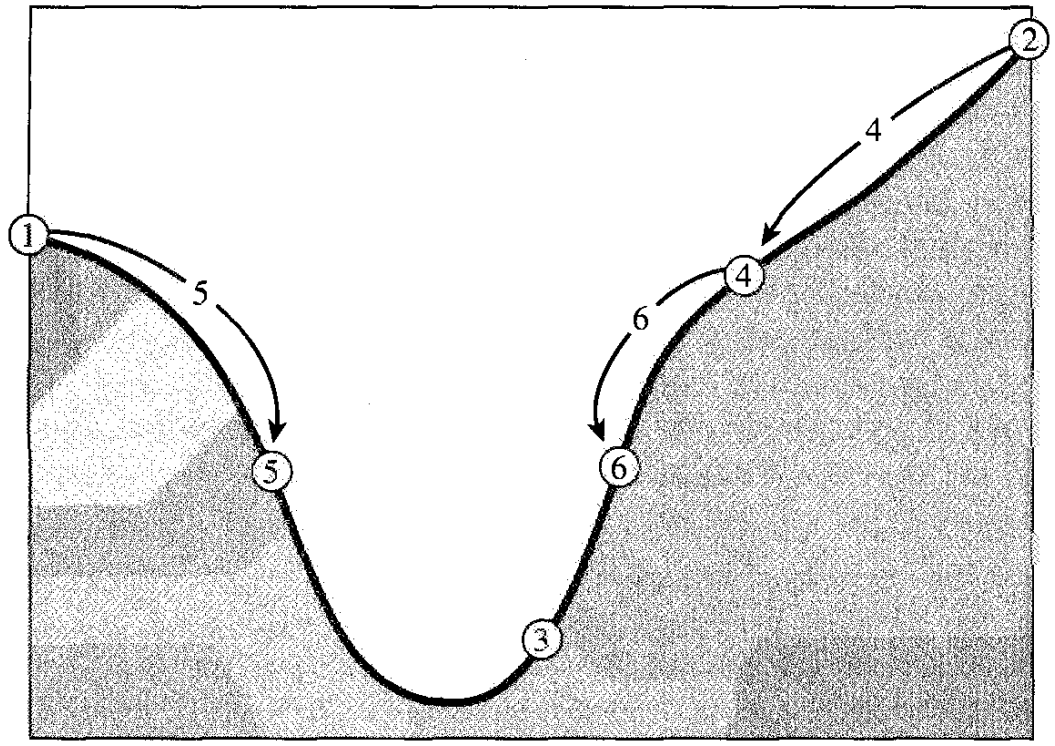


Abbildung 8.5: Einige Schritte der Goldenen-Schnitt-Suche. Das durch die Punkte 1, 2 und 3 begrenzte Minimum wird durch die folgenden Schritte immer weiter eingegrenzt (aus Press et al., 1992).

8.4.3 Simulated Annealing

Simulated Annealing ist eine bekannte Methode zur (näherungsweisen) Lösung kombinatorischer Probleme. Das Konzept des Simulated Annealing ist aus der Physik entlehnt: Kühlt man eine Flüssigkeit langsam immer weiter ab, wird sie irgendwann zuerst teilweise und dann vollständig in die feste Phase übergehen. Je langsamer die Flüssigkeit abgekühlt wird, desto niedriger wird im allgemeinen die Gesamtenergie der festen Phase später sein. Im Sinne der Optimierung ist dieser Prozeß die Suche nach globalen Minima im Raum der mikroskopischen Konfigurationen. Die Wahrscheinlichkeitsverteilung, die die Energien eines Systems im thermi-

schen Gleichgewicht beschreibt, ist die *Boltzmann-Verteilung*:

$$P(E) \sim \exp(-E/kT)$$

Die Wahrscheinlichkeit, das System mit einer bestimmten Energie vorzufinden, nimmt zu höheren Energien exponentiell ab.

Zur Lösung eines kombinatorischen Optimierungsproblems wird dieses physikalische Bild auf das Optimierungsproblem übertragen (siehe Press et al., 1992; Powell, 1994): Sei $\boldsymbol{x}_i \in \mathcal{K}$ eine mögliche Konfiguration der Parameter aus der Menge aller möglichen Parameterkonfigurationen $\mathcal{K}$. Sei außerdem $\boldsymbol{x}(n)$ die zum Zeitschritt n aktuelle Konfiguration und T die aktuelle Temperatur. Dann sind die Schritte des Simulated Annealing:

1. Wähle zufällig ein $\boldsymbol{x}_j \in \mathcal{K}$, wobei die Wahrscheinlichkeit, bei gegebenem $\boldsymbol{x}_i$ ein $\boldsymbol{x}_j$ zu wählen, durch die Elemente w_{ij} der dünn besetzten Matrix $\boldsymbol{W}$ gegeben ist.

2. Falls $f(\boldsymbol{x}_j) \leq f(\boldsymbol{x}_i)$, setze $\boldsymbol{x}(n+1) = \boldsymbol{x}_j$.

3. Andernfalls, wenn $f(\boldsymbol{x}_j) > f(\boldsymbol{x}_i)$, setze $\boldsymbol{x}(n+1) = \boldsymbol{x}_j$ bzw. $\boldsymbol{x}(n+1) = \boldsymbol{x}_i$ mit der Wahrscheinlichkeit:

$$\exp\left[-(f(\boldsymbol{x}_j) - f(\boldsymbol{x}_i))/T\right] \quad \text{bzw.} \quad 1 - \exp\left[-(f(\boldsymbol{x}_j) - f(\boldsymbol{x}_i))/T\right] \ .$$

4. Ist das System bei der Temperatur T ins thermodynamische Gleichgewicht gekommen, senke die Temperatur T.

5. Ist die Temperator klein genug, breche den Prozeß ab, sonst gehe zu Schritt 1.

Der entscheidende Punkt bei diesem Prozeß ist der Schritt 3. Hier werden auch Konfigurationen für den nächsten Zeitschritt zugelassen, deren Funktionswert größer als der bisherige Funktionswert ist. Dieser Schritt ermöglicht es dem Simulated Annealing, lokalen Minima einer Funktion zu entkommen. Man kann zeigen, daß für eine genügend große Wahl von $T(0)$ und genügend langsame Abkühlung mit der Wahrscheinlichkeit eins ein globales Minimum der Funktion gefunden wird.

In der Praxis findet das Simulated Annealing bei einem RHPN oft recht schnell ein relativ gutes Minimum. Dies liegt u. a. auch daran, daß das Simulated Annealing auf den deterministischen Knotentabellen operieren kann und daher die Verarbeitung der Eingangsdaten schnell vor sich geht. Um aber *sehr* dicht an das globale Minimum einer Funktion heranzukommen, sind u. U. viele Iterationen nötig, die sich in einer langen Laufzeit des Minimierungsprozesses niederschlagen.

Die Initialisierung einer Knotentabelle besteht beim Simulated Annealing im zufälligen Wählen der Tabelleneinträge. Bei jedem Knoten wird eine andere Tabelle „ausgewürfelt“.

9 Ein Netzwerk aus mehreren Knoten

Bis jetzt haben wir bei der Optimierung nur einen einzigen Knoten betrachtet. Wir gehen nun zur Optimierung eines Netzwerks von Knoten über. Prinzipiell ändert sich dadurch nichts an der Berechnung der statistischen Größen und an der Kostenfunktion. Es gibt aber verschiedene Varianten zur Optimierung eines Netzwerks, die wir jetzt vorstellen.

9.1 Die Statistikbewertung

Wir können in die Kostenfunktion sowohl die Statistik eines einzelnen Knotens als auch mehrerer Knoten gleichzeitig einfließen lassen. Auch hier geht es wieder darum, die Zahl der relevanten Unterschiede zu maximieren.

9.1.1 Optimierung mit lokaler Statistik

Angenommen, wir wollen das in Abbildung 9.1 gezeigte rückgekoppelte Hyperpermutationsnetzwerk optimieren. Das Netz hat eine Baumstruktur und es gibt nur Rückkopplungen innerhalb eines Knotens. Wir wissen schon, wie wir einen einzelnen Knoten optimieren: Wir maximieren am Ausgang des Knotens bei gegebenen Eingangsdaten die heiße Information. Ganz analog können wir bei einem Netzwerk vorgehen, bei dem es nur Rückkopplungen innerhalb eines Knotens gibt: Wir optimieren einen Knoten im Netz lokal, indem wir die heiße Information an dem Ausgang des Knotens maximieren. Dies impliziert, daß wir die Knoten vom Netzwerkeingang hin zum Netzwerkausgang optimieren müssen. Sonst würden wir versuchen Knoten zu optimieren, deren Vorgängerknoten noch gar keine brauchbaren Daten liefern.

Die Optimierung eines RHPN, in dem die Rückkopplungen nur knotenweise vorkommen, kann also lokal geschehen:

1. Optimiere den ersten Knoten der ersten Spalte. Da dieser Knoten am Eingang des Netzwerks ist, „sieht“ er einen Ausschnitt der hereinkommenden Daten. Ist die Optimierung beendet, maximiert der Knoten aus den ihm zur Verfügung stehenden Daten die heiße Information an seinem Ausgang.

2. Optimiere den nächsten Knoten der ersten Spalte. Da dieser Knoten einen anderen Ausschnitt der Eingangsdaten als der erste Knoten bekommt, wird der zweite Knoten im allgemeinen andere statistische Eigenschaften der Eingangsdaten nutzen als der erste Knoten, um die heiße Information an seinem Ausgang zu maximieren.

3. Optimiere die restlichen Knoten der ersten Spalte.

4. Optimiere alle Knoten der zweiten Spalte ganz analog zu den Knoten der ersten Spalte. Da nun die erste Spalte schon optimiert ist, sind die Knoten der zweiten Spalte in der „ersten Spalte des noch untrainierten Teil des Netzes“.

5. Optimiere alle verbleibenden Spalten.

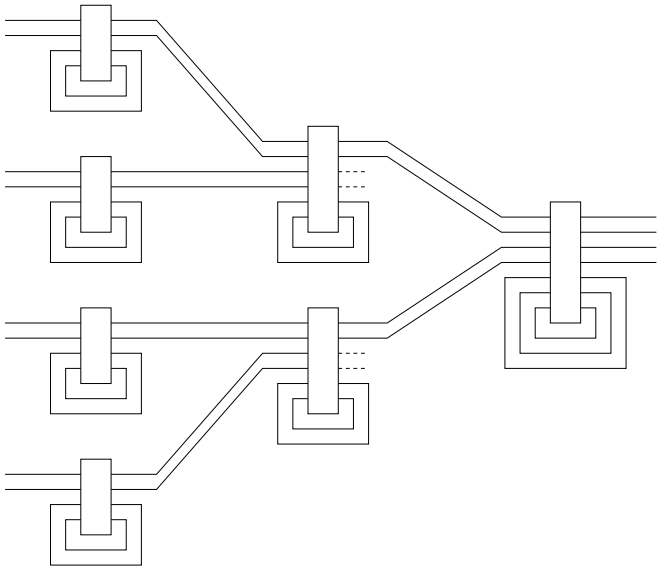


Abbildung 9.1: Baumartige Netzstruktur mit Rückkopplungen. Typische Netzstruktur, um einen zeitlichen Datenstrom in mehrere Klassen zu klassifizieren. Der Netzeingang hat eine Breite von acht Bit. Viele Sensoren liefern ihre Daten mit acht Bit Auflösung. Der Netzausgang hat eine Breite von vier Bit. Dies ergibt 16 mögliche Ausgangszustände. Mit dieser Struktur könnten wir also maximal 16 Klassen voneinander unterscheiden. Im allgemeinen ist es aber besser, mehrere Ausgangszustände pro Klasse zu verwenden, weil dann die klassenspezifischen Rückschlußwahrscheinlichkeiten feiner aufgelöst werden.

Nachdem die erste Spalte optimiert worden ist, verdichtet sie schon zu einem gewissen Teil die Information, die in den Eingangsdaten steckt. Darauf setzt die nächste Spalte des Netzes auf, um die Information wiederum auf noch weniger Bits zu komprimieren. Der Prozeß wird fortgesetzt, bis man an der letzten Spalte ankommt, wo die Information auf die Zahl der Ausgangsbits des Netzwerks verdichtet worden ist.

Ein Nachteil dieser Vorgehensweise ist, daß die Knoten einer Spalte unabhängig voneinander trainiert werden. Es kann also passieren, daß zwei verschiedene Knoten auf ähnliche statistische Merkmale in ihren Eingangsdaten trainiert werden. Konkret bedeutet dies, daß bei einem Klassifikationsproblem beide Knoten ähnliche Fehlklassifikationen machen werden, weil sie ähnliche statistische Merkmale gelernt haben.

Effektiver ist es, die verschiedenen Knoten einer Spalte auf *unterschiedliche* statistische Merkmale der Eingangsdaten zu trainieren. Was ein Knoten in einer Spalte „schon kann“, braucht ein anderer Knoten derselben Spalte nicht auch zu erlernen. Vielmehr sollte ein zu trainierender Knoten das lernen, was die anderen Knoten *noch nicht* können. Um dies zu erreichen, erweitern wie die Statistik von einem auf mehrere Knoten.

9.1.2 Optimierung mit erweiterter Statistik

Optimieren wir mit erweiterter Statistik, bedeutet das die Einbeziehung mehrerer Knoten in die Kostenfunktion, um die Güte mehrerer Knoten gleichzeitig beurteilen zu können. Die Idee dabei ist folgende: Wie wir schon in Abschnitt 4.4.1.1 gesehen haben, läßt sich aus mehreren Leitungsbündeln des Netzwerks ein neues Leitungsbündel kombinieren. Bilden wir nun aus den Ausgängen der Knoten einer Spalte ein kombiniertes Leitungsbündel, so können wir die verschiedenen Knoten gemeinsam und damit die Ausgangsstatistik der ganzen Spalte auf einmal

erfassen. Wir bilden also jetzt nicht mehr die Statistikmatrizen der einzelnen Knoten, sondern die Statistikmatrix der gesamten Spalte. Aus dieser Gesamtstatistik berechnen wir dann die heiße Information und erhalten so die Gütefunktion für die gesamte Spalte.

Nun werden die Knoten in Abhängigkeit voneinander optimiert: Angenommen, ein Knoten der Spalte kann zwei Muster aus verschiedenen Klassen auseinanderhalten. Lernt ein zweiter Knoten der Spalte, die gleichen Muster zu trennen, verbessert dies die Gütefunktion nicht. Lernt der zweite Knoten dagegen, *andere* Muster voneinander zu trennen als der erste Knoten, verbessert sich dadurch die Gütefunktion.

9.1.2.1 Statistik mehrerer Knoten

Wie sieht eine spaltenübergreifende Statistikmatrix aus? Eine Statistikmatrix ist ja nichts anderes als die Matrix, die sich ergibt, wenn wir k klassenspezifische Gewichtsvektoren als Spalten einer Matrix schreiben:

$$\boldsymbol{W} = (\boldsymbol{w}_0, \boldsymbol{w}_1, \ldots, \boldsymbol{w}_{K-1})$$

Nun hat jeder Knoten n sein eigenes Leitungsbündel $\boldsymbol{\kappa}^{(n)}$ am Knotenausgang, woraus wir normalerweise seinen eigenen Satz von klassenspezifischen Gewichtsvektoren $\boldsymbol{w}_0^{(n)}, \ldots, \boldsymbol{w}_{K-1}^{(n)}$ bestimmen.

Nun bilden wir stattdessen erst die Kombination $\boldsymbol{\kappa}$ der einzelnen Leitungsbündel $\boldsymbol{\kappa}^{(n)}$, und zwar mit Hilfe von Definition 16 auf Seite 44:

$$\boldsymbol{\kappa} = \boldsymbol{\kappa}^{(n-1)} \otimes \boldsymbol{\kappa}^{(n-2)} \otimes \cdots \otimes \boldsymbol{\kappa}^{(0)} = \bigotimes_{k=n-1}^{0} \boldsymbol{\kappa}^{(k)}$$

Der Zustandsvektor $\boldsymbol{\kappa}$ kombiniert jetzt die Einzel-Zustandsvektoren $\boldsymbol{\kappa}^{(n)}$ zu einem spaltenübergreifenden Zustandsvektor $\boldsymbol{\kappa}$, dessen Wert bei gegebenen Eingangsdaten von allen Knotentabellen der Spalte abhängt. Nun bilden wir beim Training die klassenspezifischen Gewichtsvektoren $\boldsymbol{w}_k$ von $\boldsymbol{\kappa}$ bzw. die Statistikmatrix $\boldsymbol{W}$, um daraus die Kostenfunktion zu berechnen.

9.1.2.2 Statistik der globalen Netzfunktion

Wenn wir schon in die Kostenfunktion die Gewichtsvektoren einer ganzen Spalte eingehen lassen, könnten wir auch gleich das *gesamte* Netzwerk bewerten, indem wir zur Bewertung der globalen Netzfunktion die Statistik der *letzen* Spalte des Netzes zur Berechnung der Kostenfunktion heranziehen.

Da wir ja eigentlich das gesamte Netzwerk optimieren wollen, ist gerade die Kostenfunktion der *globalen* Netzfunktion diejenige Kostenfunktion, die wir minimieren sollten. Denn wie wir schon gesehen haben, ist es im allgemeinen nicht das gleiche, ob wir die einzelnen Koten oder ihre Gesamtheit bewerten. Trotzdem hat die globale Netzfunktion auch einen Nachteil. Da wir alle Knoten „auf einmal“ optimieren, umfaßt der Parameterraum alle Knotentabellen gleichzeitig und ist damit viel größer, als wenn wir immer nur einen oder einige wenige Knoten auf einmal optimieren. Zu der gleichzeitigen Optimierung mehrerer Knoten kommen wir noch in Abschnitt 9.2.2.

Optimieren wir das gesamte Netzwerk, können wir ausnutzen, daß bei jedem Iterationsschritt die Fehlerrate bekannt ist. Wir können ja mit der kalten Nettoinformation erreichen, daß jeder Zustand möglichst „rein“ wird, also möglichst nur Daten einer einzigen Klasse enthält

(siehe Anhang 7.2.5). Dies kann aber manchmal auf Kosten der Fehlerrate gehen. Bilden wir nun eine Linearkombination aus kalter Nettoinformation I_{nk} und Fehlerrate R

$$\alpha\, I_{\mathrm{nk}} + (1-\alpha)\, R\ ,$$

wobei die kalte Nettoinformation nur schwach gewichtet wird ($\alpha \ll 1$), werden die Ausgangszustände der Netzes möglichst rein, solange dadurch die Fehlerrate nicht verschlechtert wird.

9.2 Die Optimierungsreihenfolge der Knoten

In Abhängigkeit von der Statistikbewertung gibt es verschiedene Möglichkeiten, die Knoten zu optimieren. Sie können sequentiell oder parallel optimiert werden, auch eine Kombination aus sequentieller und paralleler Optimierung ist machbar.

9.2.1 Sequentielle Optimierung

Bei der *sequentiellen Optimierung* werden die einzelnen Knoten des Netzwerks *nacheinander* optimiert. Es wird also erst ein Knoten fertig trainiert, bevor der nächste zu optimierende Knoten an die Reihe kommt. Um einen Knoten nach dem anderen komplett trainieren zukönnen, müssen zwei Voraussetzungen erfüllt sein:

1. Zwischen dem zu optimierenden Knoten und dem Netzwerkeingang darf es keinen Pfad geben, der durch einen untrainierten Knoten verläuft. Denn sonst würden wir versuchen einen Knoten auf der Basis von nicht optimalen Eingangsdaten des Knotens zu optimieren.

2. Zwischen dem zu optimierenden Knoten und dem Leitungsbündel, an dem die Statistik für die Kostenfunktion gesammelt wird, darf es ebenfalls keinen Pfad geben, der durch einen untrainierten Knoten verläuft. Denn ein untrainierter Knoten vernichtet im allgemeinen heiße Information, so daß die Güte des zu optimierenden Knotens durch den dazwischenliegenden untrainierten Knoten verfälscht wird.

Ob wir die Knoten eines Netz überhaupt sequentiell optimieren können, hängt also von der Topologie des Netzes ab. Die Topologie des Netzwerks muß so sein, daß auf allen Pfaden vom Netzwerkeingang zum Leitungsbündel, an dem die Statistik abgenommen wird, nur ein einziger untrainierter Knoten ist. Sonst „vermischt“ man die Güten mehrerer untrainierter Knoten und kann einen einzelnen Knoten nicht mehr trainieren.

9.2.1.1 Der Einfluß der Netzwerktopologie

Die Art der Netztopologie hat entscheidenden Einfluß darauf, wie wir ein RHPN optimieren können. Ein Netzwerk kann z. B. baumartig oder wie ein Verbindungsnetz aufgebaut sein. Die Rückkopplungen können nur innerhalb eines Knotens oder zwischen verschiedenen Knoten vorkommen. Manche Topologien erzwingen ein bestimmtes Vorgehen bei der Netzoptimierung. Tabelle 9.1 zeigt verschiedene Möglichkeiten zur Optimierung bei gegebener Topologie und der Art der Rückkopplungen.

Können wir die Knoten lokal oder erweitert lokal optimieren, kann dies sequentiell, also Knoten für Knoten geschehen. Ist dagegen eine globale Optimierung nötig, müssen wir alle Knoten parallel optimieren.

Topologie	Rückkopplungen		
	knotenlokal	spaltenlokal	allgemein
baumartig	lokal	erweitert lokal	global
allgemein	global	global	global

Tabelle 9.1: Optimierungsmöglichkeiten bei gegebener Netztopologie. Strenggenommen sind die Rückkopplungen ein Teil der Topologie und damit nicht von ihr separierbar. Hier sind mit Rückkopplungen (wie sonst auch) die Leitungen gemeint, die den Ausgang eines Knotens in einer Spalte i mit dem Eingang eines Knotens in einer Spalte $j \leq i$ verbinden. „Knotenlokal" bzw. „spaltenlokal" bedeutet, daß die Rückkopplungen nur innerhalb eines Knotens bzw. innerhalb einer Spalte des Netzes verlaufen. Je nach Topologie und Art der Rückkopplungen *kann* man das Netzwerk mit lokaler Statistik bzw. *muß* man es mit globaler Statistik optimieren.

9.2.2 Parallele Optimierung

Für ganz allgemeine Netze gilt die Voraussetzung, die zur sequentiellen Optimierung nötig ist, nicht mehr, da in einem allgemeinen Netz jede beliebige Vernetzung erlaubt ist. Bei der *parallelen Optimierung*, für die keine Voraussetzungen gemacht werden müssen, werden mehrere oder alle Knoten gleichzeitig trainiert. Dies bedeutet, daß der Optimierungsalgorithmus als Satz von Variablen die Tabellen von mehreren Knoten gleichzeitig hat.

9.2.2.1 Optimierung auf dem gesamten Parameterraum

Stellen wir uns ein Netz mit K Knoten vor. Jeder Knoten habe n_k zu optimierende Parameter. Dann ist die Optimierungsstrategie für das Gesamtnetz prinzipiell ganz einfach:

1. Lasse in die Kostenfunktion die Gesamtstatistik, also die Statistik am Netzwerkausgang, eingehen.

2. Vereinige die Parameterräume der einzelnen Knoten zu einem Gesamtparameterraum P der Dimension $\sum_{k=0}^{K-1} n_k$.

Nun müssen wir „nur noch" das Minimum der Kostenfunktion im zulässigen Bereich von P finden. Für kleine Netze führt dieser Weg auch schnell zum Ziel.

Problematisch bei diesem Vorgehen sind zwei Punkte:

1. Für Netze mit vielen Knoten wird die Dimension $\sum_{k=0}^{K-1} n_k$ des Gesamtparameterraums irgendwann sehr groß. Die meisten Minimierungsalgorithmen skalieren aber superlinear, so daß die Optimierung überproportional länger dauert.

2. Ein globaler Parameterraum impliziert, daß eine Bewegung des Punktes im Parameterraum im allgemeinen die gesamte Netzwerkkonfiguration ändert. Gerade bei den stochastischen Optimierverfahren wie Simulated Annealing oder der stochastischen Richtungswahl sollten im Laufe der Optimierung die Sprünge des Punktes im Parameterraum aber immer kleiner werden. Denn wenn die meisten Knoten schon recht gut optimiert sind, wäre ein Sprung in eine beliebige andere Netzkonfiguration, die *besser* ist, extrem unwahrscheinlich. Daher sollte man die Sprungrichtungen im Parameterraum beschränken, d. h. nur Sprünge in ähnliche Konfigurationen zulassen.

Durch einen Gesamtparameterraum erhalten wir also eine Freiheit, die uns gerade bei den stochastischen Optimierverfahren gar nichts nützt. Es gibt aber die Möglichkeit, diese Problematik zu umgehen, indem wir auf Unterräumen des Gesamtparameterraums optimieren.

9.2.2.2 Zyklische Optimierung auf Unterräumen des Parameterraums

Die Grundidee der zyklischen Optimierung auf Unterräumen des Parameterraums ist dieselbe wie bei der Lösung des Travelling-Salesman-Problems mit Simulated Annealing. Dort ändert man pro Iteration den Weg durch die Städte nur *ein wenig*. Beispielsweise dreht man die Richtung eines Weges zwischen zwei Städten herum. Damit Simulated Annealing erfolgreich angewendet werden kann, muß man nämlich eine (schwache) Voraussetzung erfüllen: Die Wahrscheinlichkeit, von einem Punkt i zu einem Punkt j im Parameterraum zu springen, sollte bei gegebenem i für die meisten j null sein (Powell, 1994). Dasselbe Prinzip werden wir nun auch auf den Parameterraum an.

Der Parameterraum zerfällt auf natürliche Weise in die Unterräume der einzelnen Knoten, aus denen wir ihn im vorigen Abschnitt zusammengesetzt haben. Pro Iteration werden wir nun den Punkt im gesamten Parameterraum nur noch innerhalb eines Knoten-Unterraums bewegen. Pro Iterationsschritt eines Optimierverfahrens ist es also jetzt nur noch erlaubt, die Knotentabelle *eines* Knotens zu verändern.

Was wir eben für den gesamten Parameterraum festgestellt haben, gilt bei Knoten mittlerer Größe auch schon für den Knoten-Unterraum: Auch dort ist es extrem unwahrscheinlich, durch eine Änderung der gesamten Knotentabelle die Kostenfunktion zu verringern. Daher schränken wir Bewegung des Punktes im Knoten-Unterraum weiter ein: Nur ein Tabelleneintrag darf pro Iterationsschritt verändert werden. Dadurch schränken wir den Knoten-Unterraum pro Iterationsschritt auf einen Unterraum bzw. eine Koordinatenrichtung ein, je nachdem, ob wir die Tabelleneinträge als Bitstrings oder als ganze Zahlen interpretieren.

Die parallele Optimierung eines Netzwerk mit mehreren Knoten findet also folgendermaßen statt:

1. Wähle einen Knoten des Netzwerks aus.
2. Wähle einen Tabelleneintrag des Knotens aus.
   - Beim Simulated Annealing: Ändere den Tabelleneintrag und nehme die Änderung gegebenenfalls entsprechend der Regeln des Simulated Annealing an.
   - Bei der stochastischen Richtungswahl: Wähle eine Richtung innerhalb des durch den Tabelleneintrag festgelegten Unterraums und führe auf der gewählten Richtung eine Linienminimierung durch.
3. Gehe zu Schritt 1.

Wenn eine bestimmte Zahl von Iterationen keine Verringerung der Kostenfunktion mehr eingetreten ist, wird die Optimierung abgebrochen.

9.3 Die Netzwerktopologie

Wir wissen jetzt, wie wir ein rückgekoppeltes Hyperpermutationsnetzwerk bei gegebener Netzwerktopologie optimieren können. Wie finden wir aber die beste Topologie für eine bestimmte Mustererkennungsaufgabe? Diese Frage ist ähnlich schwierig zu beantworten wie die Frage nach der optimalen Topologie eines neuronalen Netzwerks. Daher gibt es noch keinen Optimierungsalgorithmus, der neben der Opimierung eines Hyperpermutationsnetzwerks auch gleich noch die beste Netzwerktopologie findet.

Meistens ist man in der Wahl einer rückgekoppelten Netzwerktopologie durch äußere Randbedingungen eingeschränkt. Haben wir z. B. , wie später in unserem Praxisbeispiel, s Sensoren,

die jeweils b Bit breite Datenleitungen haben, gibt es für den Netzwerkeingang zwei offensichtliche Möglichkeiten:

1. Wir präsentierem dem Netz pro Zeitschritt die b Bit Daten eines Sensors, wofür das Netz b Eingangsleitungen benötigt. Dann hat das Netz nach s Zeitschritten die Daten der Sensoren gesehen. Diese Anordnung hat folgenden Vorteil: Liegt das Netz schon in Hardware vor, kann man ohne weiteres die Daten weiterer Sensoren hinzufügen. Das Netz muß dann pro Datensatz nur mehr Zeitschritte laufen.

2. Wir präsentierem dem Netz im ersten Zeitschritt das höchstwertige Bit der Daten jedes Sensors. Im zweiten Zeitschritt wird das dem höchstwertigen Bit folgende Bit der Daten jedes Sensors präsentiert usw. Das Netz hat dann soviele Eingangsleitungen, wie es Sensoren gibt und hat nach b Zeitschritten die Daten aller Sensoren gesehen. Diese Anordnung ist vorteilhaft, wenn zwar die Zahl der Sensoren fest ist, aber die Zahl der Bit pro Datenwort, also die Geauigkeit der Daten, variabel ist.

Am Eingang des Netzes ist man also durch die Zahl und Art der Sensoren einigermaßen festgelegt. Am Ausgang des Netzes ist man bei der Wahl der Ausgangsbits freier. Möchte man k Klassen unterscheiden, braucht man am Netzwerkausgang mindestens $\lceil \log_2(k) \rceil$ Bits. Je größer man die Zahl der Ausgangsbits über dieser Mindestgrenze wählt, desto feiner wird die Auflösung in Bezug auf die klassenspezifischen Rückschlußwahrscheinlichkeiten.

Angenommen, die Zahl der Ein- und Ausgangsbits des Netzwerks sind festgelegt, bleiben einem immer noch viele Möglichkeiten, die internen Netzwerkknoten und -leitungen anzuordnen. Bewährt hat sich hier eine baumartige Topologie, d. h. außer den Eingangsknoten hat jeder Knoten zwei oder mehr Vorgänger, von denen er Daten bekommt. Das Netz endet dann in einem Knoten, dessen Ausgangsleitungen den Netzwerkausgang darstellen. Am Zustand dieser Leitungen liest man dann im „Betrieb“ die klassenspezifischen Rückschlußwahrscheinlichkeiten ab.

Zur Optimierung der Netzwerktopologie könnte man sich folgendes Vorgehen vorstellen: Man betrachtet nicht nur die Knotentabellen sondern auch die Verbindungen als zu optimierende Parameter. Die Zahl der Knoten wäre also gegeben, aber der Anfangs- und Endknoten einer Leitung könnte durch die Optimierung verändert werden. Wäre Simulated Annealing der Optimierungsalgorithmus, würden die Änderungen der Knotentabellen mit einer deutlich größeren Wahrscheinlichkeit als eine Änderung der Leitungen vorgenommen. Auf diese Weise würde das Netz die meiste Zeit so optimiert wie bisher, nur ab und zu würde auch eine Verbindung geändert. Ebenso wäre vorstellbar, wärend des Trainings die Art und Zahl der Knoten zu variieren. Die Möglichkeiten automatischen Optimierung der Netzwerktopologie sind in dieser Arbeit nicht weiter untersucht worden.

Teil III

Experimente

10 Boole'sche Funktionen

In der Literatur werden Boole'sche Funktionen immer wieder zum Testen von mustererkennenden Systemen und deren Lernalgorithmen verwendet. Boole'sche Funktionen sind Funktionen $f : \mathbb{B}^n \to \mathbb{B}$, sie realisieren damit ein Zwei-Klassen-Problem. In diesem Kapitel behandeln wir zwei Standardprobleme für neuronale Netze und RAM-basierte neuronale Netze, die Boole'schen Funktionen XOR und AND. Wir zeigen, wie Hyperpermutationsnetzwerke diese Funktionen lernen.

10.1 Das xor-Problem – Ein Knoten ohne Rückkopplungen

Eigentlich ist das XOR-Problem ein Spezialfall des PARITY-Problems, nämlich das PARITY-Problem für zwei Bit. Trotzdem wird es in der Literatur als eigenständiges Problem behandelt, unter anderem auch, weil es das einfachste Problem mit zwei Bit ist, das nicht linear separierbar ist. Am XOR-Problem können wir bei Hyperpermutationsnetzwerken einige grundlegende Dinge zeigen und die optimale Lösung auch analytisch berechnen.

Die Aufgabe ist, mit einem 2-Bit-Knoten die XOR-Funktion

XOR :

$\boldsymbol{x}$	$\boldsymbol{y}$
00	0
01	1
10	1
11	0

zu lernen. In unseren Worten gesprochen geht es darum, die Bitstrings 00 und 11 von 01 und 10 zu unterscheiden. Für das XOR-Problem wird nur ein Zwei-Bit-Knoten benötigt. Die Topologie zeigt Abbildung 10.1.

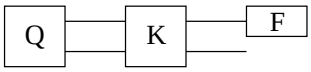


Abbildung 10.1: Q ist die Datenquelle, die die Trainingsdaten liefert. K ist der Knoten und F symbolisiert den Fokus auf dem Ausgangsbit $y_0 = {\phi^{\text{ko}}}_{0,0}(\boldsymbol{y})$ des Knotens.

Natürlich könnten wir in diesem einfachen Fall die gesuchte Knotentabelle durch Ausprobieren aller 24 möglichen Hyperpermutationen eines Zwei-Bit-Knotens finden. Da aber schon etwas größere Knoten überexponentiell mehr Hyperpermutationen haben, ist dieses Vorgehen im allgemeinen unrealistisch.

Wie sieht die Eingangsstatistik des Zwei-Bit-Knotens beim XOR-Problem aus? Dazu schreiben wir die Statistikmatrix am Eingang des Knotens als Tabelle:

Index	0	1
Zustand		
00	1	0
01	0	1
10	0	1
11	1	0

Das XOR-Problem ist separierbar, jeder Zustand wird nur genau einer Klasse zugeordnet. Wenn also der HPN-Knoten die Eingangszustände 00 und 11 auf einen anderen Ausgangszustand abbildet als die Eingangszustände 01 und 10, trennt der Knoten die beiden Klassen wie gewünscht. Da es zu jedem Eingangsbitstring in der Tabelle zwei andere Eingangsbitstrings gibt, die in einer anderen Klasse sind, gibt es insgesamt $4 \cdot 2 = 8$ relevante Unterschiede. Die heiße Information am Knoteneingang ist also $I_\mathrm{h} = 8$.

10.1.1 Die Kostenfunktion

Die Kostenfunktion, die es später zu minimieren gilt, ist sowohl von den Trainingsdaten als auch von der Hyperpermutationsmatrix des Knotens abhängig.

10.1.1.1 Die kalte Information

Als Kostenfunktion kommt sowohl die kalte Information $I_\mathrm{k}(\boldsymbol{H})$ als auch die kalte Nettoinformation $I_\mathrm{nk}(\boldsymbol{H})$ in Frage. Sie unterscheiden sich ja darin, daß die kalte Nettoinformation Redundanz am Ausgang des Knotens bestraft, die kalte Information hingegen nicht. Nun haben wir beim XOR-Problem zwei Klassen, die wir hier auf ein Ausgangsbit abbilden. Daher ist in diesem Fall gar kein Platz für zusätzliche Redundanz am Knotenausgang, I_k und I_nk unterscheiden sich in der Lage ihrer Minima nicht.

Die kalte Information am Knoteneingang erhalten wir aus den Gewichtsvektoren $\boldsymbol{w}_0$, $\boldsymbol{w}_1$ und $\boldsymbol{w} = \boldsymbol{w}_0 + \boldsymbol{w}_1$:

$$\begin{aligned} I_\mathrm{k}^\mathrm{i} &= D_\mathrm{k} - R_\mathrm{k} = \boldsymbol{w} \cdot \boldsymbol{w} - \boldsymbol{w}_0 \cdot \boldsymbol{w}_0 - \boldsymbol{w}_1 \cdot \boldsymbol{w}_1 \\ &= \boldsymbol{w}_0 \cdot \boldsymbol{w}_0 + 2 \cdot \boldsymbol{w}_0 \cdot \boldsymbol{w}_1 + \boldsymbol{w}_1 \cdot \boldsymbol{w}_1 - \boldsymbol{w}_0 \cdot \boldsymbol{w}_0 - \boldsymbol{w}_1 \cdot \boldsymbol{w}_1 \\ &= 2 \cdot \boldsymbol{w}_0 \cdot \boldsymbol{w}_1 \end{aligned}$$

Die Gewichtsvektoren am Knoteneingang erhalten wir aus unseren Trainingsdaten:

$$\begin{aligned} \text{Klasse 0 enthält 00 und 11} \quad &\rightarrow \quad \boldsymbol{w}_0 = (1,0,0,1)^\mathrm{T} \\ \text{Klasse 1 enthält 01 und 10} \quad &\rightarrow \quad \boldsymbol{w}_1 = (0,1,1,0)^\mathrm{T} \\ \text{Klasse 0 und Klasse 1} \quad &\rightarrow \quad \boldsymbol{w} = \boldsymbol{w}_0 + \boldsymbol{w}_1 = (1,1,1,1)^\mathrm{T} \end{aligned}$$

Am Knoteneingang ergibt sich damit die kalte Information I_k^i wie erwartet

$$I_\mathrm{k}^\mathrm{i} = 2 \cdot \boldsymbol{w}_0 \cdot \boldsymbol{w}_1 = 2 \cdot (1,0,0,1)^\mathrm{T} \cdot (0,1,1,0)^\mathrm{T} = 0\,,$$

dort sind die Klassen also perfekt trennbar. Da wir nur zwei Klassen unterscheiden müssen, brauchen wir nur ein Ausgangsbit des Knotens zu betrachten. Wir wählen das Ausgangsbit $y_0 = {\phi^\mathrm{ko}}_{0,0}(\boldsymbol{y})$, indem wir den Fokus des Ausgangsbündels $\boldsymbol{y}$ bilden. Die kalte Information I_k^o am Knotenausgang hängt von der gewählten Knotenmatrix $\boldsymbol{H}$ ab. Wir erhalten I_k^o aus den Gewichtsvektoren $\boldsymbol{w}_0'$, $\boldsymbol{w}_1'$ und $\boldsymbol{w}' = \boldsymbol{w}_0' + \boldsymbol{w}_1'$ am Knotenausgang. Diese ergeben sich aus denen

am Eingang des Knotens durch Transformation mit der Knotenmatrix $\boldsymbol{H}$ und anschließender Projektion mit der Projektionsmatrix $\boldsymbol{\Phi}_0$:

$$\boldsymbol{w}_0' = \boldsymbol{\Phi}_0 \cdot \boldsymbol{H} \cdot \boldsymbol{w}_0 \quad \text{und} \quad \boldsymbol{w}_1' = \boldsymbol{\Phi}_0 \cdot \boldsymbol{H} \cdot \boldsymbol{w}_1$$

Wir erhalten schließlich die Kostenfunktion

$$I_{\mathrm{k}}^{\mathrm{o}}(\boldsymbol{H}) = 2 \cdot \boldsymbol{w}_0' \cdot \boldsymbol{w}_1' = 2 \cdot (\boldsymbol{\Phi}_0 \cdot \boldsymbol{H} \cdot \boldsymbol{w}_0) \cdot (\boldsymbol{\Phi}_0 \cdot \boldsymbol{H} \cdot \boldsymbol{w}_1) \; . \tag{10.1}$$

Finden wir ein $\boldsymbol{H}$ mit $I_{\mathrm{k}}^{\mathrm{o}}(\boldsymbol{H}) = 0$ (am Knotenausgang), dann wissen wir, daß dieses $\boldsymbol{H}$ eine optimale Hyperpermutation ist.

10.1.1.2 Die Knotenmatrix

Die Knotenmatrix $\boldsymbol{H}'$ eines zwei Bit Knotens ist im allgemeinen eine doppelt stochastische $(4,4)$-Matrix, die Zeilen- und Spaltensummen sind jeweils eins und die Matrixelemente sind aus dem Intervall $[0,1]$. In Gleichung (10.1) wird die Knotenmatrix $\boldsymbol{H}'$ mit einer Projektionsmatrix multipliziert, weshalb die Elemente von $\boldsymbol{H}'$ teilweise nur in ihrer Summe wirksam werden. Es ist daher effizienter, die *effektive Knotenmatrix* $\boldsymbol{H} = \boldsymbol{\Phi}_0 \cdot \boldsymbol{H}'$ zu optimieren, die wegen der Projektion auch weniger Parameter hat. Die effektive Knotenmatrix $\boldsymbol{H}$ ist:

$$\begin{aligned}\boldsymbol{H} = \boldsymbol{\Phi}_0 \cdot \boldsymbol{H}' &= \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} h'_{00} & h'_{01} & h'_{02} & h'_{03} \\ h'_{10} & h'_{11} & h'_{12} & h'_{13} \\ h'_{20} & h'_{21} & h'_{22} & h'_{23} \\ h'_{30} & h'_{31} & h'_{32} & h'_{33} \end{pmatrix} \\ &= \begin{pmatrix} h'_{00}+h'_{20} & h'_{01}+h'_{21} & h'_{02}+h'_{22} & h'_{03}+h'_{23} \\ h'_{10}+h'_{30} & h'_{11}+h'_{31} & h'_{12}+h'_{32} & h'_{13}+h'_{33} \end{pmatrix} = \begin{pmatrix} h_{00} & h_{01} & h_{02} & h_{03} \\ h_{10} & h_{11} & h_{12} & h_{13} \end{pmatrix}\end{aligned}$$

Wir werden also die Matrix $\boldsymbol{H}$ optimieren. Durch die Projektion mit $\boldsymbol{\Phi}_0$ gilt nun für $\boldsymbol{H}$, daß die Zeilensumme gleich zwei und die Spaltensummen gleich 1 ist. Durch diese Randbedingungen sind von den acht Matrixelementen in $\boldsymbol{H}$ nur drei unabhängig. Wir wählen als unabhängige Matrixelemente h_{00}, h_{01} und h_{02} und erhalten

$$\boldsymbol{H} = \begin{pmatrix} h_{00} & h_{01} & h_{02} & 2 - h_{00} - h_{01} - h_{02} \\ 1 - h_{00} & 1 - h_{01} & 1 - h_{02} & -1 + h_{00} + h_{01} + h_{02} \end{pmatrix} \; . \tag{10.2}$$

10.1.2 Analytische Lösung

Gesucht sind die Hyperpermutationen $\boldsymbol{H}$ mit $I_{\mathrm{k}}^{\mathrm{o}}(\boldsymbol{H}) = 0$. Die Gewichtsvektoren am Knotenausgang sind:

$$\begin{aligned} \boldsymbol{w}_0' &= \boldsymbol{H} \cdot \boldsymbol{w}_0 = \boldsymbol{H} \cdot \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 2 - h_{01} - h_{02} \\ h_{01} + h_{02} \end{pmatrix} \\ \boldsymbol{w}_1' &= \boldsymbol{H} \cdot \boldsymbol{w}_1 = \boldsymbol{H} \cdot \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} h_{01} + h_{02} \\ 2 - h_{01} - h_{02} \end{pmatrix} \\ \boldsymbol{w}' &= \boldsymbol{w}_0' + \boldsymbol{w}_1' = \begin{pmatrix} 2 \\ 2 \end{pmatrix} \end{aligned} \tag{10.3}$$

Wie wir an $\boldsymbol{w}' = (2,2)^{\mathrm{T}}$ ablesen können, wird das Ausgangsbit $\boldsymbol{y}_0$ des Knotens auf den Trainingdaten zweimal 0 und zweimal 1 sein.

Die kalte Information am Knotenausgang ist mit (10.3) und $\alpha = h_{01} + h_{02}$:

$$I_{\mathrm{k}}^{\mathrm{o}} = 2 \cdot \boldsymbol{w}_0' \cdot \boldsymbol{w}_1' = 2 \cdot \binom{2-\alpha}{\alpha} \cdot \binom{\alpha}{2-\alpha} = 4 \cdot (2-\alpha) \cdot \alpha$$

Hier handelt es sich um einen der seltenen Fälle, in denen wir die Kostenfunktion visualisieren können, siehe Abbildung 10.2.

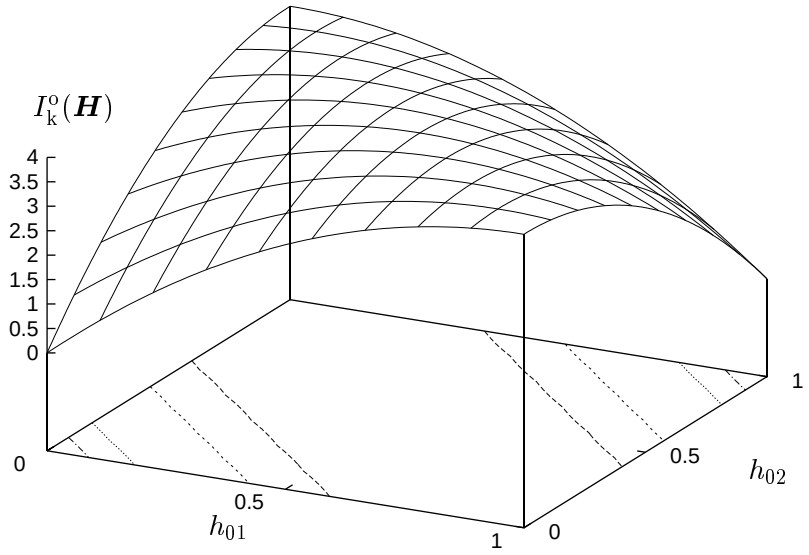


Abbildung 10.2: Kostenfunktion des XOR-Problems. Gesucht sind die Minima der Funktion in dem Gebiet $[0,1]^2$.

Wie wir wissen, gilt für die optimale Knotenmatrix $I_{\mathrm{k}}^{\mathrm{o}}(\boldsymbol{H}) = 0$:

$$I_{\mathrm{k}}^{\mathrm{o}} = 4 \cdot (2-\alpha) \cdot \alpha \stackrel{!}{=} 0 \qquad \Rightarrow \qquad \alpha = 0 \quad \text{oder} \quad \alpha = 2$$

Mit $\alpha = h_{01} + h_{02}$ und der Nebenbedingung $h_{ij} \in [0,1]$ erhalten wir:

$$\begin{array}{lllll} \alpha = 0 & \Rightarrow & h_{01} = h_{02} = 0 & \Rightarrow & h_{00} = 1 \quad \text{oder} \\ \alpha = 2 & \Rightarrow & h_{01} = h_{02} = 1 & \Rightarrow & h_{00} = 0 \end{array} \tag{10.4}$$

Setzen wir (10.4) in (10.2) ein, ergeben sich die beiden effektiven optimalen Hyperpermutati-

onsmatrizen:

$$\boldsymbol{H} = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix} \qquad \text{bzw.} \qquad \mathrm{h} = \begin{array}{|c|c|} \hline \boldsymbol{x} & \boldsymbol{y} \\ \hline 00 & 0 \\ 01 & 1 \\ 10 & 1 \\ 11 & 0 \\ \hline \end{array}$$

$$\bar{\boldsymbol{H}} = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix} \qquad \text{bzw.} \qquad \bar{\mathrm{h}} = \begin{array}{|c|c|} \hline \boldsymbol{x} & \boldsymbol{y} \\ \hline 00 & 1 \\ 01 & 0 \\ 10 & 0 \\ 11 & 1 \\ \hline \end{array}$$

Dabei wird die Klassenkodierung nicht festgelegt, weswegen wir hier die beiden äquivalenten Hyperpermutationen h und $\bar{\mathrm{h}}$ erhalten. Transformieren wir die klassenspezifischen Gewichtsvektoren am Knoteneingang mit der Matrix $\boldsymbol{H}$, erhalten wir:

$$\boldsymbol{H} \cdot \boldsymbol{w}_0 = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 2 \\ 0 \end{pmatrix}$$

$$\boldsymbol{H} \cdot \boldsymbol{w}_1 = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 2 \end{pmatrix}$$

Durch die Minimierung der Kostenfunktion wird erreicht, daß die Muster aus den verschiedenen Klassen unterschieden werden.

Die Hyperpermutationen, die wir normalerweise *nicht* suchen, sind die im Maximium der Kostenfunktion. Welche sind das? Das Maximium der Kostenfunktion erhalten wir mit:

$$\frac{\mathrm{d}I_{\mathrm{k}}^{\mathrm{o}}}{\mathrm{d}\alpha} = 4 \cdot (-\alpha + (2 - \alpha)) = 8 \cdot (1 - \alpha) \stackrel{!}{=} 0 \qquad \Rightarrow \qquad \alpha = 1 \quad \Rightarrow \quad I_{\mathrm{k}}^{\max} = 4$$

Mit der Nebenbedingung $h_{ij} \in [0, 1]$ ergibt sich

$$\alpha = 1 \quad \Rightarrow \quad h_{01} + h_{02} = 1 \qquad \Rightarrow \quad h_{00} \in [0, 1] \; .$$

Transformieren wir die klassenspezifischen Gewichtsvektoren am Knoteneingang mit den „schlechten“ Hyperpermutationen $\boldsymbol{H}^{I_{\mathrm{k}}^{\max}}$, erhalten wir:

$$\boldsymbol{H}^{I_{\mathrm{k}}^{\max}} \cdot \boldsymbol{w}_0 = \begin{pmatrix} h_{00} & h_{01} & 1 - h_{01} & 1 - h_{00} \\ 1 - h_{00} & 1 - h_{01} & h_{01} & h_{00} \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

$$\boldsymbol{H}^{I_{\mathrm{k}}^{\max}} \cdot \boldsymbol{w}_1 = \begin{pmatrix} h_{00} & h_{01} & 1 - h_{01} & 1 - h_{00} \\ 1 - h_{00} & 1 - h_{01} & h_{01} & h_{00} \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Die Hyperpermutationen mit $I_{\mathrm{k}}^{\mathrm{o}}(\boldsymbol{H}) = I_{\mathrm{k}}^{\max}$ ordnen beide Elemente *einer* Klasse am Ausgang des Knotens zwei *verschiedenen* Klassen zu. Anders herum ausgedrückt, ordnen sie je zwei Elemente aus verschiedenen Klassen am Ausgangs des Knotens der gleichen Klasse zu. Sie machen also den „maximal möglichen Fehler". Die Größe $I_{\mathrm{k}}^{\max} = 4$ zeigt uns, daß von den relevanten Unterschieden vier unterschlagen werden. Da es am Knoteneingang überhaupt nur acht relevante Unterschiede gibt, werden am Knotenausgang 50% falsch zugeordnet. Diese 50% sind gerade die zwei falschen von vier Bitstrings.

10.1.3 Numerische Lösung

Sobald ein Problem praxisnäher als das XOR-Problem wird, ist es im allgemeinen analytisch nicht mehr lösbar. Wir werden nun das XOR-Problem numerisch lösen.

10.1.3.1 Liniensuche

Wir lassen nun den in Abschnitt 8.4.2 vorgestellten Liniensuche-Algorithmus die optimale Knotentabelle finden. Als Kostenfunktion verwenden wir dabei zum Vergleich sowohl die kalte Information I_{k} als auch die kalte Nettoinformation I_{nk}. Um die Eingenschaften des Algorithmus beim Training zu finden, wurde das XOR-Problem in 1000 unabhängigen Läufen trainiert. Dabei dauerte ein Trainingslauf (inclusive der Protokollgenerierung) im Mittel 5 ms (Pentium-Rechner), und jeder Lauf fand die optimale Tabelle. Abbildung 10.3 zeigt, nach wievielen Funktionsaufrufen die Lösung gefunden wurde, wobei im Mittel eine Linienminimierung etwas weniger als zwei Funktionsaufrufe benötigt.

Ein „Funktionsaufruf" bezeichnet dabei eine einmalige Auswertung der Kostenfunktion. Bei rückkopplungsfreien Netzen müssen wir dazu die Trainingsdaten *nicht* wiederholt durchlaufen, es reicht, einmal am Anfang des Trainings die Gewichtsvektoren der Trainingsdaten zu bestimmen. Bei Netzen mit Rückkopplungen müssen wir für die Auswertung der Kostenfunktion zumindest einen Teil der Trainingsdaten „durch das Netz" schicken. Dort dauert ein Funktionsaufruf entsprechend länger.

Der Algorithmus optimiert jeweils eine Spalte der (2,4)-Knotenmatrix, er kann also frühestens nach vier Linienoptimierungen die Lösung gefunden haben. Da die zu optimierende Spalte zufällig ausgewählt wird, muß der Algorithmus im Mittel $4/4 + 4/3 + 4/2 + 4/1 = 8,\overline{3}$ mal würfeln, bevor er jede Spalte der Matrix mindestens einmal optimiert hat. In Abbildung 10.3 sehen wir, daß ungefähr nach 13 Funktionsaufrufen die Hälfte der Trainingsläufe die Lösung gefunden hat, was damit übereinstimmt, daß eine Linienminimierung im Mittel etwas weniger als zwei Funktionsaufrufe benötigt.

Wie ändert sich die Knotenmatrix bei einem ausgewählten Trainingslauf? Sie nimmt wärend einer ausgewählten Optimierung folgende sechs Werte an, wobei die vorangestellte Zahl die Nummer des Funktionsaufrufs angibt:

$$0: \quad \boldsymbol{H} = \begin{pmatrix} 0.5 & 0.5 & 0.5 & 0.5 \\ 0.5 & 0.5 & 0.5 & 0.5 \end{pmatrix} \qquad 4: \quad \boldsymbol{H} = \begin{pmatrix} 0.5 & 0.5 & 1 & 0.5 \\ 0.5 & 0.5 & 0 & 0.5 \end{pmatrix}$$

$$6: \quad \boldsymbol{H} = \begin{pmatrix} 0.5 & 0.5 & 1 & 0 \\ 0.5 & 0.5 & 0 & 1 \end{pmatrix} \qquad 8: \quad \boldsymbol{H} = \begin{pmatrix} 0.5 & 0.5 & 1 & 0 \\ 0.5 & 0.5 & 0 & 1 \end{pmatrix}$$

$$10: \quad \boldsymbol{H} = \begin{pmatrix} 0.5 & 1 & 1 & 0 \\ 0.5 & 0 & 0 & 1 \end{pmatrix} \qquad 12: \quad \boldsymbol{H} = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix}$$

Abbildung 10.4 zeigt einen exemplarischen Trainingslauf mit der Kostenfunktion I_{k} und den anderen protokollierten statistischen Größen. Wir sehen, daß zu jeder Änderung der Knotenmatrix eine Änderung der Kostenfunktion gehört. Im einzelnen passiert folgendes:

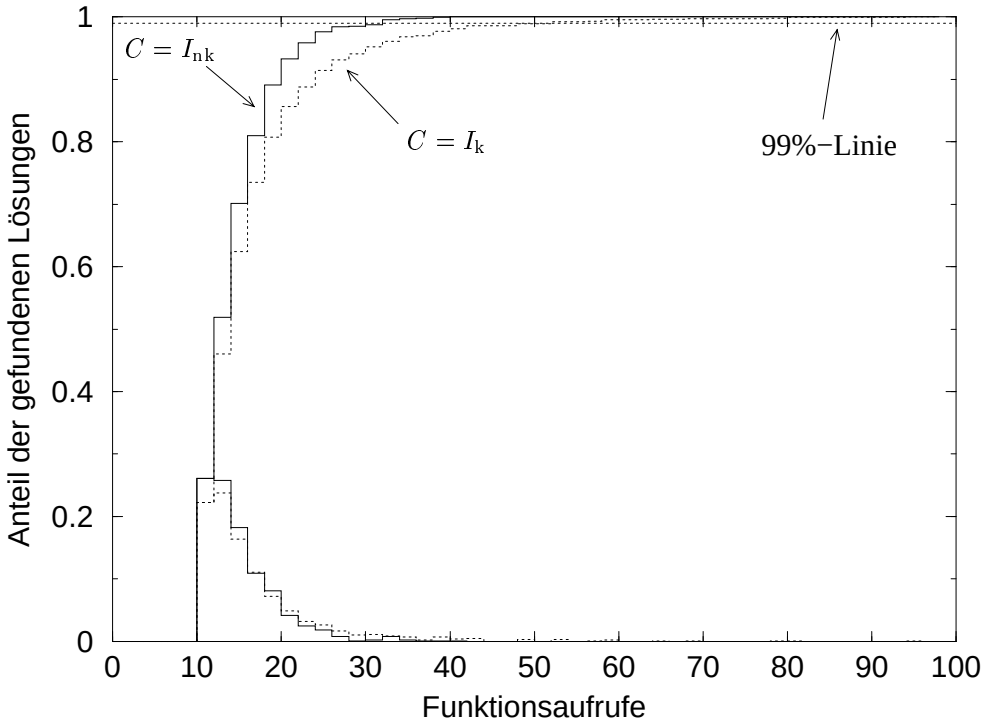


Abbildung 10.3: Relativer Anteil der gefundenen Lösungen zu der Gesamtzahl der Lösungen beim XOR-Problem nach n Funktionsaufrufen. Das Histogramm (untere Kurve) und das kumulative Histogramm (obere Kurve) sind eingezeichnet. Die gestrichelte Linie gehört zur kalten Information, die durchgezogene Linie zur kalten Nettoinformation. Ist die kalte Nettoinformation die Kostenfunktion, terminiert der Algorithmus im Mittel schneller, als wenn die kalte Information die Kostenfunktion ist.

0: Dies ist der Ausgangswert der Matrix vor der Optimierung. Alle Matrixelemente sind gleich, d. h. die Wahrscheinlichkeit für eine 0 bzw. 1 am Knotenausgang ist für jedes Eingangssymbol 1/2.

4: Die dritte Spalte ist durch die Linienminimierung optimiert worden und hat den Wert $\binom{1}{0}$. Das bedeutet, daß das Eingangssymbol 10 nun auf dem Ausgangsbit y_0 auf den Wert 0 abgebildet wird.

6: Die vierte Spalte wurde optimiert.

8: Eine der beiden schon optimierten Spalten wurde erneut zum Optimieren ausgewählt, was aber an dem schon gefundenen Wert nichts geändert hat.

10: Die zweite Spalte wurde optimiert.

12: Die erste Spalte wurde optimiert.

Die gefundene Matrix ist die gleiche, die wir auch analytisch in Abschnitt 10.1.2 berechnet haben.

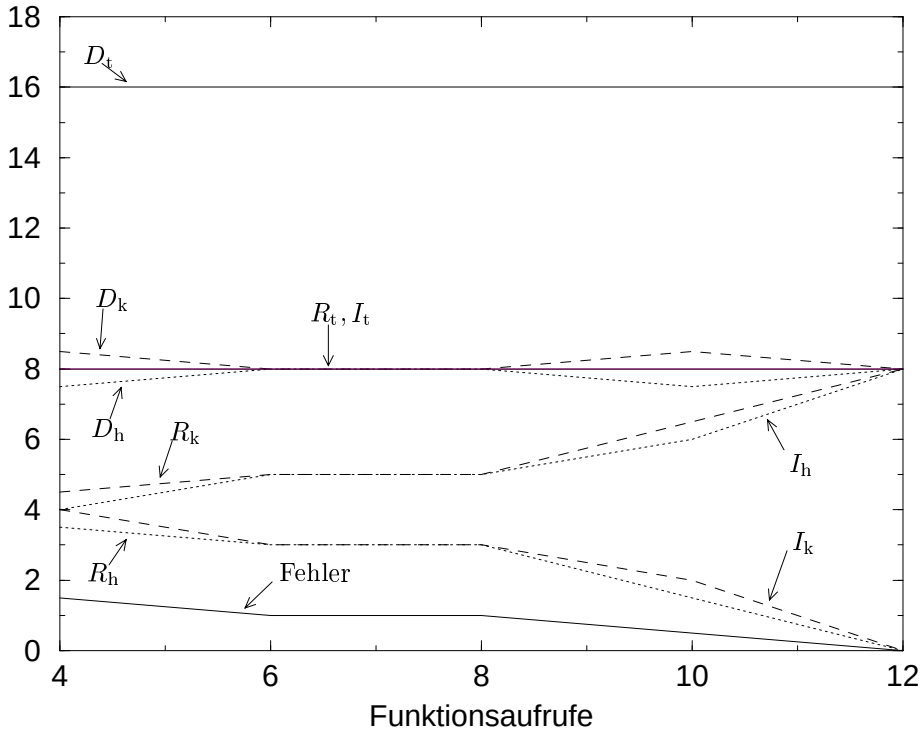


Abbildung 10.4: Trainingslauf des XOR-Problems. Die Kostenfunktion ist $C = I_k$. Die heißen und kalten Anteile einer statistischen Größe addieren sich immer zum Totalen dieser Größe ($D_t = D_h + D_k$ etc.), außerdem ist jeweils $D = I + R$. Die Kostenfunktion ist am Anfang $I_k = 4 = I_t/2$, d. h. die Hälfte der informativen Unterschiede geht verloren. Am Ende der Optimierung ist $I_k = 0$, d. h. kein informativer Unterschied geht verloren. Die in den Daten vorhandene Information ist $I_t = 8$, die am Knotenausgang verfügbare Information I_h steigt von $I_h = I_t/2$ bis auf $I_h = I_t$ am Ende der Optimierung, womit das XOR-Problem gelöst ist und der Trainingslauf beendet wird. Entsprechend sinkt die Zahl der Fehler am Ende der Optimierung auf null.

10.1.4 Numerische Lösung bei zwei Bit am Knotenausgang

Wir wollen nun den Unterschied der beiden Kostenfunktionen I_k und I_nk an einem Knoten untersuchen, bei dem der Fokus nicht auf ein Ausgangsbit sondern auf beide Ausgangsbits gerichtet ist, siehe Abbildung 10.1. Für das XOR-Problem reicht ein Ausgangsbit aus. Also sollte die Kostenfunktion Lösungen „produzieren“, die mit zwei der vier möglichen Ausgangszustände eines 2-Bit-Knotens auskommen. Auf diese Weise würde vom Ausgangsraum des Knotens nur derjenige Teil benutzt, der zur Lösung der Aufgabe wirklich notwendig ist.

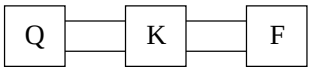


Abbildung 10.5: Der Fokus F erstreckt sich nun über beide Ausgangsbits des Knotens K.

Hier kommen die beiden Kostenfunktionen I_k und I_nk ins Spiel: I_k bestraft die Redundanz am Knotenausgang nicht, I_nk bestraft sie. Formal fassen wir nun I_k und I_nk zu einer Kostenfunktionen mit einem Parameter zusammen:

$$I(\lambda) = D_\mathrm{k} - \lambda R_\mathrm{k}$$

Damit ist $I_\mathrm{k} = I(1)$ und $I_\mathrm{nk} = I(D_\mathrm{t}/R_\mathrm{t})$. Um die Kostenfunktionen zu vergleichen, führen wir wieder 1000 Trainingsläufe durch und untersuchen die gefundenen optimalen Knotentabellen. Abbildung 10.6 zeigt, welche Werte der Elemente bei den optimalen Knotentabellen auftreten.

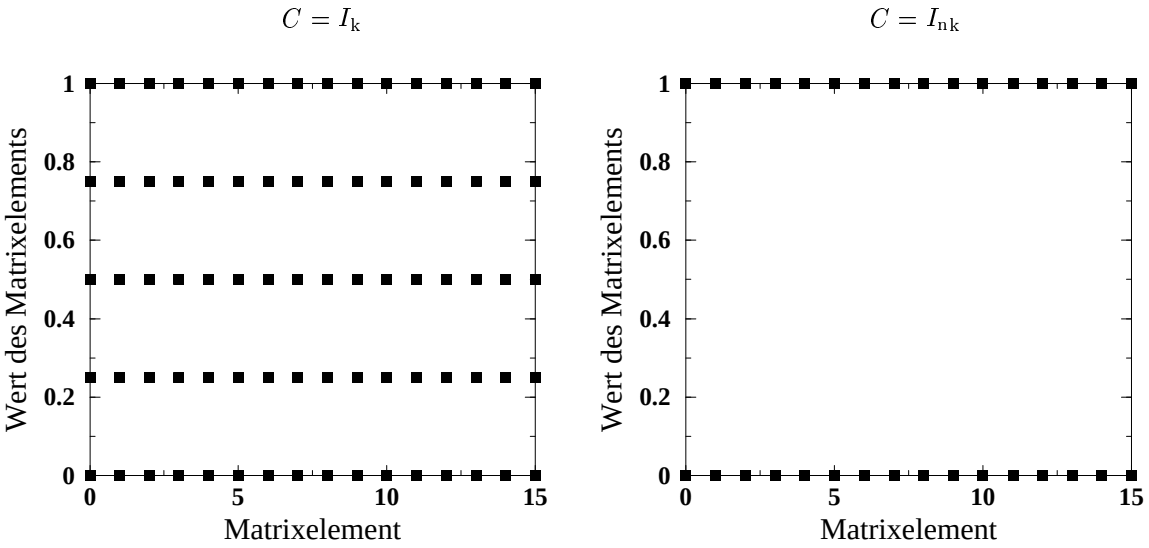


Abbildung 10.6: Vergleich von I_k und I_nk beim XOR-Problem. Im linken Bild ist die Kostenfunktion $C = I_\mathrm{k}$, im rechten ist $C = I_\mathrm{nk}$ bzw. $C = I(\lambda = 1.001)$. Eingezeichnet sind die Werte der Elemente der optimalen Knotenmatrix, die bei 1000 Trainingsläufen vorgekommen sind. Ist $C = I_\mathrm{k}$, kommen optimale Knotenmatrizen vor, die Elemente mit Werten innerhalb des Intervalls $[0, 1]$ enthalten. Dagegen haben bei den Kostenfunktionen $C = I_\mathrm{nk}$ bzw. $C = I(\lambda = 1.001)$ die Elemente der optimale Knotenmatrix nur die Werte 0 oder 1.

Wie kommt dieses unterschiedliche Verhalten zu Stande? Ist die Kostenfunktion $C = I_\mathrm{k}$ ist eine Knotenmatrix optimal, wenn alle Eingangsmuster, die zu verschiedenen Klassen gehören,

am Knotenausgang noch unterschieden werden können. Daraus folgt aber nicht, daß eine solche optimale Knotenmatrix auch deterministisch sein müßte. Die Matrix

$$\begin{pmatrix} 1 & 0 & 0 & 0.25 \\ 0 & 0.5 & 0.5 & 0 \\ 0 & 0 & 0 & 0.75 \\ 0 & 0.5 & 1 & 0 \end{pmatrix}$$

bildet z. B. die Eingangsbitstrings 00 und 11 auf die Ausgangsbitstrings 00 und 10 ab. Dagegen werden die Eingangsbitstrings 01 und 10 auf die Ausgangsbitstrings 01 und 11 abgebildet, die Matrix kann also die beiden Klassen am Knotenausgang auseinanderhalten. Eine solche Matrix hat aber zwei Nachteile:

1. Eine stochastische Matrix läßt sich nicht als Tabellennachschlag implementieren, dafür müßten die Matrixelemente nur aus Nullen und Einsen bestehen.

2. Am Knotenausgang werden alle vier Ausgangssymbole verwendet, obwohl wir zur Lösung des XOR-Problems nur zwei verschiedene Ausgangssymbole benötigen. Mit dem Platz im Ausgangsraum wird „verschwenderisch umgegangen".

Im Fall $C = I_{\mathrm{nk}}$ bzw. $C = I_{\mathrm{k}}(\lambda = 1.001)$ sehen die optimalen Knotenmatrizen z. B. so aus:

$$\begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} .$$

Diese Matrix ist nicht nur deterministisch. Von den vier möglichen Ausgangszuständen werden hier nur zwei benutzt, nämlich die beiden Zustände 00 und 10. Haben wir also einen großen Zustandsraum am Ausgang des Knotens, können wir mit einer Kostenfunktion $C = I(\lambda > 1)$ die Ausbreitung der Zustände im Zustandsraum eindämmen.

10.2 Das xor-Problem – Ein Knoten mit Rückkopplungen

Ein wichtiger Punkt bei HPNs ist, daß wir Netzwerke mit und ohne Rückkopplungen im Prinzip mit den gleichen Kostenfunktionen trainieren können, da sich die statistischen Größen Diversität, Redundanz und Information gleichermaßen dafür eignen.

Wir trainieren nun wie im vorigen Abschnitt das XOR-Problem, aber wir speisen die beiden Bits des Eingangsbitstrings *nacheinander* in den Knoten ein und registrieren, welchen Zustand die rückgekoppelten Leitungen nach einem Eingangsmuster haben. Wir können die Bitstrings unterscheiden, wenn die Rückkopplungen nach Eingangsmustern aus verschiedenen Klassen verschiedenen Zustände haben.

10.2.1 2-Bit-Knoten

Der kleinste mögliche Knoten mit einem Eingangsbit und einer Rückkopplung ist der 2-Bit-Knoten. Die Topologie dieses „Mini-Netzwerks" zeigt Abbildung 10.7.

Um die Leistungsfähigkeit der rückgekoppelten Anordnung zu bewerten, wurden wieder 1000 Trainingsläufe durchgeführt, wobei jeder Trainingslauf ca. 10 ms dauerte. Wie zu erwarten war, werden die Lösungen des XOR-Problems auch mit einem rückgekoppelten Knoten gefunden. Abbildung 10.8 zeigt den relativen Anteil der gefundenen Lösungen und Abbildung 10.9 den Verlauf der statistischen Größen bei einem exemplarischen Trainingslauf.

Abbildung 10.7: XOR-Problem von zwei Bit – Knoten mit Rückkopplungen. Q ist die Datenquelle, die die Trainingsdaten liefert. K ist der Knoten und F symbolisiert den Fokus auf dem Ausgang $y_1 = \phi^{\mathrm{ko}}{}_{1,1}(\boldsymbol{y})$ des Knotens. Formal duplizieren wir die rückgekoppelte Leitung, um ihren Wert auslesen zu können, so wie links gezeigt. In einer Softwaresimulation oder einer Hardwareimplementierung können wir aber auch mit dem Fokus direkt auf die rückgekoppelte Leitung „blicken“, so wie rechts gezeigt.

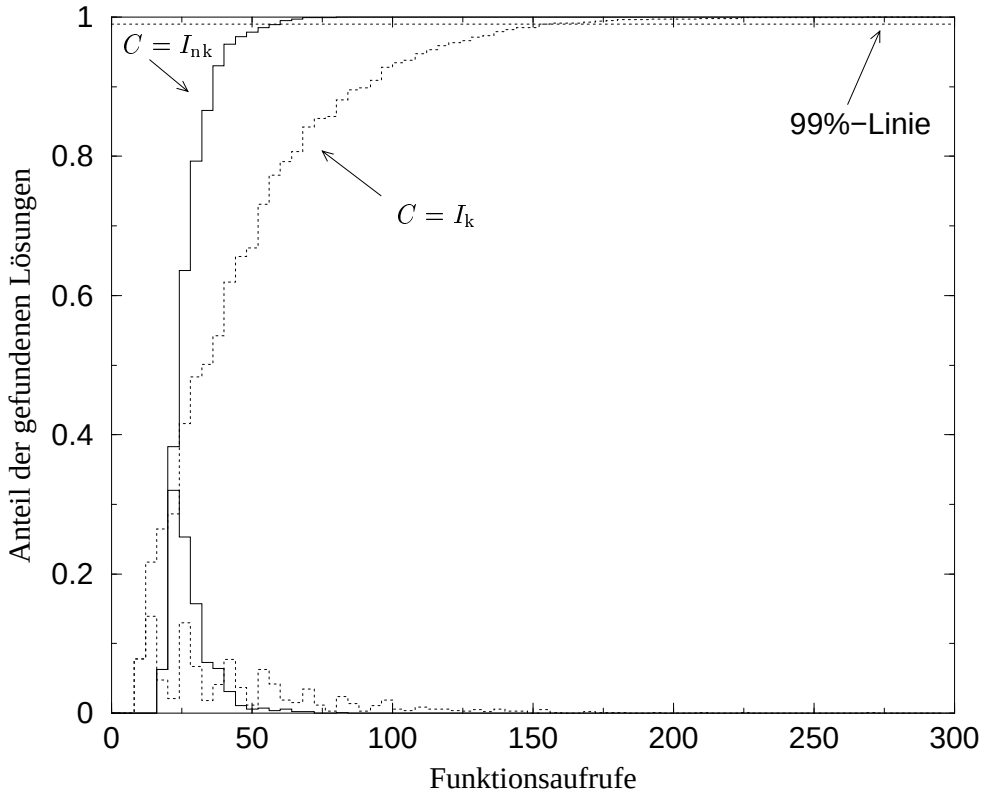


Abbildung 10.8: Relativer Anteil der gefundenen Lösungen zu der Gesamtzahl der Lösungen beim XOR-Problem mit einem Rückkopplungsbit nach n Funktionsaufrufen. Im Mittel dauert es bei einem Knoten mit Rückkopplungen etwas länger als bei einem Knoten ohne Rückkopplungen, die Lösung zu finden.

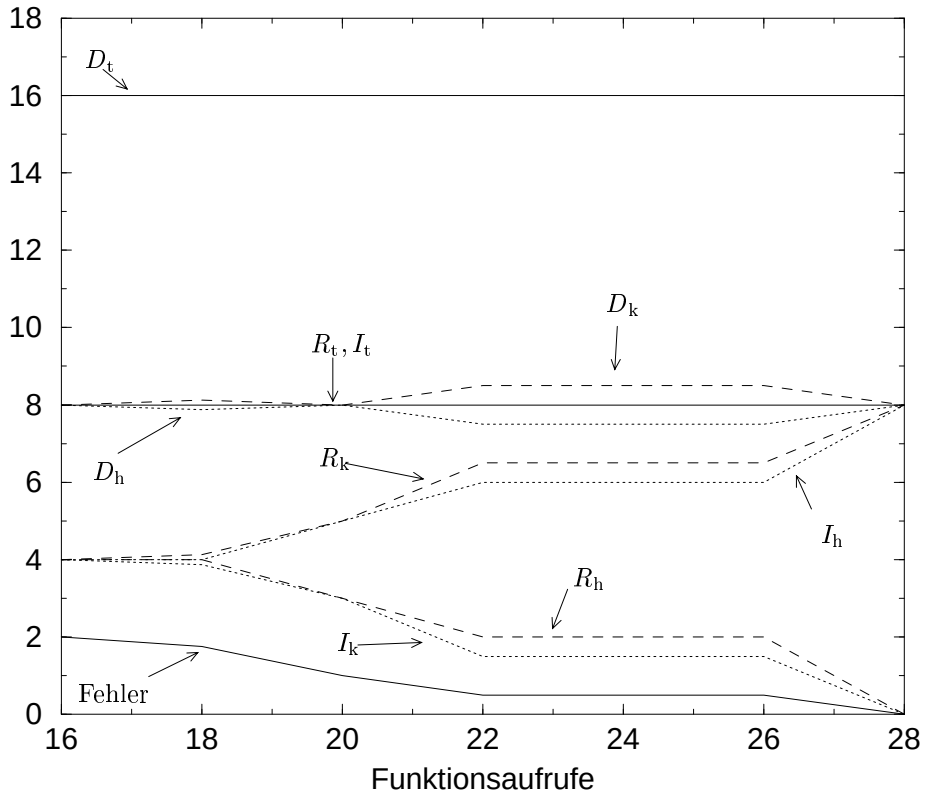


Abbildung 10.9: Trainingslauf des XOR-Problems von zwei Bit mit einem Rückkopplungs-Bit. Der Verlauf der statistischen Größen ist dem rückkopplungsfreien Fall recht ähnlich.

10.2.2 Die parity-Funktion

Ein wichtiger Aspekt dieses Beispiels ist, daß ein rückgekoppelter 1-Bit-Knoten, der das XOR-Problem gelernt hat, auch dessen Verallgemeinerung, das PARITY-Problem, lösen kann. Die Eingangsbitmuster dürfen also nicht nur die Länge zwei sondern eine beliebige Länge haben. Der Knoten unterscheidet die Eingangsbitmuster dann danach, ob sie eine gerade oder ungerade Zahl von Einsen enthalten.

Warum kann auch das PARITY-Problem gelöst werden? Um die Parität eines Bitstrings zu berechnen, braucht man nur ein einziges Bit als Speicher, wenn man den Bitstring seriell abarbeitet. Jedesmal, wenn man auf eine Eins trifft, muß man das Speicherbit invertieren. Hatte das Speicherbit am Anfang den Wert 0, steht am Ende eine 1 im Speicherbit für eine ungerade Zahl von Einsen im Bitstring und eine 0 für eine gerade Zahl von Einsen. Die gerade beschriebene Situation wird aber gerade von einem Knoten mit einem rückgekoppelten Bit realisiert. Dies ist ein Beispiel, bei dem mit einer sehr kleinen rückgekoppelten Struktur Eingangsbitmuster beliebiger Länge verarbeitet werden können.

10.3 Das and-Problem – Ein Knoten ohne Rückkopplungen

Das AND ist im Gegensatz zum XOR eigentlich ein einfaches Problem, da die beiden Klassen im Merkmalsraum linear separierbar sind. Allerdings gibt es von der Klasse 1 nur *einen* Repräsentanten, den Bitstring mit allen Bits gleich Eins. Damit hat die Klasse 1 bei n Bit von allen Bitstrings nur einen Anteil von $1/2^n$, was es statistischen Verfahren erschweren kann, den einen Repräsentanten der Klasse 1 im Merkmalsraum zu isolieren.

10.3.1 Das and-Problem auf drei Bit

Wir trainieren eine AND-Funktion von drei Bit. Die Trainingsdaten bestehen aus den acht möglichen Eingangskombinationen von drei Bit und dem zugehörigen Klassenindex, der hier dem Wahrheitswert entspricht. Die AND-Funktion hat folgende Wahrheits- bzw. Statistiktabelle am Knoteneingang:

and :

x	y
000	0
001	0
010	0
011	0
100	0
101	0
110	0
111	1

Statistik :

Index	0	1
Zustand		
000	1	0
001	1	0
010	1	0
011	1	0
100	1	0
101	1	0
110	1	0
111	0	1

Für das AND-Problem von drei Bit wird nur ein Knoten benötigt, bei dem wir wieder den Fokus auf das Ausgangsbit y_0 setzen. Bildet der Knoten die Eingangszustände 000, 001, ..., 110 auf einen anderen Zustand von y_0 ab als den Eingangszustand 111, hat er das Problem gelöst. Da der Knoten auf den einen Ausgangszustand sieben, auf den anderen Ausgangszustand aber nur einen Bitstring abbildet muß, kann dieses ein „reiner“ 3-Bit-Knoten nicht bewerkstelligen. Denn eine solche Abbildung ist, ebenso wie die AND-Funktion, nicht bijektiv.

Wir führen formal ein Blindbit ein, so daß der Merkmalsraum 16 Eingangsmuster hat, von denen aber nur die acht mit $x_3 = 0$ genutzt werden. Das Blindbit hat immer den Zustand 0, wes-

halb in einer Implementierung nur die „Hälfte“ der Knotentabelle wirklich gespeichert werden muß. Nun ist im Ausgangsraum des Knotens „genug Platz“, da es jeweils acht Ausgangsmuster mit $y_0 = 0$ bzw. $y_0 = 1$ gibt. Die Topologie zeigt Abbildung 10.10.

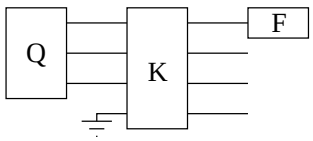


Abbildung 10.10: Q ist die Datenquelle, K der Knoten und F symbolisiert den Fokus auf dem Ausgangsbit y_0 des Knotens. Der Knoten hat ein Blindbit.

10.3.2 Die Kostenfunktion

Wie beim XOR-Problem stellen wir die Kostenfunktion auf.

10.3.2.1 Die kalte Information

Unsere Kostenfunktion ist wiederum die kalte Information $I_\mathrm{k}(\boldsymbol{H})$, die wir aus den Gewichtsvektoren $\boldsymbol{w}_0$, $\boldsymbol{w}_1$ und $\boldsymbol{w} = \boldsymbol{w}_0 + \boldsymbol{w}_1$ erhalten:

$$I_\mathrm{k} = 2 \cdot \boldsymbol{w}_0 \cdot \boldsymbol{w}_1$$

Die Gewichtsvektoren am Knoteneingang sind:

$$\begin{aligned}
\text{Klasse 0 enthält } 000, 001, \ldots, 110 \quad &\rightarrow \quad \boldsymbol{w}_0 = (1,1,1,1,1,1,1,0)^\mathrm{T} \\
\text{Klasse 1 enthält nur } 111 \quad &\rightarrow \quad \boldsymbol{w}_1 = (0,0,0,0,0,0,0,1)^\mathrm{T} \\
\text{Klasse 0 und Klasse 1} \quad &\rightarrow \quad \boldsymbol{w} = \boldsymbol{w}_0 + \boldsymbol{w}_1 = (1,1,1,1,1,1,1,1)^\mathrm{T}
\end{aligned}$$

Am Knoteneingang ergibt sich damit wie erwartet die kalte Information

$$I_\mathrm{k}^\mathrm{i} = 2 \cdot \boldsymbol{w}_0 \cdot \boldsymbol{w}_1 = 2 \cdot (1,1,1,1,1,1,1,0)^\mathrm{T} \cdot (0,0,0,0,0,0,0,1)^\mathrm{T} = 0 \,,$$

und die kalte Information I_k^o am Knotenausgang ist wieder

$$I_\mathrm{k}^\mathrm{o}(\boldsymbol{H}) = 2 \cdot \boldsymbol{w}_0' \cdot \boldsymbol{w}_1' = 2 \cdot (\boldsymbol{\Phi}_0 \cdot \boldsymbol{H} \cdot \boldsymbol{w}_0) \cdot (\boldsymbol{\Phi}_0 \cdot \boldsymbol{H} \cdot \boldsymbol{w}_1) \,.$$

Gesucht sind die $\boldsymbol{H}$ mit $I_\mathrm{k}^\mathrm{o}(\boldsymbol{H}) = 0$.

10.3.2.2 Die Knotenmatrix

Die Knotenmatrix $\boldsymbol{H}'$ eines 4-Bit-Knotens ist eine doppelt stochastische $(16,16)$-Matrix. Da wir nur zwei Klassen unterscheiden müssen, wählen wir ein Ausgangsbit, und zwar $y_0 = {\phi^\mathrm{ko}}_{0,0}(\boldsymbol{y})$ als Fokus des Knotens. Unsere effektive Knotenmatrix $\boldsymbol{H}$ ist damit nur noch eine $(2,16)$-Matrix. Das vierte Bit des Knotens, das Blindbit, berücksichtigen wir dadurch, daß wir die Nebenbedingung „Zeilensumme gleich Eins“ fallen lassen, wodurch wir das Blindbit „einsparen“ können. Damit wird die effektive Knotenmatrix $\boldsymbol{H}$ unter Berücksichtigung der Nebenbedingung „Spaltensumme gleich Eins“ eine $(2,8)$-Matrix mit acht unabhängigen Elementen:

$$\begin{aligned}
\boldsymbol{H} &= \begin{pmatrix} h_{00} & h_{01} & h_{02} & h_{03} & h_{04} & h_{05} & h_{06} & h_{07} \\ h_{10} & h_{11} & h_{12} & h_{13} & h_{14} & h_{15} & h_{16} & h_{17} \end{pmatrix} \\
&= \begin{pmatrix} h_{00} & h_{01} & h_{02} & h_{03} & h_{04} & h_{05} & h_{06} & h_{07} \\ 1-h_{00} & 1-h_{01} & 1-h_{02} & 1-h_{03} & 1-h_{04} & 1-h_{05} & 1-h_{06} & 1-h_{07} \end{pmatrix}
\end{aligned}$$

10.3.3 Analytische Lösung

Die Gewichtsvektoren am Knotenausgang sind mit $\alpha := h_{00} + h_{01} + h_{02} + h_{03} + h_{04} + h_{05} + h_{06}$:

$$\begin{aligned} \boldsymbol{w}'_0 &= \boldsymbol{H} \cdot \boldsymbol{w}_0 = \boldsymbol{H} \cdot (1,1,1,1,1,1,1,0)^{\mathrm{T}} = \begin{pmatrix} \alpha \\ 7-\alpha \end{pmatrix} \\ \boldsymbol{w}'_1 &= \boldsymbol{H} \cdot \boldsymbol{w}_1 = \boldsymbol{H} \cdot (0,0,0,0,0,0,0,1)^{\mathrm{T}} = \begin{pmatrix} h_{07} \\ 1-h_{07} \end{pmatrix} \\ \boldsymbol{w}' &= \boldsymbol{w}'_0 + \boldsymbol{w}'_1 = \begin{pmatrix} \alpha + h_{07} \\ 8-\alpha-h_{07} \end{pmatrix} \end{aligned} \tag{10.5}$$

Wie wir an $\boldsymbol{w}' = (\alpha + h_{07}, 8 - \alpha - h_{07})^{\mathrm{T}}$ sehen können, hängt die Zahl der Bitstrings in einer Klasse nun von der Wahl von $\boldsymbol{H}$ ab. Dies ist die Folge des Blindbits, weil wegen des Blindbits die „Zeilensumme gleich Eins“-Bedingung fallengelassen wurde.

Die kalte Information am Knotenausgang ist mit (10.5):

$$I_{\mathrm{k}}^{\mathrm{o}} = 2 \cdot \boldsymbol{w}'_0 \cdot \boldsymbol{w}'_1 = 2 \cdot \begin{pmatrix} \alpha \\ 7-\alpha \end{pmatrix} \cdot \begin{pmatrix} h_{07} \\ 1-h_{07} \end{pmatrix} = 2 \cdot (\alpha\, h_{07} + (7-\alpha)(1-h_{07}))$$

Wir suchen die Nullstellen von $I_{\mathrm{k}}^{\mathrm{o}}$ unter der Nebenbedingung $h_{ij} \in [0,1]$:

$$\begin{aligned} I_{\mathrm{k}}^{\mathrm{o}} &= 2 \cdot (\alpha\, h_{07} + (7-\alpha)(1-h_{07})) = 2 \cdot (7 + \alpha(2\, h_{07} - 1) - 7h_{07}) \stackrel{!}{=} 0 \\ \Rightarrow \qquad \alpha(h_{07}) &= \frac{7\, h_{07} - 7}{2\, h_{07} - 1} \end{aligned}$$

Die Funktion $\alpha(h_{07})$ zeigt Abbildung 10.11.

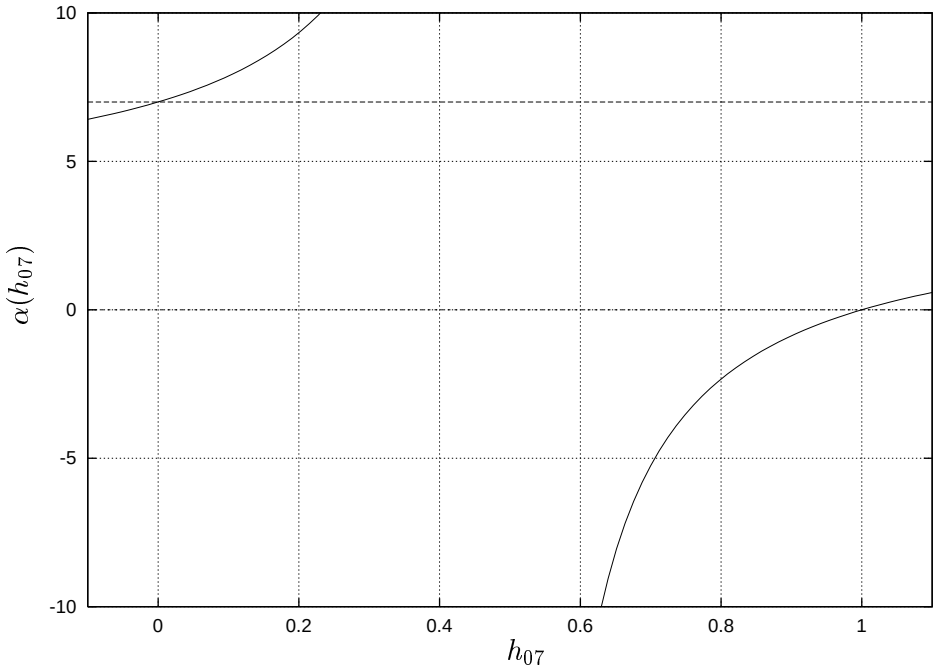


Abbildung 10.11: Die Funktion $\alpha(h_{07}) = (7h_{07} - 7)/(2h_{07} - 1)$. Sie hat eine Polstelle bei $h_{07} = 1/2$. Die gestrichelte Linie ist die Gerade $\alpha(h_{07}) = 7$.

Eine Analyse von $\alpha(h_{07})$ zeigt, daß es im Intervall $h_{07} \in [0,1]$ wegen der Bedingung $\alpha \in [0,7]$ nur zwei zulässige Punkte gibt:

$$\begin{array}{lllll} h_{07}=0 & \Rightarrow & \alpha(0)=7 & \Rightarrow & h_{0i}=1 \quad \text{mit} \quad i=0,\ldots,6 \\ h_{07}=1 & \Rightarrow & \alpha(1)=0 & \Rightarrow & h_{0i}=0 \quad \text{mit} \quad i=0,\ldots,6 \end{array}$$

Damit sind die beiden optimalen Hyperpermutationen:

$$\boldsymbol{H} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \qquad \text{bzw.} \qquad \mathrm{h} =$$

$\boldsymbol{x}$	$\boldsymbol{y}$
000	0
001	0
010	0
011	0
100	0
101	0
110	0
111	1

$$\bar{\boldsymbol{H}} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \end{pmatrix} \qquad \text{bzw.} \qquad \bar{\mathrm{h}} =$$

$\boldsymbol{x}$	$\boldsymbol{y}$
000	1
001	1
010	1
011	1
100	1
101	1
110	1
111	0

Transformieren wir die klassenspezifischen Gewichtsvektoren am Knoteneingang mit $\boldsymbol{H}$, erhalten wir:

$$\boldsymbol{H} \cdot \boldsymbol{w}_0 = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \cdot (1,1,1,1,1,1,1,0)^{\mathrm{T}} = \begin{pmatrix} 7 \\ 0 \end{pmatrix}$$

$$\boldsymbol{H} \cdot \boldsymbol{w}_1 = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \cdot (0,0,0,0,0,0,0,1)^{\mathrm{T}} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

Die sieben Bitstrings aus Klasse 0 werden auf den Zustand 0 abgebildet und der Bitstring aus Klasse 0 auf den Zustand 1.

10.3.4 Numerische Lösung mit Liniensuche

Wir suchen wieder mit dem Liniensuche-Algorithmus aus Abschnitt 8.4.2 die optimale Knotentabelle. Als Kostenfunktion verwenden wir sowohl die kalte Information I_{k} als auch die kalte Nettoinformation I_{nk}. Es wurden 1000 unabhängige Trainingsläufe durchgeführt, wobei ein Trainingslauf im Mittel 10 ms (Pentium-Rechner) dauerte, und jeder Lauf die optimale Tabelle fand. Abbildung 10.3 zeigt, nach wievielen Funktionsaufrufen die Lösung gefunden wurde.

Nun optimiert der Algorithmus jeweils eine Spalte der (2,8)-Knotenmatrix. Der Algorithmus muß im Mittel $\sum_{i=8}^{1} 8/i = 21.7$ mal würfeln, bevor er jede Spalte der Matrix mindestens einmal optimiert hat. Im besten Fall ist er nach 8 Linienminimierungen fertig, was ca. 16 Funktionsaufrufen entspricht. Wann die verschiedenen Trainingsläufe terminierten, ist in Abbildung 10.12 zu sehen und Abbildung 10.13 zeigt einen exemplarischen Trainingslauf.

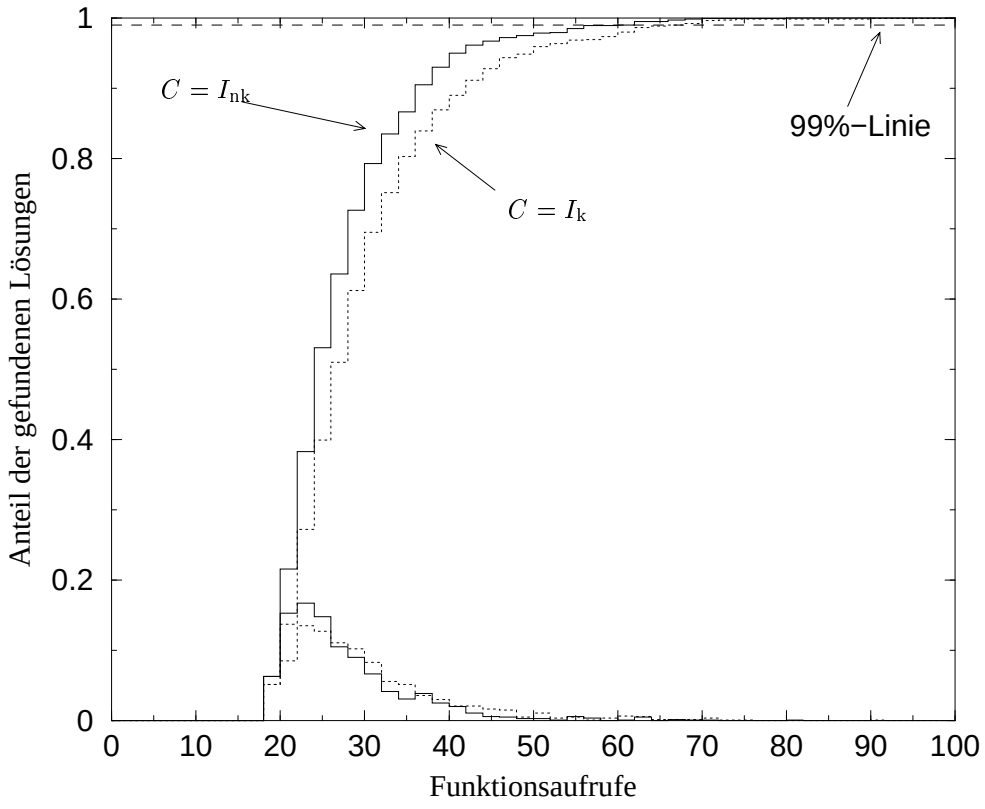


Abbildung 10.12: Relativer Anteil der gefundenen Lösungen zu der Gesamtzahl der Lösungen beim AND-Problem nach n Funktionsaufrufen. Die untere Kurve ist das Histogramm und die obere Kurve das kumulative Histogramm. Wie beim XOR-Problem terminiert der Algorithmus im Mittel schneller, wenn die Kostenfunktion die kalte Nettoinformation ist.

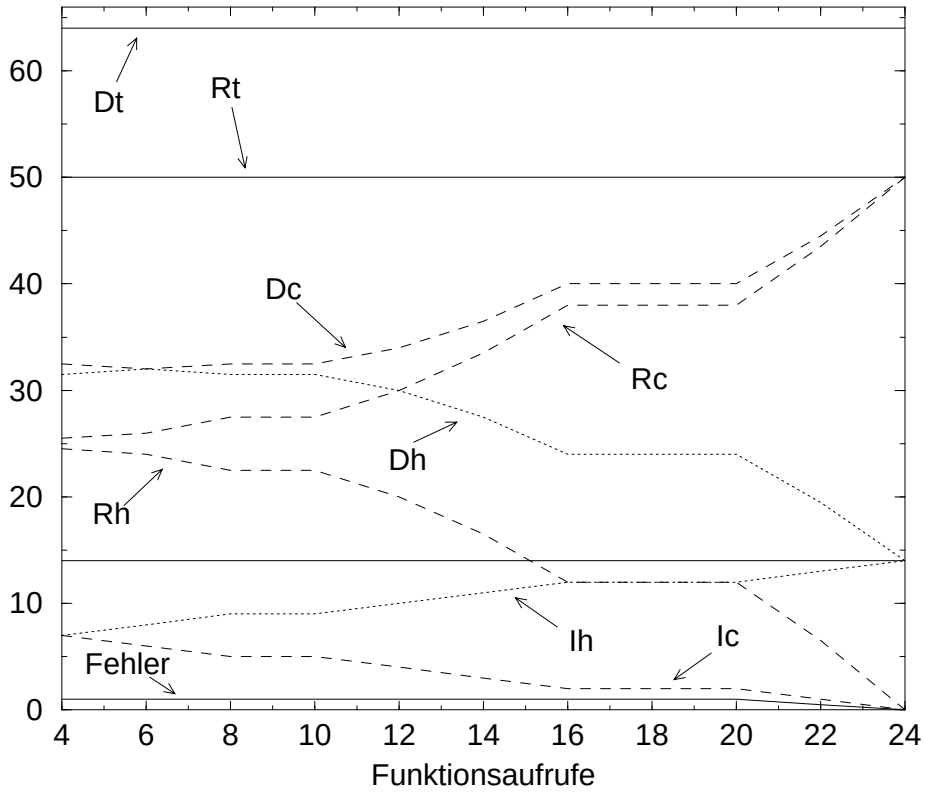


Abbildung 10.13: Trainingslauf des AND-Problems. Die Kostenfunktion war I_k. Die heißen und kalten Anteile einer statistischen Größe addieren sich immer zum Totalen dieser Größe ($D_t = D_h + D_k$ etc.), außerdem ist jeweils $D = I + R$.

10.4 Das and-Problem – Ein Knoten mit Rückkopplungen

Das Training des AND-Problems mit einem rückgekoppelten Knoten läuft genauso ab wie das Training des XOR-Problems mit einem rückgekoppelten Knoten. Die Topologie des Knotens ist praktisch identisch mit der des XOR-Problems (siehe Abbildung 10.7), nur braucht der Knoten jetzt ein Blindbit. Auch die Trainingszeiten und die Verteilung der gefundenen Lösungen sind sehr ähnlich, weswegen wir die entsprechenden Bilder an dieser Stelle nicht noch einmal wiederholen.

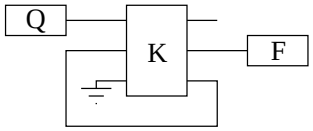


Abbildung 10.14: AND-Problem mit einem Bit Rückkopplungen. Q ist die Datenquelle, die die Trainingsdaten liefert. K ist der Knoten und F symbolisiert den Fokus auf der ausgekoppelten rückgekoppelten Leitung $y_1 = \phi^{\mathrm{ko}}{}_{1,1}(\boldsymbol{y})$ Knotens.

Wir beschränken uns hier auf die Frage, *warum* sich das AND-Problem auf n Bit mit einem Knoten mit nur einer rückgekoppelten Leitung erlernen läßt. Der Grund dafür ist der gleiche wie beim PARITY-Problem. Wir brauchen zur Realisierung eines n-dimensionalen AND Bausteins nur ein Bit Speicher, wenn wir die einzelnen Eingangsbits *nacheinander* abarbeiten:

Wir starten den Knoten mit der rückgekoppelten Leitung im Zustand 1. Nun verarbeiten wir ein Bit des Eingangsbitstrings nach dem anderen. Hat das Eingangsbit den Wert 1, bleibt auch die rückgekoppelte Leitung im Zustand 1. Hat das Eingangsbit dagegen den Wert 0, setzen wir die rückgekoppelte Leitung in den Zustand 0, in dem sie unabhängig von den noch folgenden Eingangsbits bleibt. Nachdem der gesamte Bitstring verarbeitet worden ist, erhalten wir also genau dann eine 1 auf der rückgekoppelten Leitung, wenn alle Bits des Bitstrings den Wert 1 hatten. Wie der Knoten die rückgekoppelte Leitung für $t+1$ in Abhängigkeit der rückgekoppelten Leitung und des Eingangsbits zum Zeitpunkt t setzen muß, zeigt die nachfolgende Tabelle:

r x	r
0 0	0
0 1	0
1 0	0
1 1	1

'e' steht für Eingangsbit und 'r' für Rückkopplung. Nur wenn das rückgekoppelte Bit und das Eingangsbit den Wert 1 haben, hat das rückgekoppelte Bit auch im nächsten Zeitschritt den Wert 1. Dies ist aber gerade die Wahrheitstabelle der AND-Funktion. Um also das AND-Problem auf beliebig langen Bitstrings mit Rückkopplungen zu lösen reicht es, das Ausgangsbit im Fokus eines vorwärtsgerichteten HPN-AND-Knotens mit einem seiner Eingangsbits zu verbinden.

10.5 Vergleich mit neuronalen Netzen

Da das Lernen von Boole'schen Funktion mit neuronalen Netzen intensiv untersucht wurde, vergleichen wir hier die HPN-Knoten mit neuronalen Netzen.

10.5.1 xor

Das XOR-Problem wurde im Zuammenhang mit neuronalen Netzen von sehr vielen Autoren untersucht. Wir beziehen uns hier auf die bekannten Arbeiten von Rumelhart et al. (1986). Das XOR-Problem wird auch von neuronalen Netzen innerhalb von wenigen Sekunden mit Backpropagation gelöst, weshalb das Training im Detail nicht so interessant ist. Rumelhart trainierte das XOR-Problem viele Hundert mal, wobei den Netzen die vier Trainingsmuster $200 - 600$ mal präsentiert wurden. Der HPN-Knoten benötigte im Mittel 12 Auswertungen der Eingangsstatistik (siehe Abbildung 10.3 auf Seite 137). Je mehr versteckte Neuronen ein Netz hatte, desto schneller lernte es das XOR-Problem. Zweimal terminierte das Training in einem lokalen Minimum. Wir fragen uns nun, wie sich die Netzwerktopologien unterscheiden. Dazu zeigt Abbildung 10.15 zwei minimale Lösungen des XOR-Problems aus Rumelhart et al. (1986).

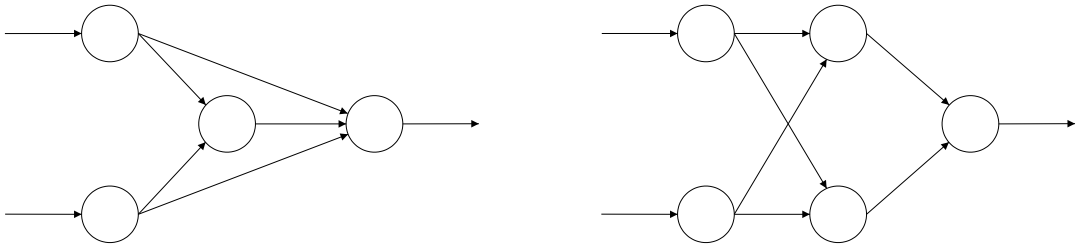


Abbildung 10.15: Neuronale Netze zur Lösung des XOR-Problem. Nur, wenn Shortcut-Verbindungen zugelassen werden, kommt ein Netz mit vier Neuronen aus (links). Ohne Shortcut-Verbindungen braucht ein neuronales Netz für das XOR-Problem fünf Neuronen.

Wie sieht es nun mit dem Ressourcenverbrauch aus? Die folgende Tabelle stellt die benötigten Ressourcen gegenüber.

Netzwerk	Neuronen / HPN-Knoten	Gewichte / Tab.-Einträge	Speicher
Neuronales Netz	4	5	20 Byte
Neuronales Netz	5	6	24 Byte
HPN	1	4	4 Bit

Hier wurde wie sonst auch damit gerechnet, daß die Gewichte der neuronalen Netze nur mit einfacher Genauigkeit (d. h. 4 Byte pro Gewicht) gespeichert werden. Der HPN-Knoten braucht also deutlich weniger Speicher als die beiden neuronalen Netze. Zudem braucht selbst das kleinere neuronale Netz $1+2 = 3$ Additionen und $3+2 = 5$ Multiplikationen um einen Eingangsvektor zu verarbeiten, wogegen der HPN-Knoten nur einen Tabellennachschlag benötigt.

10.5.2 parity

Vom PARITY-Problem zeigten Minsky und Papert (1988), daß es sich mit einem linearen Perzeptron nicht lösen läßt. Man kann das PARITY-Problem der Dimension n aber mit einem Multi-Layer-Perzeptron lösen, wobei meistens zwischen n (Rumelhart et al., 1986) und $2n$ (Fahlman, 1989) versteckte Gewichte verwendet werden. Trotzdem ist das PARITY-Problem für neuronale Netze ein schwieriges Problem. Für $n = 2, \ldots, 8$ wachsen die Trainingszeiten mit 4^n (Fahlman, 1989).

Der interessante Punkt bei Hyperpermutationsnetzwerken ist, daß man das PARITY-Problem einfach serialisiert mit einem rückkgekoppelten Knoten trainieren kann. Man vermeidet dadurch das Anwachsen der Netzdimension bei länger werdenden Eingangsvektoren.

10.5.3 and

Beim AND-Problem sind die Verhältnisse anders als beim XOR-Problem. Das AND in n Dimensionen zeichnet sich dadurch aus, daß von den möglichen 2^n Eingangsvektoren einer, nämlich der mit ausschließlich Einsen, von allen anderen Eingangsvektoren separiert werden muß. Wir können uns leicht ein Schwellwert-Neuron überlegen, das diese Aufgabe löst: Das Neuron hat n Eingänge, alle Gewichte sind 1 und die Schwelle ist n. Dieses Neuron hat genau dann die Ausgabe 1, wenn alle Eingänge den Wert 1 haben, sonst ist der Ausgang des Neurons 0. Die beiden Klassen des AND-Problems sind also linear separierbar, womit das Problem für ein neuronales Netz (bzw. ein Neuron) geradezu prädestiniert ist und es von einem Neuron entsprechend schnell (innerhalb einer Sekunde mit Backpropagation) gelernt wird.

Wie ist der Ressourcenverbrauch beim AND-Problem für ein Neuron bzw. einen HPN-Knoten, ein baumartiges Hyperpermutationsnetz mit 2-Bit-Knoten und einen rückgekoppelten Knoten? Tabelle 10.5.3 zeigt den Ressourcenverbrauch für das n-dimensionale AND-Problem mit $n \in \{2, 4, 8, 2^k\}$.

Netzwerk	Neur. / HPN-Knoten	Gewichte / Tab.-Einträge	Speicher
Neuron	1	2	4 Byte
HPN-Knoten	1	4	4 Bit
HPN, baumartig	1	4	4 Bit
RHPN-Knoten	1	4	4 Bit
Neuron	1	4	16 Byte
HPN-Knoten	1	16	2 Byte
HPN, baumartig	$2+1=3$	$3 \cdot 4 = 12$	12 Bit
RHPN-Knoten	1	4	4 Bit
Neuron	1	8	32 Byte
HPN-Knoten	1	256	32 Byte
HPN, baumartig	$4+2+1=7$	$7 \cdot 4 = 28$	28 Bit
RHPN-Knoten	1	4	4 Bit
Neuron	1	2^k	2^{k+2} Byte
HPN-Knoten	1	2^{2^k}	2^{2^k-3} Byte
HPN, baumartig	$2^k - 1$	$(2^k - 1) \cdot 4$	$(2^k - 1)/2$ Byte
RHPN-Knoten	1	4	4 Bit

Tabelle 10.1: Der Ressoucenverbrauch beim AND-Problem. Der vier Blöcke zeigen die nötigen Ressourcen für $n = \{2, 4, 8, 2^k\}$. Löst man das AND-Problem mit nur einem rückkopplungsfreien HPN-Knoten, wächst der Speicherplatz mit n exponentiell an. Daher macht es für größere n nur Sinn, ein baumartigen HPN bzw. einen rückgekoppelten Knoten zu verwenden. Das baumartige HPN braucht für $n = 2^k$ Eingangsbit $2^k - 1$ Knoten in Netz. Es wächst also wie das Neuron linear mit der Zahl der Eingangsdimensionen n. Der rückgekoppelte Knoten, der das Eingangsmuster seriell verarbeitet, wächst bei steigender Dimension gar nicht.

11 Das T-C Problem

Auch das T-C Problem, ursprünglich von Rumelhart et al. (1986, Seite 348 ff.), wurde schon als Testproblem für neuronale Netze verwendet, u. a. von Minsky und Papert (1988); Cowan und Sharp (1988); Zagorski (1994) und McDonnel und Waagen (1993).

11.1 Was ist das T-C Problem?

Beim T-C Problem geht es darum, ein 'T' von einem 'C' zu unterscheiden, das an einer beliebigen Stelle auf einem Bild mit einer von vier möglichen Orientierungen auftritt. Die einzelnen Pixel des Bildes sind zweiwertig, es gibt also nur „Pixel gesetzt (schwarz)" oder „Pixel nicht gesetzt (weiß bzw. Hintergrund)". Das 'T' und das 'C' zeigt Abbildung 11.1.

Abbildung 11.1: 'T' und 'C' in ihren vier möglichen Orientierungen. Jedes Zeichen besteht aus fünf Pixeln.

11.2 Die Ordnung des T-C Problems

Minsky und Papert verwenden den Begriff der *Ordnung*, um die Lokalität eines Problems zu quantifizieren. Dies soll hier kurz erläutert werden, um zu verstehen, warum das T-C Problem nicht ganz einfach ist und es daher „wert ist zu studieren" (Terry Sejnowski). Die „Erkennungsmaschine", die Minsky und Papert untersuchten, ist das *Perzeptron*.

Sei nun X eine geometrische Figur (z. B. ein Quadrat) in der Ebene $\mathbb{R}^2$, also eine Teilmenge der Punkte aus $\mathbb{R}^2$. Sei $\Psi(X)$ eine Boole'sche Funktion auf $\mathbb{R}^2$, ein sog. Prädikat. Mit Hilfe von $\Psi(X)$ können wir etwas über die Eigenschaften von X erfahren. Einige Beispiele für Prädikate sind:

$$\Psi_{\text{Kreis}}(X) = \begin{cases} 1 & \text{falls } X \text{ ein Kreis ist,} \\ 0 & \text{sonst.} \end{cases}$$

$$\Psi_{\text{konvex}}(X) = \begin{cases} 1 & \text{falls } X \text{ konvex ist,} \\ 0 & \text{sonst.} \end{cases}$$

$$\Psi_p(X) = \begin{cases} 1 & \text{falls } p \text{ in } X \text{ ist,} \\ 0 & \text{sonst.} \end{cases} \qquad (p \text{ sei ein Punkt in } \mathbb{R}^2.)$$

Das letzte Prädikat ist ein Beispiel für ein sehr lokales Prädikat: es bewertet nur einen Punkt p in der Ebene. Das Perzeptron ist eine Schwellwertfunktion, die die Werte von mehreren (meist

lokalen) Prädikaten $\phi_1, \phi_2, \ldots, \phi_n$ (mit $\phi_i(Y) \in \{0,1\}$ und $Y \subseteq \mathbb{R}^2$) gewichtet aufsummiert und gegen einen Schwelle prüft:

$$\Psi(X) = \begin{cases} 1 & \text{falls } \sum_{\varphi \in \Phi} \alpha_\varphi \, \varphi(X) > \theta \,, \\ 0 & \text{sonst.} \end{cases} \tag{11.1}$$

Hierbei sind die α_φ die Gewichte und θ ist die Schwelle.

Etwas vereinfacht gesagt gibt die Ordnung eines Unterscheidungsproblems die kleinste Pixelanzahl an, die man für ein Prädikat $\varphi \in \Phi$ benötigt, um die zu unterscheidenden Muster mit einem Perzeptron noch voneinander trennen zu können. Dementsprechend gilt ein Problem für ein Perzeptron als um so schwieriger zu lernen, je höher die Ordnung ist. Beispielsweise ist das Problem, die beiden Muster in Abbildung 11.2 zu unterscheiden, von der Ordnung eins. Die beiden Muster haben nämlich eine unterschiedliche Zahl schwarzer Pixel. Daher ist es ausreichend, ein Pixel nach dem anderen lokal zu betrachten und die Zahl der gesetzten Pixel pro Muster zu zählen. Dann lassen sich anhand dieser Zahl die beiden Muster voneinander unterscheiden.

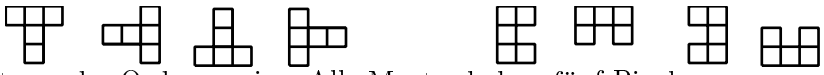
Abbildung 11.2: Unterscheidungsproblem der Ordnung eins. Die beiden Muster lassen sich dadurch unterscheiden, daß man die gesetzten Pixel zählt.

Das Unterscheidungsproblem in Abbildung 11.3 ist dagegen von der Ordnung zwei. Man muß also mindestens Pixelpaare betrachen, um die beiden Muster unterscheiden zu können. Das linke Muster hat die Pixelpaare $2 \times$  und $1 \times$  . Dagegen hat das rechte Muster die Pixelpaare $1 \times$  , $1 \times$  und $1 \times$  . Die Häufigkeiten der lokalen Untermuster der Ordnung n eines Musters heißt auch *Spektrum* der Ordnung n, hier also Spektrum der Ordnung zwei. Sind wie in diesem Beispiel die beiden Spektren verschieden, lassen sich die beiden Muster bereits innerhalb dieser Ordnung unterscheiden.

Abbildung 11.3: Unterscheidungsproblem der Ordnung zwei. Beide Muster haben drei Pixel. Daher lassen sie sich nicht durch die Zahl der Pixel unterscheiden.

Bei allen Spektren mit einer Ordnung größer als eins kann man nun noch weitergehen, indem man für eine Unterscheidungsfunktion Translations- und Rotationsinvarianz fordert und sich damit mehr auf die „geometrische Eigenschaft“ (Minsky) eines Musters konzentriert. Das bedeutet aber auch, das nun die Paare , oder  ,  etc. ununterscheidbar werden.

Betrachten wir nun das T-C Problem. Das ‘T’ läßt sich von dem ‘C’ weder im Spektrum der Ordnung eins noch im Spektrum der Ordnung zwei unterscheiden, wie Abbildung 11.4 zeigt. Man muß also zum Spektrum der Ordnung drei übergehen, um die Muster voneinander trennen zu können. Konkret bedeutet das, ein Entscheidungskriterium muß mindestens *Pixeltripel* zur Unterscheidung heranziehen. Daher ist das Erlernen eines Entscheidungskriteriums für ein Problem höherer Ordnung schwieriger als für ein Problem niedrigerer Ordnung.



Spektrum der Ordnung eins: Alle Muster haben fünf Pixel
Spektrum der Ordnung zwei: Alle Muster haben:

- 4 ×  oder 
- 2 ×  oder 
- 2 ×  oder 
- 1 ×  oder 
- 1 ×  oder 

Abbildung 11.4: Spektren der Ordnung eins und zwei von 'T' und 'C'. Sie sind für alle acht Muster gleich, weshalb man zur Unterscheidung mindestens zum Spektrum der Ordnung drei übergehen muß.

11.3 Trennung der 'T' von den 'C' mit einem rückgekoppelten Knoten

Wie schon erwähnt, sollen Bilder voneinander unterschieden werden, die an einer beliebigen Stelle ein 'T' oder 'C' in einer der vier möglichen Orientierungen enthalten, wie in Abbildung 11.5 gezeigt. Ein rückgekoppelter Knoten soll also lernen, alle Bilder mit einem 'T' der einen und alle Bilder mit einem 'C' der anderen Klasse zuzuordnen.

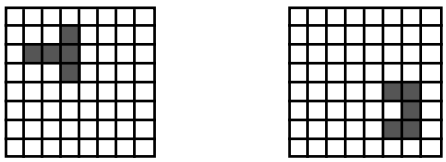


Abbildung 11.5: 8 × 8 Pixel großes Bild, links ein rotiertes 'T' ('T'-Bild), rechts ein rotiertes 'C' ('C'-Bild).

Zunächst stellt sich die Frage, wie man das zweidimensionale Bild serialisiert, um es dem Knoten als Eingabe zu präsentieren. Die naheliegende Möglichkeit ist, das Bild wie in Abbildung 11.6 zeilenweise abzuarbeiten und so einen 1 Bit breiten Datenstrom zu generieren.

Die Anordnung des rückgekoppelten Knotens ist nun so wie in Abbildung 11.7 skizziert: Der Datengenerator erzeugt einen Datenstrom, der aus einem serialisierten Bild besteht. Pro Zeittakt wird dem Knoten ein Bit über die erste Eingangsleitung zugeführt. Das Symbol am Eingang des Knotens setzt sich aus den Werten der Eingangsleitungen $\boldsymbol{x}$ und den Werten der rückgekoppelten Leitungen $\boldsymbol{r}$ zusammen. Bei jedem Zeittakt transformiert der Knoten das

Abbildung 11.6: Serialisierung eines 4×4-Bildes. Eine Eins repräsentiert ein gesetztes Pixel, eine Null ein nicht gesetztes Pixel.

Eingangssymbol mit Hilfe einer Hyperpermutation in ein Ausgangsmuster, welches er auf seine Ausgangsleitungen schreibt.

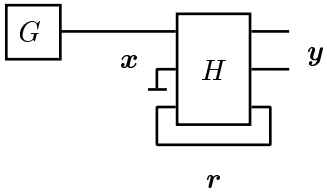


Abbildung 11.7: Datengenerator G mit nachgeschaltetem rückgekoppeltem Hyperpermutationsknoten. Der Datengenerator liefert zeilenweise die Bits des Bildes, die in den rückgekoppelten Knoten als Eingangsdaten gehen. Jede Leitung in dem Bild entspricht einer *oder mehreren* Leitungen im Modell. Die Eingangsleitung(en) mit einem stilisierten Erdungssymbol sind Blindbits.

Die Klassifikation läuft nun folgendermaßen ab: Wenn der Knoten startet, sind alle rückgekoppelten Leitungen gleich null, nur die Eingangsleitung hat den Wert des ersten Bits des Bildes. Nun werden alle Bits des Bildes nacheinander vom Knoten verarbeitet, dann wird das System angehalten. Den Zustand, in dem sich die rückgekoppelten Leitungen jetzt befinden, bezeichnen wir als *Endzustand*. Wir bekommen also für jedes Bild genau einen Endzustand. Wir können zwei Bilder genau dann unterscheiden, wenn sie zu zwei verschiedenen Endzuständen führen. In der Lernphase werden wir die Knotentabelle so organisieren, daß die 'T'-Bilder andere Endzustände liefern als die 'C'-Bilder. Der Knoten soll also für verschiedene Klassen in verschiedenen Unterräumen des Zustandsraums anhalten. Das Training wurde wie im vorherigen Abschnitt durchgeführt, in dem die kalte Nettoinformation minimiert wurde.

11.4 Ergebnisse

Es wurden verschiedene Fenstergrößen von 3×3 bis 20×20 Pixel getestet. Tabelle 11.1 zeigt einige ausgewählte Bild- und Knotengrößen, wobei die Klassifikation der 'T'- und 'C'-Bilder immer fehlerfrei war.

Fenstergröße	Rückkopplungen	Blindbits	Gesamtzahl Bilder	Gesamtzahl Bits
4×4	2	2	32	512
6×6	3	1	128	4608
16×16	4	1	1568	401408

Tabelle 11.1: Knotengrößen für verschiedene Bildgrößen des T-C Problems. Die Spalte „Gesamtzahl Bilder" gibt an, wieviele verschiedene 'T'- bzw. 'C'-Bilder bei einem $n \times n$ -Fenster überhaupt möglich sind. Die Spalte „Gesamtzahl Bit" zeigt die Größe des kompletten Datenstroms in Bit.

Bei den Experimenten zeigte sich, daß ein Knoten mit nur zwei rückgekoppelten Leitungen in der Lage war, *verschieden große* 'T'- und 'C'-Bilder fehlerfrei zu klassifizieren, wenn die Seitenlänge der Bilder eine ungerade Pixelanzahl hatte. Daß diese Beobachtung Allgemeingültigkeit hat, zeigt uns

Satz 10. *Sei $\mathcal{M}$ eine Menge mit 'T'- und 'C'-Bildern verschiedener Größen im Sinne des T-C Problems, wobei die Seitenlänge der Bilder $2n+1$ mit $n \geq 2$ ist. Ein rückgekoppelter HPN-Knoten erhält die Bilddaten, indem die Bilder serialisiert in ein Eingangsbit des Knotens eingespeist werden. Dann ist ein rückgekoppelter Knoten mit der richtigen Knotentabelle und zwei rückgekoppelten Leitungen in der Lage, alle 'T'- und 'C'-Bilder aus $\mathcal{M}$ korrekt zu klassifizieren.*

Beweis. Es gibt vier verschiedene 'T'- und vier verschiedene 'C'-Orientierungen der Seitenlänge $2n+1$ und pro Orientierung $(2n+1-2)^2$ verschiedene Positionen. Wir unterteilen die Bilder nach den vier verschiedenen 'T'- und vier verschiedenen 'C'-Orientierungen für alle $n \geq 2$. Jetzt haben wir nur noch acht verschiedene Fälle, für die wir in Tabelle 11.2, zeigen, daß die zwei rückgekoppelten Leitungen eines Knotens für die 'T'-Bilder in einem anderen Zustand anhalten als für die 'C'-Bilder, wenn die Knotentabelle

x	y
000	00
001	01
010	10
011	11
100	11
101	01
110	10
111	00

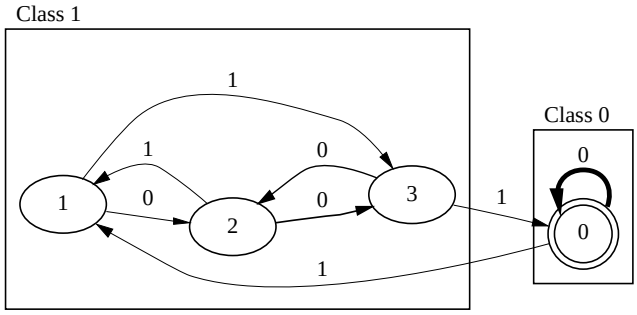
ist. Da jedes Ausgangsbitmuster zwei mal auftritt, kommt der Knoten ohne Blindbits aus. Das dritte Ausgangsbit ist nicht aufgeführt, da es hier nicht benötigt wird. Die Knotentabelle wurde durch Training ermittelt. □

11.5 Vergleich mit neuronalen Netzen

Die bisher durchgeführten Experimente zum T-C Problem wurden mit neuronalen Netzen durchgeführt. Rumelhart et al. (1986) und Cowan und Sharp (1988) verwendeten zum Training ihrer neuronalen Netze rezeptive Felder. Die Zahl der versteckten Neuronen war bei einer Seitenlänge der Bilder von x Pixeln jeweils $(x-2)^2$. Leider gaben die Autoren keine weiteren Daten zum Training an, nur, daß die acht Trainingsbilder $5000-10000$ mal präsentiert wurden, bevor die neuronalen Netze das T-C Problem gelernt hatten.

McDonnel und Waagen (1993) und Zagorski (1994; 1998) haben dagegen zum Training des T-C Problems genaue Angaben gemacht. Dabei beschränkten sich die Autoren auf das T-C Problem mit Bildern einer Seitenlänge von vier Pixeln. Diese Beschränkung kam dadurch zustande, daß die Zahl der Eingangsneuronen eines voll verbundenen neuronalen Netzes quadratisch mit der Seitenlänge der Bilder wächst.

Würden wir nun den rückgekoppelten HPN-Knoten mit einem voll verbundenen neuronalen Netz vergleichen, das das T-C Problem lösen kann, wäre das neuronale Netz aufgrund der 16 Eingangs- und 16 versteckten Neuronen viel größer, als der rückgekoppelte Knoten. Daher



	0	1 1 1	0	1	0	1	0
N	≥ 0	1	u	1	g	1	≥ 1
Z	0	0	0	1	3	0	**0**
	0	1	0	1 1 1	0	1	0
N	≥ 0	1	g	1	g	1	≥ 2
Z	0	1	3	3	3	0	**0**
	0	1	0	1	0	1 1 1	0
N	≥ 0	1	g	1	u	1	≥ 0
Z	0	1	3	0	0	0	**0**
	0	1	0	1 1 1	0	1	0
N	≥ 0	1	g	1	g	1	≥ 0
Z	0	1	3	3	3	0	**0**

	0	1 1	0	1	0	1 1	0
N	≥ 0	1	u	1	g	1	≥ 0
Z	0	3	2	1	3	1	**1/2/3**
	0	1	0	1	0	1 1 1	0
N	≥ 0	1		1	g	1	≥ 0
Z	0	1	2	1	3	3	**2/3**
	0	1 1	0	1	0	1 1	0
N	≥ 0	1	g	1	u	1	≥ 0
Z	0	3	3	0	0	3	**2/3**
	0	1 1 1	0	1	0	1	0
N	≥ 0	1	g	1		1	≥ 0
Z	0	0	0	1	2	1	**1/2/3**

Tabelle 11.2: Haltezustände für 'T'- und 'C'-Bilder mit ungerader Seitenlänge. Jeder der acht Blöcke der Tabelle zeigt eine der möglichen Orientierungen. In der obersten Zeile eines Blockes stehen die Eingangsbitmuster, die für das serialisierte Bild entstehen. Die Zeile „N" gibt an, wie oft hintereinander das darüber stehende Eingangsbitmuster auftritt. Dabei steht „u" für eine ungerade und „g" für eine gerade Zahl. Die Zeile „Z" gibt an, welchen Zustand die rückgekoppelten Leitungen haben, wenn das darüber stehende Eingangsbitmuster verarbeitet worden ist. Wie wir sehen, ist der Haltezustand der rückgekoppelten Leitungen in der rechten Spalte für die 'T'- und 'C'-Bilder verschieden. Das Bild oberhalb der Tabelle zeigt den Übergangsgraphen der rückgekoppelten Zustände, Start ist im Zustand 0. Die Zahlen an den Kanten geben den Wert des Eingangsbits an. Die Dicke der Kanten zeigt, wie oft eine Kante auf den Trainingsdaten durchlaufen wurde.

vergleichen wir den rückgekoppelten Knoten mit neuronalen Netzen, bei denen schon mit einem Pruning-Verfahren nicht benötigte Neuronen entfernt wurden. Abbildung 11.8 zeigt drei verschiedene Netzwerktopologien und Tabelle 11.5 die Zahl der Gewichte pro Netzwerk.

Netzwerk	Topologie	Gewichte	Speicher (Byte)
a) MLP voll verb.	16-16-1	288	1152
b) MLP aus McDonnel und Waagen (1993)	15-7-1	60	240
c) MLP aus Zagorski (1994)	10-2-1	27	108
d) HPN-Knoten	2 Rückk.	8 Tab.-Eintr.	2

Tabelle 11.3: Zahl der Gewichte von MLPs zur Lösung des T-C Problems der Seitenlänge vier. Werden die Gewichte als Gleitkommazahlen mit einfacher Genauigkeit gespeichert, benötigt ein Gewicht vier Byte Speicherplatz. Das kleinste neuronale Netz kommat damit auf 108 Byte Speicherplatz für die Gewichte. Ein rückgekoppelter HPN-Knoten braucht für das T-C Problem nur zwei Rückkopplungen. Die Knotentabelle hat 8 Einträge mit je zwei Bit, der Speicherplatz beträgt somit 16 Bit = 2 Byte. Bei Bildern mit ungerader Seitenlänge kommt der rückgekoppelte HPN-Knoten sogar für beliebige Bildgrößen nur mit zwei Rückkopplungen aus.

Wir sehen hier unmittelbar, daß sich für bestimmte Probleme eine diskrete rückgekoppelte Stuktur vorteilhaft erweist:

1. Die Stuktur ist sehr klein und kompakt.
2. Sie kann Merkmalsvektoren, in diesem Fall serialisierte Bilder, verschiedener Länge verarbeiten. Werden die Merkmalsvektoren länger, muß die rückgekoppelte Stuktur nicht notwendigerweise mitwachsen.

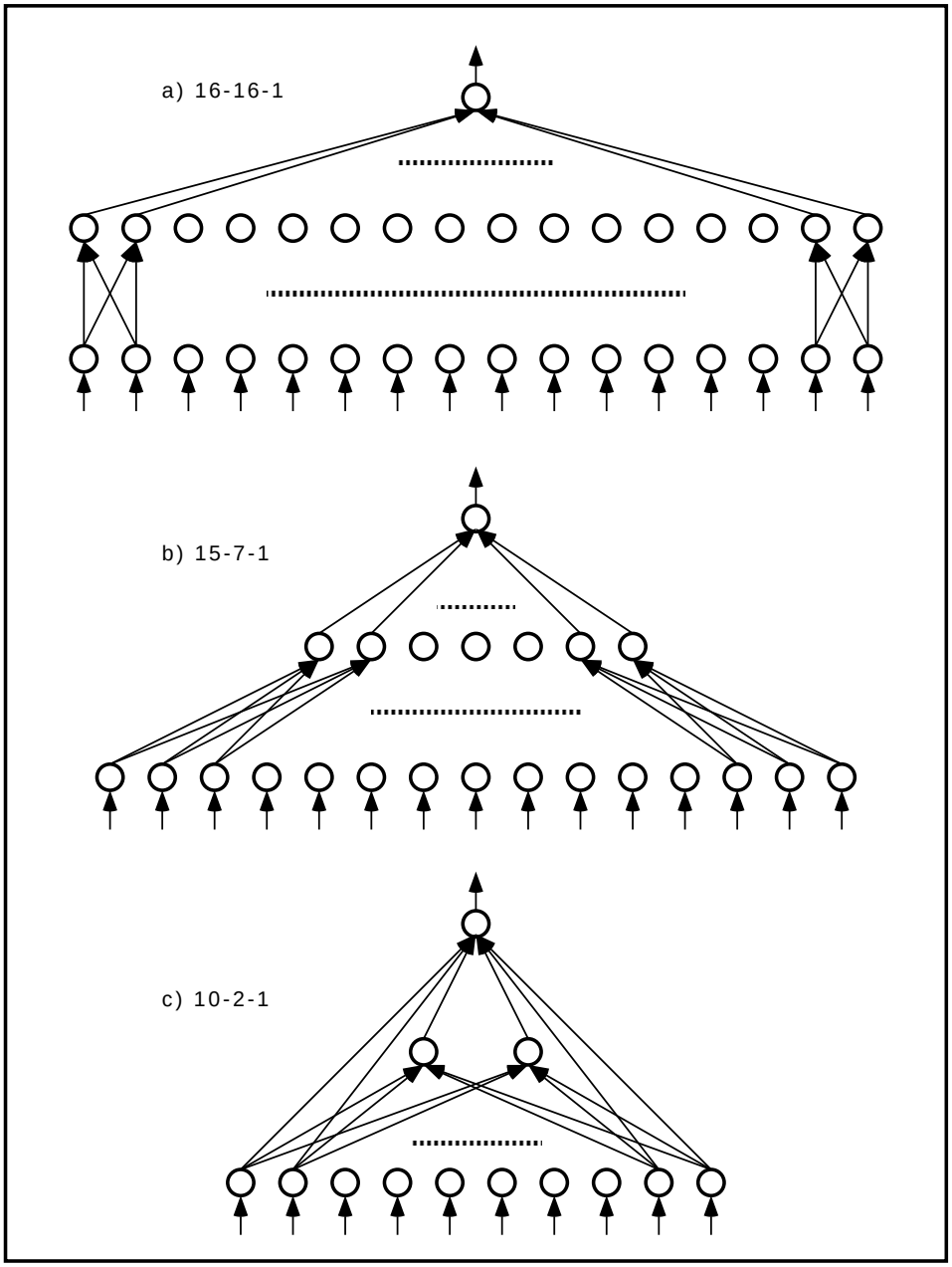


Abbildung 11.8: Drei neuronale Netze zur Lösung des T-C Problems. Netzwerk a) ist das normale vollverbundene MLP mit einer versteckten Schicht. Netzwerk b) wurde vom Pruning-Algorithmus aus McDonnel und Waagen (1993) und Netzwerk c) vom Pruning-Algorithmus aus Zagorski (1994) gefunden (Abbildung aus (Zagorski, 1994)).

12 Die Sitzbelegungserkennung

In diesem Kapitel trainieren wir ein rückgekoppeltes Hyperpermutationsnetzwerk mit realen Daten und vergleichen es mit klassischen Ansätzen.

12.1 Das Anwendungsszenario

Bei dieser Anwendung geht es um den Airbag, der in den meisten modernen PKWs zumindest auf der Fahrer- und Beifahrerseite eingebaut wird.

12.1.1 Der Airbag

Die Aufgabe eines Airbags ist, bei einem Unfall die Insassen vor einer Kollision mit harten Gegenständen des Fahrzeuginnenraums, z. B. dem Lenkrad, zu schützen. In der Anfangszeit des Airbags gab es den Airbag nur auf der Fahrerseite. Beim Fahrer-Airbag ist die Steuerung zur Auslösung des Airbags relativ einfach: Wenn Sensoren am Fahrzeug melden, das es zu einem Unfall kommt, wird die Treibladung des Airbags gezündet und der Airbag bläst sich auf.

Auf der Fahrerseite ist man sich sicher, daß dort ein Mensch sitzt, wenn das Fahrzeug gefahren wird. Beim Airbag auf der Beifahrerseite ist die Lage komplizierter. Dort kann auch ein Mensch sitzen, ebenso kann der Beifahrersitz aber auch leer sein, oder es kann ein Kind in einem Kindersitz darauf festgeschnallt sein. Befindet sich auf dem Beifahrersitz ein Kindersitz, kann es für das Kind wichtig sein, daß sich der Airbag trotz eines Unfalls *nicht* aufbläst. Denn der Airbag ist dafür ausgelegt, den Kopf eines Erwachsenen zu schützen. Dadurch ist er aber wegen der anderen Geometrie gleichzeitig nicht in der Lage, ein in einem Kindersitz liegendes Kind zu schützen. Im schlimmsten Fall könnte er es sogar verletzen. Je nach Belegung des Beifahrersitzes ergeben sich für eine Unfallsituation folgende Vorgaben:

- Beifahrersitz ist leer: Der Airbag sollte nicht gezündet werden. Diese Forderung ergibt sich aus der Tatsache, daß jede Zündung eines Airbags Schäden am Innenraum des Fahrzeugs verursacht. Daher sollte man ihn nur dann zünden, wenn es nötig ist. Es ist aber nicht so schlimm, wenn er bei leerem Beifahrersitz trotzdem gezündet wird.
- Auf dem Beifahrersitz sitzt ein Erwachsener: Der Airbag muß auf jeden Fall gezündet werden.
- Auf dem Beifahrersitz befindet sich ein Kind liegend in einem Kindersitz: Der Airbag darf auf keinen Fall gezündet werden.

Ein Fahrzeug mit Beifahrer-Airbag sollte also die Möglichkeit haben, diesen abzuschalten.

Normalerweise wird eine solche Abschaltung manuell vorgenommen, der Airbag wird mit einem Schalter ein- und ausgeschaltet. Eine manuelle Abschaltung hat aber den Nachteil, daß sie vergessen werden kann. Daher wird schon seit längerem daran gearbeitet, diese Abschaltung automatisch vorzunehmen. Ein schon praktiziertes Verfahren ist, daß sich ein Kindersitz über Funk „beim Fahrzeug meldet“ und die Abschaltung veranlaßt. Dazu muß aber ein solcher

spezieller Kindersitz vorhanden sein. Wir werden die Möglichkeit verfolgen, die Art der Person auf dem Beifahrersitz automatisch zu erkennen, so daß das Fahrzeug den Airbag von selbst abschalten kann.

12.1.2 Die Hardware

Die Hardware am Anfang des Erkennungsprozesses ist der Sensor. Um zu erkennen, wer oder was sich auf dem Beifahrersitz befindet, benötigen wir einen Sensor, der die Untschiede zwischen einem leeren Sitz, einem Erwachsenen und einem Kindersitz prinzipiell erfassen kann. Von Mercedes wurde zu diesem Zweck ein achtstrahliger Infrarotsensor ausgewählt, der mit jedem seiner acht Strahlen Entfernungen messen kann. Dieser Sensor „sieht“ von der Sonnenblende auf den Beifahrersitz herab und mißt das Entfernungprofil dessen, was sich gerade auf dem Beifahrersitz befindet. Die Entfernung gibt der Sensor pro Kanal auf einem sechs Bit breiten Datenausgang aus. Jede Messung besteht damit aus acht Datenworten mit sechs Bit Auflösung. Mit diesem Sensor wurden bei Mercedes in einem Fahrzeug Trainingsdaten generiert, indem die Meßwerte des Sensors für den leeren Sitz, Kinder in Kindersitzen und verschiedene Erwachsene auf dem Sitz aufgezeichnet wurden.

Die vorgegebenen Randbedingungen für die Hardware des Erkennungsprozesses selbst waren sehr restriktiv. Es gab die Randbedingung, daß sowenig RAM wie möglich verwendet werden sollte, maximal aber wenige hundert Byte. Dies lag daran, daß die sog. Sitzbelegungserkennung nicht auf einer eigenen Hardware, sondern auf einer schon bestehenden Hardware als zusätzliches Modul laufen sollte. Der auf der Hardware integrierte Prozessor beherrschte nur Ganzzahlarithmetik mit 8 bzw. 16 Bit. Daraus ergab sich das Ziel, die Erkennungsaufgabe als solche möglichst gut zu lösen, gleichzeitig aber den Ressourcenverbrauch so gering wie möglich zu halten. Denn selbst ein Erkennungssystem, das die Aufgabe perfekt löst, hilft nicht weiter, wenn es nicht in die vorgegebene Hardware paßt.

12.2 Training des rückgekoppelten Netzwerks

Im Prinzip wurde wie in den schon vorgestellten Beispielen trainiert, nur daß es sich diesmal um reale Daten handelte. Da wir die globale Netzstatistik zum Training verwendet haben, konnten wir, so wie in Abschnitt 9.1.2.2 auf Seite 123 beschrieben, bei der Kostenfunktion zusätzlich zur kalten Nettoinformation die Fehlerrate mit einfließen lassen.

12.2.1 Die Trainingsdaten

Die Trainingsdaten bestanden aus 4195 Lern- und 2096 Testdatenvektoren. Jeder Datenvektor bestand aus acht natürlichen Zahlen mit sechs Bit Auflösung, z. B. $(14, 9, 17, 28, 11, 8, 12, 3)$, wobei jede Zahl für die gemessene Entfernung eines Sensorstrahls steht. Die Daten waren ursprünglich in fünf Klassen aufgeteilt, wobei drei Klassen verschiedene Variationen von Kindersitzen waren. Bei zwei der drei Kindersitz-Klassen sitzt das Kind mit dem Gesicht nach vorne, bei einer Kindersitz-Klasse liegt das Kind auf dem Rücken mit dem Kopf nach vorne im Kindersitz. Da der Airbag auch auslösen soll, wenn das Kind mit dem Gesicht nach vorne sitzt, wurden diese zwei Kindersitz-Klassen und die Erwachsenen-Klasse zu einer neuen „Erwachsenen-Klasse“ zusammengelegt, so daß die drei Klassen „Leer“, „Erwachsener“ und „Kindersitz“ trainiert wurden. Die Zahl der Lern- und Testdatenvektoren teilten sich folgendermaßen auf die drei Klassen auf:

Klasse	Lerndaten	Testdaten
Leer	244	122
Erwachsener	1352	674
Kindersitz	2600	1300

12.2.2 Parallele Verarbeitung gleichrangiger Bits verschiedener Sensoren

Es wurden hauptsächlich zwei verschiedene Netzwerktopologien trainiert, die sich in der zeitlichen Verarbeitung der Datenbits unterschieden. Die parallele Verarbeitung gleichrangiger Bits verschiedener Sensoren bedeutet folgendes: Beim ersten Zeittakt der Verarbeitung eines Datenvektors wird von jeder der sechs Bit breiten Zahlen des Datenvektors das höchstwertige Bit in das RHPN eingespeist. Beim ersten Takt verarbeitet des RHPN also die acht höchstwertigen Bits der Sensoren. Beim nächsten Takt wird das dem höchstwertigen Bit folgende Bit vom RHPN verarbeitet. Dies wird fortgeführt, bis auch das letzte Bit von Netz verarbeitet wurde. Dann geht, wie üblich, der Zustand der Ausgangsleitungen des Netzwerks in die Traininsstatistik ein. Eine der untersuchten Netztopologien für diese Art der Serialisierung wurde schon in Abbildung 9.1 auf Seite 122 gezeigt.

Das eigentliche Training wurde sowohl mit der stochastischen Richtungswahl als auch mit Simulated Annealing vorgenommen. Es zeigte sich, daß bei rückgekoppelten HPN die stochastische Richtungswahl um eine Größenordnung langsamer als Simulated Annealing ist. Das liegt daran, daß Simulated Annealing gleich auf den deterministischen Knotentabellen operiert und sich die bei der kontinuierlichen Optimierung auftretenden Matrizenmultiplikationen sparen kann. Da sich die Güten der trainierten Netze in der Regel kaum unterschieden, wurden nach anfänglichen Tests beider Verfahren die weiteren Optimierungen nur noch mit Simulated Annealing durchgeführt.

Simulated Annealing findet ein globales Minimum nach der Theorie nur dann mit Wahrscheinlichkeit Eins, wenn die Zahl der Iterationen gegen Unendlich geht. Nun kann man nicht „so lange waren“, weswegen man stattdessen den Algorithmus abbricht, wenn die Kostenfunktion eine bestimmte Zahl von Iterationen nicht mehr kleiner wird. Bei den gewählten Parametern dauerte ein Optimierungslauf um die 30 Minuten. Um nun eine Aussage über die Lage der erreichten Minima zu erhalten, wurde die Optimierung von verschiedenen Startpunkten aus wiederholt laufen gelassen. Bei 300 Optimierungsläufen war der Median der Fehlerrate auf den Lerndaten bei 2.89%. Abbildung 12.1 zeigt den Trainingslauf des besten Netzwerks. Die Topologie des besten Netzwerks zeigt Abbildung 12.2.

Die Fehlerraten, die Vertauschungsmatrix und die klassenspezifischen Rückschlußwahrscheinlichkeiten waren

- Fehlerraten: Lerndaten: 2.65%, Testdaten: 2.53%
- Vertauschungsmatrizen $\boldsymbol{V}$:

Lerndaten			
	L	E	K
L	236	8	0
E	7	1302	43
K	8	45	2547

Testdaten			
	L	E	K
L	121	1	0
E	3	648	23
K	6	20	1274

Die Zeilen- und Spaltenbezeichnungen stehen für leer (L), Erwachsener (E) und Kindersitz (K). Ein Matrixelement v_{ij} der Vertauschungsmatrix gibt die Zahl der Datenvektoren an, die zur Klasse i gehören und in die Klasse j klassifiziert wurden.

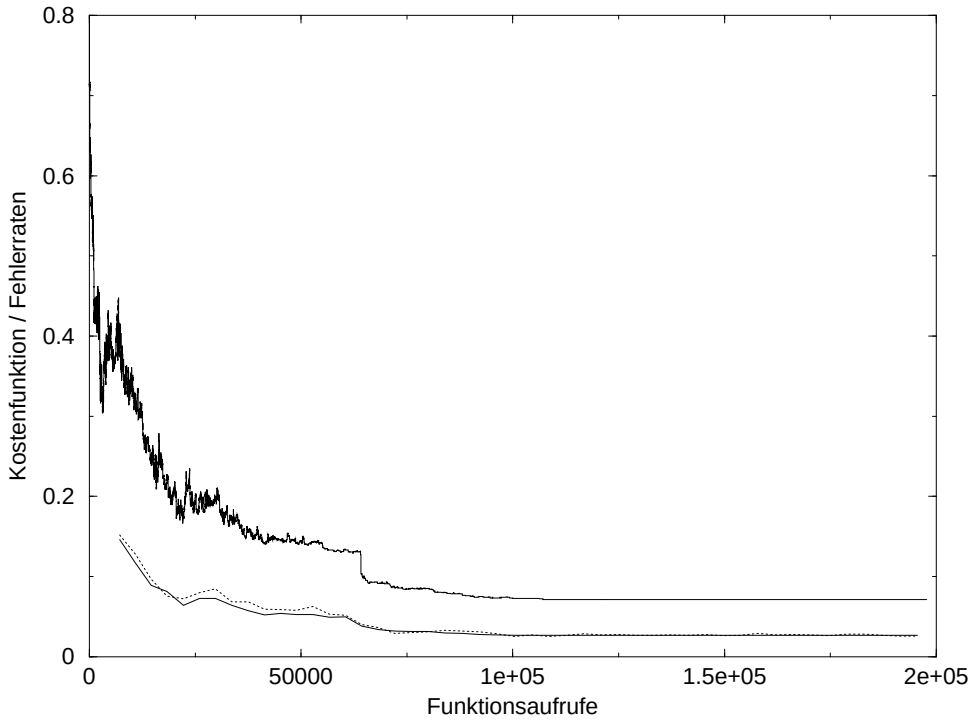


Abbildung 12.1: Kostenfunktion und Fehlerraten bei der Sitzbelegung. Die obere durchgezogene Linie ist die Kostenfunktion. Ihre Schwankungen sind am Anfang der Optimierung mit Simulated Annealing noch stark und nehmen zum Ende der Optimierung immer mehr ab. Die unteren beiden Kurven sind die Fehlerraten auf den Lerndaten (durchgezogene Linie) und auf den Testdaten (gestrichelte Linie). An diesem Beispiel kann man schön sehen, daß das RHPN lernt, von den Trainingsdaten auf die Testdaten zu generalisieren.

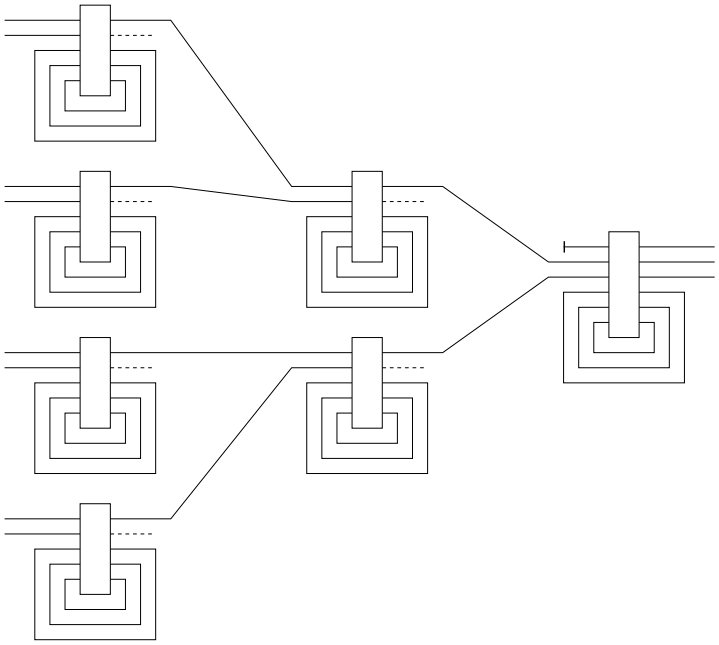


Abbildung 12.2: Topologie des besten rückgekoppelten HPNs zur parallelen Verarbeitung gleichrangiger Bits verschiedener Sensoren. Jeder Knoten hat drei Rückkopplungen. Das Netz hat drei Ausgangsbits, weshalb acht mögliche Ausgangszustände auftreten können. Die gestrichelten Ausgangsbits der Knoten werden nicht beachtet.

- Klassenspezifische Rückschlußwahrscheinlichkeiten:

	L	E	K	BW
Ausgangszustand				
0	94	2.79	3.19	5.98
1	0.365	99.6	0	6.53
2	0	4.29	95.7	12.8
3	0	0.619	99.4	42.4
4	0	3.26	96.7	6.58
5	1.47	90	8.5	8.13
6	0	99.3	0.711	10.1
7	0.629	95.3	4.09	7.58

In den Spalten L, E, und K stehen die klassenspezifischen Rückschlußwahrscheinlichkeiten und in der Spalte BW steht die relative Besuchshäufigkeit eines Zustands, jeweils in Prozent. Beispielsweise hielt das RHPN für 42.4% der Trainingsdaten im Zustand 3 an, der eine Rückschlußwahrscheinlichkeit von 99.4% auf die Klasse „Kindersitz“ hat. Wenn wir nun z. B. eine Mindestentscheidungssicherheit von 95% forderten, würden wir alle unbekannten Eingangsdaten zurückweisen, die in einen Zustand mit einer Rückschlußwahrscheinlichkeit kleiner als 95% fallen. Dann müßte man z. B. einen weiteren Sensor zur Entscheidung heranziehen.

12.2.3 Parallele Verarbeitung der Datenbits eines Sensors

Die andere Möglichkeit zur Serialisierung eines Datenvektors besteht darin, jeweils alle Bits eines Sensorstrahls gleichzeitig, dafür aber die Sensorstrahlen nacheinander zu verarbeiten. Es ist kaum vorherzusagen, welche Topologie bessere Ergebnisse bringt, weshalb beide Möglichkeiten getestet wurden.

Diese und die eben vorgestellte Topologie unterscheiden sich vor allem in der Variierbarkeit der Auflösung bzw. der Zahl der Sensoren. Verarbeitet man jeweils einen Sensor nach dem anderen, ist es problemlos möglich, die Zahl der Sensoren zu verändern, *ohne* auch die Topologie des Netzwerks ändern zu müssen. Verarbeitet man dagegen die einzelnen Bits der Sensoren nacheinander, so kann die Zahl der Bits pro Entfernungszahl ohne Änderung der Netzwerktopologie verändert werden.

Auch bei dieser Verarbeitung der Eingangsdaten wurden wieder verschiedene Trainingsläufe durchgeführt. Bei 150 Optimierungsläufen war der Median der Fehlerrate auf den Lerndaten bei 5.6%. Abbildung zeigt 12.3 den Trainingslauf des besten Netzwerks und Abbildung 12.4 die Topologie des besten Netzwerks.

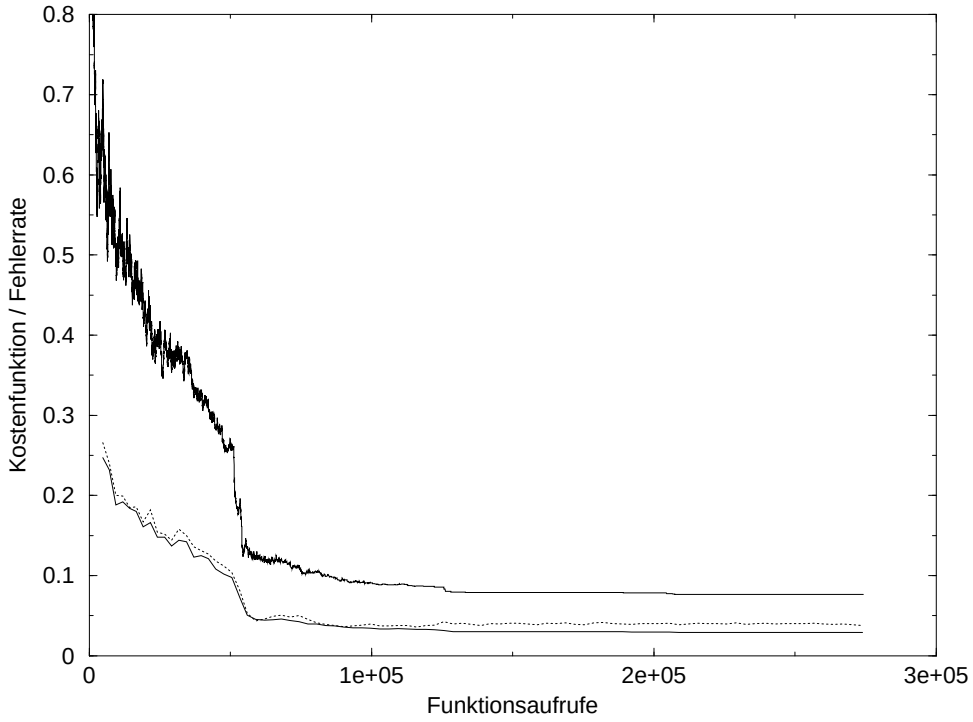


Abbildung 12.3: Kostenfunktion und Fehlerraten bei der Sitzbelegung. Die obere durchgezogene Linie ist die Kostenfunktion. Die unteren beiden Kurven sind die Fehlerraten auf den Lerndaten (durchgezogene Linie) und auf den Testdaten (gestrichelte Linie). Hier ist die Fehlerrate auf den Trainingsdaten etwas geringer als auf den Testdaten.

Die Fehlerraten und die Vertauschungsmatrix waren

- Fehlerraten: Lerndaten: 2.91%, Testdaten: 3.77%
- Vertauschungsmatrizen $\boldsymbol{V}$:

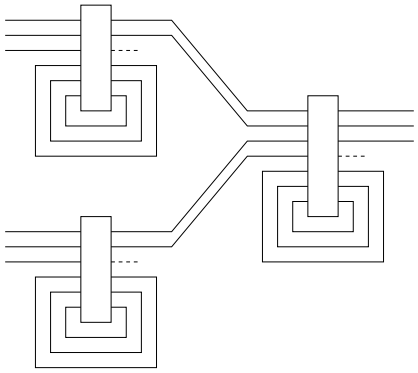


Abbildung 12.4: Bestes SRHPN zur parallelen Verarbeitung der Datenbits eines Sensors. Von der ersten zur zweiten Netzwerkschicht werden von jedem Knoten zwei Bit weitergereicht. Es können wieder acht mögliche Ausgangszustände auftreten. Die gestrichelten Ausgangsbits der Knoten werden nicht beachtet

Lerndaten			
	L	E	K
L	228	16	0
E	8	1306	38
K	5	55	2540

Testdaten			
	L	E	K
L	120	2	0
E	6	635	33
K	2	36	1262

Die Zeilen- und Spaltenbezeichnungen stehen für leer (L), Erwachsener (E) und Kindersitz (K).

- Klassenspezifische Rückschlußwahrscheinlichkeiten:

	L	E	K	BW
Ausgangszustand				
0	0	0	100	0.0477
1	0	1.37	98.6	61.4
2	0	40	60	0.001
3	0	14.3	85.7	0.0954
4	0	0	100	0.238
5	1.17	94.8	4.01	32
6	94.6	3.32	2.07	6.1
7	0	100	0	0.0954

In den Spalten L, E, und K stehen die klassenspezifischen Rückschlußwahrscheinlichkeiten und in der Spalte BW steht die relative Besuchshäufigkeit eines Zustands, jeweils in Prozent. Sortieren wir so wie in Abbildung 12.5 die Zustände nach ihren Rückschlußwahrscheinlichkeiten, können wir sehen, wieviel Prozent der Trainingsdaten mit welcher Sicherheit klassifiziert werden.

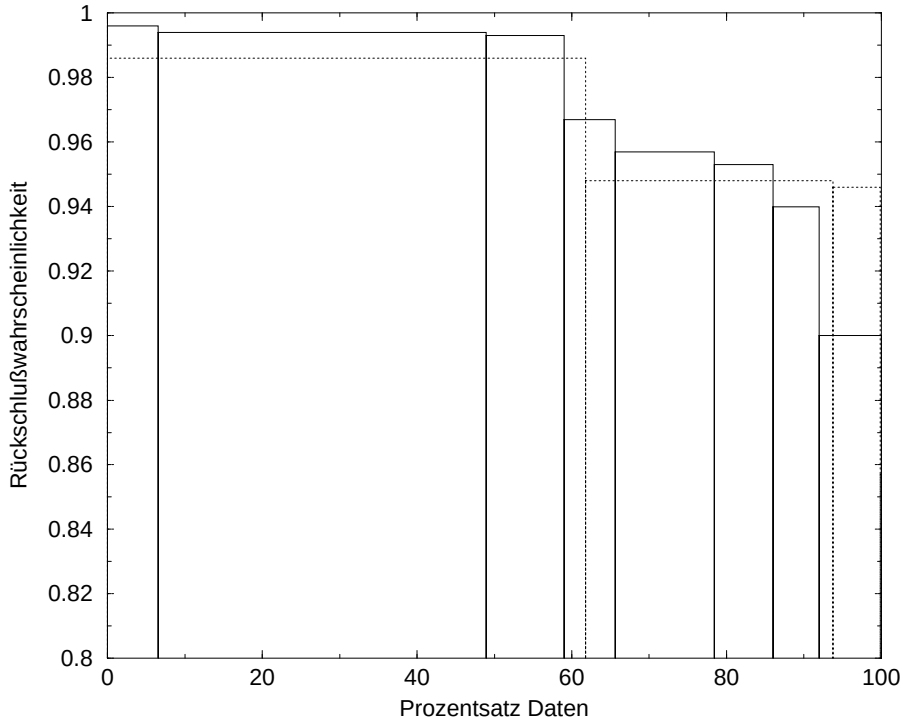


Abbildung 12.5: Rückschlußwahrscheinlichkeiten der Trainigsdaten. Die durchgezogene Linie zeigt die Rückschlußwahrscheinlichkeiten des RHPNs aus dem vorherigen Abschnitt, die gestrichelt Linie die Rückschlußwahrscheinlichkeiten des RHPNs aus diesem Abschnitt. Wir sehen, wieviel Prozent der Trainingsdaten mit welcher Sicherheit klassifiziert werden. Beispielsweise werden von dem RHPN aus dem vorherigen Abschnitt knapp 60% der Trainingsdaten mit einer Sicherheit von mindestens 99% klassifiziert. Wir können in dieser Darstellung genau sehen, auf wieviel Prozent der Trainingsdaten das in diesem Abschnitt trainierte RHPN eine geringere Rückschlußwahrscheinlichkeit hat.

12.3 Vergleich mit Supportvektor-Maschine, Polynomklassifikator und neuronalem Netz

Im Rahmen des Projektes wurden von Kollgegen auch andere Verfahren untersucht, um dieses Klassifikationsproblem zu lösen (Kienzle, 1998). Hierbei wurden die starken Hardwarebeschränkungen zunächst nicht berücksichtigt, sondern auch Verfahren getestet, die eigentlich die Hardwarebeschränkungen überschritten. Seitens der Auftraggeber wurden die Hardwarebeschränkungen folgendermaßen formuliert: Für die Klassifikation der Daten stehen nur wenige Hundert Byte Speicherplatz und ein einfacher Ganzzahlprozessor zur Verfügung. Untersucht wurden viele verschiedene Variationen der folgenden Klassifikationsansätze:

Supportvektor-Maschine: Getestet wurden Supportvektor-Maschinen mit Trennfunktionen vom Grad $G = [2, 3, 4]$, wobei eine Supportvektor-Maschine mit dem Grad 3 das beste Ergebnis lieferte.

Polynomklassifikator: Von den untersuchten Polynomklassifikatoren mit dem Grad $G = [2, 3, 4]$ war ein Klassifikator mit dem Grad 4 am besten.

Neuronales Netz: Je nach gewählter Topologie zeigten die untersuchten neuronalen Netze unterschiedliche Erkennungsraten. Die untersuchten Topologien waren u. a. : 8-5-4-3, 8-7-7-3, 8-11-11-3, 8-10-3. Das beste Netz hatte die Topologie 8-11-11-3.

Tabelle 12.1 zeigt die besten der untersuchten Ansätze im Vergleich mit dem besten RHPN aus Abschnitt 12.2.2, ihre Fehlerraten, ihren Speicherplatzverbrauch für die freien Parameter und die benötigte Trainingszeit.

Klassifikator	FL in %	FT in %	Größe in Byte	Trainingszeit (min)
Supportvektor-Maschine	0.24	1.38	1980	ca. 15
Neuronales Netz	1.14	1.86	968	15
Polynomklassifikator	1.15	2.15	5940	12
RHPN	2.65	2.53	120	30

Tabelle 12.1: Fehlerraten, Speicherplatz und Trainingszeiten für verschiedene Klassifikationsverfahren. FL ist die Fehlerrate auf den Trainingsdaten und FT die Fehlerrate auf den Testdaten. Der Speicherplatz für die oberen drei Verfahren wurde mit der Annahme von vier Byte pro Gleitkommaparameter berechnet. Es wurde also schon davon ausgegangen, daß die Verfahren sparsam mit Speicherplatz umgehen, da normalerweise Gleitkommazahlen mit acht Byte abgespeichert werden. Beim RHPN kann man den Spreicherplatz direkt aus der Zahl und Größe der Knotentabellen berechnen ($6 \cdot 2^5/2 + 2^5 \cdot 6/8 = 120$ Byte). Die Fehlerraten werden von unten nach oben immer kleiner, die Supportvektor-Maschine schneidet am besten ab. Der benötigte Speicherplatz variiert stark bei den verschiedenen Verfahren. Das rückgekoppelte Hyperpermutationsnetzwerk braucht mit 120 Byte acht mal weniger Speicher als das neuronale Netz. Außerdem benötigt es als einziges Verfahren wegen der Arithmetikfreiheit für die Klassifikation der Daten keinen Prozessor. Dafür ist seine Fehlerrate geringfügig größer.

Tabelle 12.1 zeigt, daß es von dem Einsatzgebiet abhängt, welches Klassifikationsverfahren am geeignetsten ist. Hat man „unbegrenzt“ Ressourcen zur Verfügung, z. B. eine moderne Workstation, so kann man sich z. B. für das Klassifikationsverfahren entscheiden, das die kleinste Fehlerrate hat.

Hat man es dagegen mit starken Beschränkungen der Hardware zu tun, so wie es beim Einsatz im Serienfahrzeug fast immer der Fall ist, scheiden einige Verfahren von vornherein aus

und man kann ein Verfahren wie das rückgekoppelte Hyperpermutationsnetzwerk aufgreifen, um das Klassifikationsproblem zu lösen. Neben der Klassifikation liefert einem das RHPN auch die klassenspezifischen Rückschlußwahrscheinlichkeiten eines unbekannten Datenvektors, so daß für jeden Datenvektor auch die Sicherheit der Klassifikation bekannt ist. Im Falle des RHPN „bezahlt“ man die extrem geringe Größe und die Möglichkeit, auf einen Prozessor ganz zu verzichten, mit einer etwas höheren Fehlerrate.

Um die Fehlerraten der verschiedenen Verfahren für den Einsatz im Fahrzeug weiter zu senken, sollte ein anderer, besser geeigneter Sensor eingesetzt werden.

13 Zusammenfassung

In dieser Arbeit haben wir untersucht, ob und wie man rückgekoppelte Hyperpermutationsnetzwerke trainieren kann, um damit Aufgaben der maschinellen Wahrnehmung mit möglichst wenig Ressourcen zu lösen. Die Forderung nach einer möglichst ressourcensparenden Lösung von Wahrnehmungsaufgaben kommt aus dem Bereich der Serienprodukte, in unserem Fall handelte es sich um Automobile. Bei der Produktion eines Fahrzeugs wird um jede Mark gerungen, so daß ein Ansatz zur maschinellen Wahrnehmung, der keinen Prozessor benötigt, besonders interessant ist. Ein Hyperpermutationsnetzwerk (HPN) kommt ohne Prozessor aus, weil es sich um ein Netzwerk aus Tabellen handelt, deren Inhalt beim Prozessieren der Daten nur nachgeschlagen werden und nicht arithmetisch verarbeitet werden muß.

Da wir in dieser Arbeit einen statistischen Ansatz untersucht haben, wurden als bereits bekannte Ansätze zur statistischen Mustererkennung neuronale Netze vorgestellt, da sie in der statistischen Mustererkennung sehr häufig eingesetzt werden. Besonderes Augenmerk wurde dabei auf RAM-basierte neuronale Netze gerichtet, die wie HPNs diskrete Ansätze zur statistischen Mustererkennung sind. Bei der formalen Einführung der Hyperpermutationsnetzwerke haben wir die statischen und dynamischen Eigenschaften von HPNs untersucht und unter anderem gezeigt, daß man mit Hilfe eines HPNs mit Knoten einer Dimension $d \leq 3$ jede Funktion $\mathbb{B}^n \rightarrow \mathbb{B}^n$ realisieren kann. Das Konzept des Produktraums hat es uns ermöglicht, die durch ein HPN realisierte Funktion als lineare Abbildung zu verstehen, wodurch sich das Optimierungsproblem kontinuisieren ließ. Der Vergleich mit vorhandenen mathematischen Modellen hat gezeigt, daß Hyperpermutationsnetzwerke von ihrer Architektur, nicht aber von ihren dynamischen Eigenschaften, den Reversible Computing Netzen von Toffoli (1980a) am ähnlichsten sind. Im Gegensatz gerade zu den jüngeren Ansätzen aus dem Bereich der RAM-basierten neuronalen Netze muß man bei Hyperpermutationsnetzwerken keine Kompromisse eingehen, wenn es um die Hardware-Realisierbarkeit geht, da für ein HPN ausschließlich Lookup-Tabellen benötigt werden. Da der Ausgang eines Hyperpermutationsnetzwerks mehrere binäre Leitungen umfaßt, lassen sich auf Grund der verschiedenen Ausgangszustände eines HPNs die klassenspezifischen Rückschlußwahrscheinlichkeiten der Trainingsdaten im Gegensatz zu den meisten RAM-basierten neuronalen Netzen während des Trainings ermitteln. Dies ermöglicht uns, während des „Betriebs“ des HPNs als Klassifikator nicht nur die Klassenzugehörigkeit unbekannter Daten, sondern auch die Sicherheit, mit der die Entscheidung getroffen worden ist, zu ermitteln.

Um ein statistisches Verfahren zur Mustererkennung trainieren zu können, muß man notwendigerweise eine Güte- bzw. Kostenfunktion definieren, die die Güte eines Mustererkennungssystems quantifiziert. Wir haben zu diesem Zweck die sog. Information (nicht die Shannon'sche Information) eingeführt. Sie quantifiziert, wieviele für die Aufgabe relevante Unterschiede in den Eingangsdaten nach der Datenreduktion mit einem HPN am Ausgang des Netzwerks noch verfügbar sind. Je mehr wichtige Unterschiede von dem HPN bei der Datenreduktion erhalten werden, desto größer ist die Information am Ausgang des Netzwerks und desto größer ist auch seine Güte. Dadurch läßt sich jedem Paar aus Hyperpermutationsnetzwerk und Trainingsdaten eindeutig eine Güte zuordnen, so daß auf dieser Basis das Trainings des Netzes vollzogen werden kann. Da die hier eingeführte Information kein Abstandsmaß sondern nur eine Gleich-

heitsrelation benötigt, ist sie für die Bewertung digitaler Daten besonders geeignet.

Bei der Optimierung von Hyperpermutationsnetzwerken wurden zunächst Netzwerke ohne Rückkopplungen betrachtet. Da die kombinatorische Optimierung eines HPN NP-vollständig ist, wurde schon von Oberländer (1997) der sogenannte PAP-Algorithmus entwickelt, der die kontinuisierte Variante des Optimierungsproblems löst. Dieser Algorithmus wurden mit klassischen gradientenbasierten Verfahren zur nichtlinearen Optimierung unter Gleichungs- und Ungleichungsnebenbedingungen verglichen. Der PAP-Algorithmus war um Größenordnungen schneller als z. B. die Optimierverfahren aus dem Programmpaket MINOS und terminierte außerdem nur sehr selten in lokalen Minima.

Bei rückgekoppelten Hyperpermutationsnetzwerken (RHPN) wird die Optimierung im Gegensatz zu HPNs ohne Rückkopplungen dadurch erschwert, daß der Grad der kontinuierlichen Kostenfunktion nun größer als zwei ist und daß die Trainingsdaten wiederholt vom Netz prozessiert werden müssen. Um das Optimierungsproblem von RHPNs zu lösen, wurden verschiedene Ansätze zur Optimierung unter Gleichungs- und Ungleichungsnebenbedingungen untersucht. Auf der einen Seite wurde der PAP-Algorithmus auf RHPNs erweitert. Auf der anderen Seite wurden gradientenbasierte und gradientenfreie Verfahren zur Minimierung mit Nebenbedingungen, wie z. B. Algorithmen vom Downhill-Simplex-Typ und Verfahren der Richtungswahl mit Linienminimierung, auf das spezielle Problem angepaßt und miteinander verglichen. Aus dem Bereich der stochastischen Verfahren zur kombinatorischen Optimierung wurde Simulated Annealing verwendet. Bei den numerischen Experimenten zeigte sich, daß Simulated Annealing am besten für die Optimierung geeignet ist. Es ist nicht nur gegenüber lokalen Minima der Kostenfunktion relativ robust, sondern ist auch eine Größenordnung schneller als die untersuchten Verfahren, die im Kontiuum operieren. Dieser viel geringere Rechenbedarf kommt daher, daß die bei kontinuierlichen Verfahren nötigen Matrizenmultiplikation bei einem diskreten Optimierverfahren wegfallen.

Die entwickelten Optimierverfahren wurden an verschiedenen klassischen Testproblemen aus dem Bereich der neuronalen Netze, und zwar den Boole'schen Funktionen XOR, PARITY, AND und dem T-C Problem, und an einem echten Praxisproblem getestet. Bei den Testproblemen konnte ein RHPN meistens schneller trainiert werden und kam mit weniger Speicher aus, als ein entsprechendes neuronales Netz. Durch die Rückkopplungsleitungen konnte man bei einigen Problemen mit einem RHPN eine sehr kompakte Realisierung erreichen. Außerdem wurden für die Boole'schen Funktionen und das T-C Problem RHPNs vorgestellt, die das jeweilige Problem unabhängig von der Länge der Eingangsvektoren lösen konnten.

Das Praxisbeispiel der Sitzbelegungserkennung im letzten Kapitel hat gezeigt, daß ein rückgekoppeltes Hyperpermutationsnetzwerk ein typisches praxisnahes Klassifikationsproblem mit sehr wenig Speicherverbrauch und ohne Arithmetik, also ohne auf einen Prozessor angewiesen zu sein, lösen kann. Dabei war die Klassifikationsleistung eines klassischen neuronalen Netzes mit Arithmetik und achtfach höherem Speicherverbrauch nur geringfügig besser. Damit eignen sich rückgekoppelte HPN vor allem für die Bereiche der maschinellen Wahrnehmung, in denen man mit extrem geringen Hardwareanforderungen auskommen muß. Die Trainingszeiten skalierten bei rückgekoppelten HPN wie bei neuronalen Netzen linear mit der Größe des Trainigsdatensatzes.

Eine nicht untersuchte und daher noch offene Frage ist die der Topologieoptimierung. Es wäre wünschenswert, wenn man zur Lösung einer maschinellen Wahrnehmungsaufgabe nur noch die Trainingsdaten bereitstellen müßte, und neben der optimalen Tabellenbelegung auch die optimale Topologie des HPNs vollautomatisch gefunden würde. Denkbar wäre z. B. bei einem Verfahren wie Simulated Annealing, nicht nur die Tabellen der Netzwerk-Knoten zu verändern, sondern auch die Zahl der Knoten und ihre Verknüpfungen automatisch zu variieren. Noch muß die Frage der Netzwerktopologie von einem Menschen beantwortet werden.

A Anhang

A.1 Die Netzfunktion

Mit Hilfe des Kronecker-Produkts läßt sich die Netzfunktion eines spaltenweise verbundenen HPNs ohne Rückkopplungen kompakt angeben. Ebenso wie die Leitungen kann man auch die Hyperpermutationen in den einzelnen Knoten mittels des Kronecker-Produkts zuerst spaltenweise und dann spaltenübergreifend zusammenfassen. Man erhält auf diese Weise eine Hyperpermutationsmatrix $\boldsymbol{H}^{\text{Netz}}$ für das gesamte Netz, also eine Netzfunktion, die dann auf den gesamten Netzwerkeingang $\boldsymbol{\chi}^{\text{Netz}}$ angewandt werden kann.

Dazu stellen wir uns vor, daß HPN habe J Spalten und in der j-ten Spalte I_j Knoten. Außerdem gäbe es nur Leitungen zwischen benachbarten Spalten. Dann gehen wir zur Konstruktion der Netzfunktion folgendermaßen vor:

1. Wie fassen mit dem Kronecker-Produkt analog zur Zusammenfassung von Leitungsvektoren die Hyperpermutationsmatrizen der Knoten einer Spalte j zu der spaltenübergreifenden Hyperpermutationsmatrix

$$\bigotimes_{i=0}^{I_j-1} \boldsymbol{H}_{ij}$$

   zusammen.

2. Die Leitungen zwischen Knoten zweier Spalten stellen nichts anderes als Permutationen dar, weswegen wir sie durch Permutationsmatrizen $\boldsymbol{V}_j$ repräsentieren können. Nennen wir den spaltenübergreifenden Eingangsvektor der j-ten Spalte $\boldsymbol{\chi}_j$ und den spaltenübergreifenden Ausgangsvektor $\boldsymbol{\kappa}_j$, gilt $\boldsymbol{\chi}_j = \boldsymbol{V}_j \cdot \boldsymbol{\kappa}_{j-1}$, $\boldsymbol{V}_0$ ist die Einheitsmatrix.

3. Jede Spalte j führt also auf dem gesamten Eingangsvektor $\boldsymbol{\chi}_j$ der Spalte die Matrixmultiplikation

$$\left(\bigotimes_{i=0}^{I_j-1} \boldsymbol{H}_{ij} \right) \cdot \boldsymbol{V}_j$$

   durch.

4. Nun reihen wir die einzelnen Spalten mittels normaler Matrixmultiplikation aneinander, und erhalten mit dem Netzeingangsvektor $\boldsymbol{\chi}^{\text{Netz}} = \boldsymbol{\chi}_0$ und dem Netzausgangsvektor

$\boldsymbol{\kappa}^{\text{Netz}} = \boldsymbol{\kappa}_{J-1}$:

$$\begin{aligned}\boldsymbol{\kappa}_{J-1} &= \prod_{j=J-1}^{0} \left[\left(\bigotimes_{i=0}^{I_j-1} \boldsymbol{H}_{ij} \right) \cdot \boldsymbol{V}_j \right] \cdot \boldsymbol{\chi}_0 \\ &= \boldsymbol{H}^{\text{Netz}} \cdot \boldsymbol{\chi}_0 \\ \rightarrow \boldsymbol{H}^{\text{Netz}} &= \prod_{j=J-1}^{0} \left[\left(\bigotimes_{i=0}^{I_j-1} \boldsymbol{H}_{ij} \right) \cdot \boldsymbol{V}_j \right]\end{aligned}$$

Wir können also die Netzfunktion im Produktraum formal als eine einzige Matrix ausdrücken. Da bei einer Netzwerkdimension d diese Matrix eine $2^d \times 2^d$-Matrix ist, nimmt sie schon für Netzwerke mit einigen Zehn Eingangsbits eine astronomische Größe an und ist daher nicht von praktischer Relevanz. Aus diesem Grund „verteilt“ man gerade die Funktion eines Netzwerks auf mehrere Knoten.

A.2 Diversität und Entropie

In diesem Abschnitt untersuchen wir die Beziehung zwischen Entropie und Diversität. Wir zeigen, daß Entropie und heiße relative Diversität ihr Maximum bei der gleichen Wahrscheinlichkeitsverteilung, der Gleichverteilung, erreichen. Wir untersuchen das Verhalten der beiden Funktionen am Maximum und zeigen außerdem, daß die heiße relative Diversität weder obere noch untere Schranke der Entropie ist. Als Beispiele dienen eine binäre und eine ternäre stochastische Quelle.

A.2.1 Grundbegriffe der Shannon'schen Informationstheorie

In Abschnitt 5.2.1 auf Seite 67 haben wir bereits statistische Grundbegriffe definiert. Nu führen wir die *Shannon'sche Information* und die *Entropie* ein. Wir betrachten dazu eine stochastische Quelle $Q = (\mathcal{Q}, \boldsymbol{p})$, die Z verschiedene Symbole $z \in \mathcal{Q}$ emittieren kann. Eine solche Quelle nennt man auch Z-äre Quelle. Ein Symbol z werde mit der Wahrscheinlichkeit p_z mit $\sum_z p_z = 1$ emittiert[1]. Die Häufigkeiten der einzelnen Symbole sei n_z und die Gesamtzahl der emittierten Symbole $N = \sum_z n_z$. Die erwartungstreuen Schätzungen für die Symbolwahrscheinlichkeiten sind $p_z = n_z/N$. Kommen wir nun zu

Definition 25. Shannon'sche Information: Die Shannon'sche Information I^{S} eines emittierten Symbols ist als

$$I^{\mathrm{S}} := \log_a\left(\frac{1}{p_z}\right) = -\log_a(p_z) \tag{A.1}$$

definiert, wobei $a > 1$ eine reelle Konstante ist. Die Wahl von a legt die Einheit von I^{S} fest. Für $a = 2$ ist die Einheit 'bit', für $a = e$ ist die Einheit 'nat' und für $a = 10$ ist die Einheit 'Hartley'.

Entropie: Die Entropie H ist als der Erwartungswert der Shannon'schen Information definiert:

$$H_a = \left\langle I^{\mathrm{S}} \right\rangle = \langle -\log_a(p_z) \rangle = -\sum_z p_z \log_a(p_z) \; .$$

A.2.2 Das Maximum von Entropie und heißer relativer Diversität

Die Entropie einer Z-ären Quelle Q erreicht ihr Maximum bei der Gleichverteilung $p_z = 1/Z \;\forall z$ (Beweis siehe Heise und Quattrocchi, 1995, Seite 116) und das Maximum ist

$$H_a^{\max}(p_z = 1/Z)) = -\sum_z p_z \log_a(p_z) = -Z\,\frac{1}{Z}\,\log_a\left(\frac{1}{Z}\right) = -\log_a\left(\frac{1}{Z}\right) = \log_a(Z) \; .$$

Für die heiße relative Diversität gilt der

Satz 11. *Die heiße relative Diversität d_{h} hat ihr Maximum bei der Gleichverteilung $p_z = 1/Z \;\forall z$. Der Wert des Maximums ist*

$$d_{\mathrm{h}}(p_z = 1/Z) = 1 - \frac{1}{Z} \; .$$

[1] Wenn nicht anders angegeben, geht die Summe über z immer von 0 bis $N-1$: $\sum_z \equiv \sum_{z=0}^{N-1}$.

Beweis. Wir schreiben die Wahrscheinlichkeiten p_z als $p_z = 1/Z + \alpha_z$, wobei wegen $\sum_z p_z = \sum_z (1/Z + \alpha_z) = 1$ nun $\sum_z \alpha_z = 0$ gelten muß. Wir erhalten

$$\begin{aligned} d_{\mathrm{h}} &= 1 - \sum_z p_z^2 = 1 - \sum_z \left(\frac{1}{Z} + \alpha_z\right)^2 \\ &= 1 - \sum_z \frac{1}{Z^2} - \frac{2}{Z} \underbrace{\sum_z \alpha_z}_{=0} - \sum_z \alpha_z^2 \\ &= 1 - \frac{1}{Z} - \sum_z \alpha_z^2 \end{aligned}$$

Die heiße relative Diversität ist genau dann am größten, wenn der Term $\sum_z \alpha_z^2$ am kleinsten ist. Der kleinste Wert, den der Term prinzipiell annehmen kann, ist $\sum_z \alpha_z^2 = 0$, da es sich um eine Summe von Quadraten handelt. Also muß $\alpha_z = 0 \ \forall z$ gelten, was auch die Nebenbedingung $\sum_z \alpha_z = 0$ erfüllt. Der Wert des Maximums ist also

$$d_{\mathrm{h}}^{\max}(p_z = 1/Z) = 1 - \frac{1}{Z}$$

□

Da wir bei der Entropie Skalierungsspielraum haben, verlangen wir nun, daß die maximale Entropie gleich der maximalen heißen relativen Diversität ist:

$$\begin{aligned} H_a^{\max} &\stackrel{!}{=} d_{\mathrm{h}}^{\max} \\ \log_a(Z) &= 1 - \frac{1}{Z} \\ Z &= a^{\frac{Z-1}{Z}} \\ a &= Z^{\frac{Z}{Z-1}} \end{aligned}$$

Wählen wir also $a = Z^{Z/(Z-1)}$, ist $H_a^{\max} = d_{\mathrm{h}}^{\max}$.

A.2.3 Die heiße relative Diversität: Eine Schranke der Entropie?

Wie wir gesehen haben, können wir mit der richtigen Wahl der Basis a die Entropie $H_a^{\max}$ der relativen heißen Diversität $d_{\mathrm{h}}^{\max}$ gleichsetzen. Wie ist aber das Verhältnis der Entropie zur relativen heißen Diversität bei anderen Wahrscheinlichkeitsverteilungen p_z? Um dies zu beantworten, untersuchen wir, wie sich in der Umgebung des Maximums die heiße relative Diversität der Entropie nähert. Wir beantworten diese Frage mit

Satz 12. *Sei Q eine Z-äre Quelle mit ganzzahligem $Z \in [2, \infty)$ und sei $a = Z^{Z/(Z-1)}$. In einer lokalen Umgebung des Maximums von heißer relativer Diversität und Entropie gilt*

1. *Für $Z \in \{2, 3\} : d_{\mathrm{h}} \leq H_a$.*
2. *Für $Z > 3 : d_{\mathrm{h}} \geq H_a$.*

Beweis. Zunächst berechnen wir die Hesse-Matrix der relativen heißen Diversität. Die Lagrange-Funktion der heißen relativen Diversität ist

$$L(p_z, \lambda) = d_{\mathrm{h}} + \lambda \cdot \left(\sum_z p_z - 1\right)$$

Die Gradientenkomponenten sind

$$\frac{\partial}{\partial p_k} L = \frac{\partial}{\partial p_k} \left[1 - \sum_z p_z^2 + \lambda \cdot \left(\sum_z p_z - 1 \right) \right] = -2p_k + \lambda \stackrel{!}{=} 0 \tag{A.2}$$

Daraus folgt $p_k = \lambda/2$. Mit der Nebenbedingung $\sum_z p_z = 1$ erhalten wir für λ:

$$\begin{aligned} \sum_z p_z &= 1 \\ Z\,\lambda/2 &= 1 \\ \lambda &= \frac{2}{Z} \end{aligned}$$

Eingesetzt in (A.2) ergibt sich

$$\begin{aligned} -2p_k + \frac{2}{Z} &= 0 \\ p_k &= \frac{1}{Z}\,. \end{aligned}$$

Der Gradient verschwindet also im Punkt $p_z = 1/Z$. Die Hesse-Matrix ist

$$\frac{\partial^2}{\partial p_k\, \partial p_l} L = \frac{\partial}{\partial p_l}(-2p_k + \lambda) = -2\delta_{kl}\,.$$

Die Hesse-Matrix ist diagonal, und alle Eigenwerte der Hesse-Matrix haben den Wert -2.

Nun untersuchen wir die Hesse-Matrix der Entropie. Dazu bilden wir mit Hilfe der Lagrange-Funktion zunächst den Gradienten von H unter der Nebenbedingung $\sum_z p_z = 1$:

$$\begin{aligned} \frac{\partial L(p_z,\lambda)}{\partial p_k} H &= \frac{\partial}{\partial p_k} \left[-\sum_z p_z \log_a(p_z) + \lambda \cdot \left(\sum_z p_z - 1 \right) \right] \\ &= -\log_a(p_k) - \frac{1}{\ln(a)} + \lambda \stackrel{!}{=} 0 \\ \rightarrow \quad \log_a(p_k) &= \lambda - \frac{1}{\ln(a)} \\ p_k &= a^{\lambda - \frac{1}{\ln(a)}} \end{aligned}$$

Setzen wir nun p_k in die Nebenbedingung $\sum_z p_z = 1$ ein, können wir die p_z eliminieren und λ berechnen. Wichtig ist hier nur, daß λ also *nicht* von den p_z abhängt. Die Hesse-Matrix ist eine Diagonalmatrix:

$$\frac{\partial^2 L(p_z,\lambda)}{\partial p_k \partial p_l} H = \frac{\partial}{\partial p_l} \left[-\log_a(p_k) - \frac{1}{\ln(a)} + \lambda \right] = -\frac{\delta_{kl}}{\ln(a)\, p_l}$$

Im Punkt $p_z = 1/Z\ \forall z$ ist der Wert auf der Hauptdiagonalen $f(Z) = -Z/\ln(Z^{Z/(Z-1)}) = (1-Z)/\ln(Z)$. Die Ableitung von $f(Z)$ ist

$$f'(Z) = \frac{Z-1}{Z\,\ln(Z)^2} - \frac{1}{\ln(Z)} = \frac{Z(1-\ln(Z)) - 1}{Z\,\ln(Z)^2}$$

Nun ist $Z(1-\ln(Z))-1 < 0$ für $Z \in (0,\infty)\backslash 1$ und damit ist $f'(Z) < 0$ im selben Intervall. Daher ist $f(Z)$ streng monoton fallend. Da die Funktionswerte $f(3) = -1.82048$ und $f(4) = -2.16404$

sind, schneidet $f(Z)$ zwischen $Z = 3$ und $Z = 4$ die Gerade $y = -2$. Nun ist die Hesse-Matrix der heißen relativen Diversität genau wie die Hesse-Matrix der Entropie eine Diagonalmatrix, aber mit dem Wert -2 auf der Hauptdiagonalen. Eine diagonale Hesse-Matrix mit denselben Hauptdiagonalelementen bedeutet, daß die Krümmung im Maximum in jeder Raumrichtung gleich ist. Also können wir für eine ϵ-Umgebung des Maximums schließen:

1. $Z \in \{2, 3\} \rightarrow |f(Z)| < |-2| \rightarrow H$ hat schwächere Krümmung als $d_\mathrm{h} \rightarrow d_\mathrm{h} \leq H_a$.
2. $Z > 3 \rightarrow |f(Z)| > |-2| \rightarrow H$ hat stärkere Krümmung als $d_\mathrm{h} \rightarrow d_\mathrm{h} \geq H_a$.

□

Nun kommen wir zur Frage, ob die heiße relative Diversität obere oder untere Schranke der Entropie ist:

Satz 13. *Sei Q eine Z-äre Quelle mit ganzzahligem $Z \in [2, \infty)$, reellen $p_z \in [0, 1]$ und sei $a = Z^{Z/(Z-1)}$.*
Wenn ein $p_k = 1$ oder $p_z = 1/Z \ \forall z$ (Gleichverteilung) ist, dann gilt

$$H_a = d_\mathrm{h} \ .$$

Für alle anderen Verteilungen p_z von Q gilt

1. *Für $Z = 2$ ist d_h untere Schranke von H.*
2. *Für $Z > 2$ ist d_h weder obere noch untere Schranke von H.*

Beweis. **Falls ein $\mathbf{p_k = 1}$ ist**, gilt (siehe Abschnitt A.2.2):

$$\begin{aligned} H_a &= -\sum_z p_z \log_a(p_z) = -\log_a(e) \sum_z p_z \ln(p_z) \\ &= -\log_a(e) \left(\sum_{z \neq k} \underbrace{(0 \cdot \ln(0))}_{\lim_{x \to 0} x \ln(x) = 0} - 1 \cdot \ln(1) \right) = 0 \end{aligned}$$

$$d_\mathrm{h} = 1 - \sum_z p_z^2 = 1 - \sum_{z \neq k} 0^2 - 1^2 = 0$$

Im Fall der Gleichverteilung, d. h. $p_z = 1/Z \ \forall z$, gilt:

$$\begin{aligned} H_a = H_a^{\max} &= \log_a(Z) = \log_{Z^{Z/(Z-1)}}(Z) = \frac{Z-1}{Z} = 1 - \frac{1}{Z} \\ d_\mathrm{h} = d_\mathrm{h}^{\max} &= 1 - \frac{1}{Z} \end{aligned}$$

Für alle anderen Verteilungen gilt:
Falls $\mathbf{Z = 2}$ und damit $a = Z^{Z/(Z-1)} = 2^{2/(2-1)} = 4$, $p_0 = p$ und $p_1 = 1 - p$:

$$\begin{aligned} \Delta :=& H_a - d_\mathrm{h} = -(p \ln_4(p) + (1-p) \ln_4(1-p)) - \left(1 - p^2 - (1-p)\right)^2 \\ =& \frac{(p-1)\ (p \ \ln(16) + \ln(1-p)) - p \ \ln(p)}{\ln(4)} \end{aligned}$$

$$\Delta' = \frac{-1 + p\,\ln(16) + (p-1)\,\left(-\frac{1}{1-p} + \ln(16)\right) + \ln(1-p) - \ln(p)}{\ln(4)}$$
$$= \frac{(2\,p-1)\,\ln(16) + \ln(1-p) - \ln(p)}{\ln(4)}$$

$$\Delta'' = \frac{-\frac{1}{1-p} - \frac{1}{p} + 2\,\ln(16)}{\ln(4)} = 4 + \frac{1}{-\left(p\,\ln(4)\right) + p^2\,\ln(4)}$$

Δ hat an den drei Nullstellen $p = 0$, $p = 1/2$, $p = 1$ die Ableitungen $\Delta'(0) = \infty$, $\Delta'(1/2) = 0$, $\Delta'(1) = -\infty$. Die Krümmung bei $p = 1/2$ ist $\Delta''(1/2) = 4 - 4/\ln(4) > 0$. $\Delta(1/2) = 0$ ist also ein striktes lokales Minimum. Daher ist $\Delta(p) > 0$ für $p \in (0,1)\backslash\{1/2\}$ und $d_\mathrm{h} < H$ (siehe Abschnitt A.2.5.1 für ein Bild von Δ für $Z = 2$).

Falls $\mathbf{Z > 2}$: Wir untersuchen den Gradienten der beiden Größen, wenn ein $p_l = 1$ und alle anderen $p_z = 0$ sind. Der Nebenbedingung $\sum_z p_z = 1$ werden wir dadurch gerecht, daß wir $p_{Z-1} = 1 - \sum_{z=0}^{Z-2} p_z$ setzen.

$$\left.\frac{\partial d_\mathrm{h}}{\partial p_k}\right|_{p_l=1} = \frac{\partial}{\partial p_k}\left[1 - \sum_{z=0}^{Z-2} p_z^2 - \left(1 - \sum_{z=0}^{Z-2} p_z\right)^2\right] = -2\,p_k + 2\left(1 - \sum_{z=0}^{Z-2} p_z\right)$$
$$= \begin{cases} -2\cdot 1 + 2\,(1-1) = -2 & \text{falls } k = l \\ -2\cdot 0 + 2\,(1-1) = 0 & \text{falls } k \neq l \end{cases}$$

$$\left.\frac{\partial H}{\partial p_k}\right|_{p_l=1} = \frac{\partial}{\partial p_k}\left[-\sum_{z=0}^{Z-2} p_z \log_a(p_z) - \left(1 - \sum_{z=0}^{Z-2} p_z\right)\log_a\left(1 - \sum_{z=0}^{Z-2} p_z\right)\right]$$
$$= -\log_a(p_k) - \frac{1}{\ln(a)} + \log_a\left(1 - \sum_{z=0}^{Z-2} p_z\right) + \frac{1}{\ln(a)}$$
$$= \log_a\left(1 - \sum_{z=0}^{Z-2} p_z\right) - \log_a(p_k)$$
$$= \begin{cases} \log_a(1-1) - \log_a(1) = -\infty & \text{falls } k = l \\ \log_a(1-1) - \log_a(0) = 0 & \text{falls } k \neq l \end{cases}$$

In den Ecken $p_l = 1$ des Kompaktums $p_z \in [0,1]^{Z-1}$ mit $\sum_z p_z = 1$ und $p_z \geq 0\ \forall z$ gilt also

$$0 > \frac{\partial d_\mathrm{h}}{\partial p_l} > \frac{\partial H}{\partial p_l} \quad \rightarrow \quad \left|\frac{\partial H}{\partial p_l}\right| > \left|\frac{\partial d_\mathrm{h}}{\partial p_l}\right| > 0\,.$$

In der lokalen Umgebung einer Ecke $(0,\ldots,1,\ldots,0)^\mathrm{t} + \boldsymbol{\alpha}$ mit $|\boldsymbol{\alpha}| \ll 1$ muß wegen der Nebenbedingungen $\sum_z p_z = 1$ aber $p_l < 1$ und damit $\alpha_l < 0$ gelten. Wegen

$$\boldsymbol{\nabla} H|_{p_l=1}\cdot\boldsymbol{\alpha} = -\infty \cdot \underbrace{\alpha_l}_{<0} = +\infty \quad > \quad \boldsymbol{\nabla} d_\mathrm{h}|_{p_l=1}\cdot\boldsymbol{\alpha} = -2\cdot\underbrace{\alpha_l}_{<0}$$

gilt in der lokalen Umgebung einer Ecke $H > d_\mathrm{h}$. In einer Umgebung des Maximums der beiden Größen ist aber nach Satz 12 $d_\mathrm{h} \geq H$ für $Z > 3$. Daraus folgt, daß sich die Hyperflächen von H und d_h zwischen den zulässigen Ecken des Kompaktums $p_z \in [0,1]^{Z-1}$ und dem Maximum

bei $p_z = 1/Z$ mindestens einmal schneiden müssen. Daher kann die heiße relative Diversität für $Z > 3$ weder obere noch untere Schranke der Entropie sein.

Im Fall Z = 3 genügt es, einen Punkt zu finden, an dem $H < d_\mathrm{h}$ ist, weil in einer lokalen Umgebung des Maximum und der Ecken des Kompaktums $H \geq d_\mathrm{h}$ gilt. Hat man einen Punkt mit $H < d_\mathrm{h}$ gefunden, müssen sich die beiden Hyperflächen schneiden, und die heiße relative Diversität kann weder obere noch untere Schranke der Entropie sein. Ein Punkt mit $H < d_\mathrm{h}$ ist z. B. $p_0 = p_1 = 0.45$, $p_2 = 1 - 2 \cdot 0.45 = 0.1$:

$$H = -0.45 \log_{3^{3/2}}(0.45) - 0.45 \log_{3^{3/2}}(0.45) - 0.1 \log_{3^{3/2}}(0.1) = 0.575827$$
$$d_\mathrm{h} = 1 - 0.45^2 - 0.45^2 - 0.1^2 = 0.585$$

□

A.2.4 'Abstand' der Entropie von der heißen relativen Diversität

Wir interessieren uns für den 'Abstand' der Entropie von der heißen relativen Diversität. Vor allem möchten wir den maximalen Abstand der beiden Größen kennen. Dazu bilden wir mit Hilfe der Lagrange-Funktion den Gradienten der Differenzfunktion $\Delta = H_a - d_\mathrm{h}$ unter der Nebenbedingung $\sum_z p_z = 1$.

$$\begin{aligned}
\frac{\partial L(p_z, \lambda)}{\partial p_k} &= \frac{\partial}{\partial p_k}\left[\Delta + \lambda \cdot \left(\sum_z p_z - 1\right)\right] \\
&= \frac{\partial}{\partial p_k}\left[-\sum_z p_z \log_a(p_z) - \left(1 - \sum_z p_z^2\right) + \lambda \cdot \left(\sum_z p_z - 1\right)\right] \\
&= -\log_a(p_k) - \frac{1}{\ln(a)} + 2p_k + \lambda \stackrel{!}{=} 0
\end{aligned} \tag{A.3}$$

Mit der Nebenbedingung $\sum_z 2p_z = 2$ erhalten wir:

$$\sum_z 2p_z = \sum_z \left(\log_a(p_z) + \frac{1}{\ln(a)} - \lambda\right) = \sum_z \log_a(p_z) + \frac{Z}{\ln(a)} - Z\lambda = 2$$

$$\rightarrow \quad \lambda = \frac{\sum_z \log_a(p_z) - 2}{Z} + \frac{1}{\ln(a)}$$

Eingesetzt in (A.3) ergibt sich:

$$\begin{aligned}
\frac{\partial L(p_z, \lambda)}{\partial p_k} &= -\log_a(p_k) - \frac{1}{\ln(a)} + 2p_k + \frac{\sum_z \log_a(p_z) - 2}{Z} + \frac{1}{\ln(a)} \\
&= 2p_k + \frac{\sum_z \log_a(p_z) - 2}{Z} - \log_a(p_k) \stackrel{!}{=} 0
\end{aligned} \tag{A.4}$$

Um ein Extremum zu finden, müssen alle Ableitungen verschwinden. Wie erwartet, verschwindet der Gradient im Punkt $p_z = 1/Z \ \forall z$:

$$\begin{aligned}
\left.\frac{\partial L(p_z, \lambda)}{\partial p_k}\right|_{p_z = 1/Z} &= 2\frac{1}{Z} + \frac{\sum_z \log_{Z^{(Z/(Z-1))}}(\frac{1}{Z}) - 2}{Z} - \log_{Z^{(Z/(Z-1))}}\left(\frac{1}{Z}\right) \\
&= \frac{2}{Z} + \frac{Z\frac{1-Z}{Z} - 2}{Z} - \frac{1-Z}{Z} = \frac{1-Z}{Z} - \frac{1-Z}{Z} = 0
\end{aligned}$$

Die anderen Lösungen der Z Gleichungen (A.4) (und damit mögliche Maxima) sind analytisch nicht trivial zu finden. Sie beinhalten wahrscheinlich die Produkt-Logarithmus-Funktion $W(z)$, die die Lösung für w in $z = we^w$ ist. Eine andere Möglichkeit, den Abstand der beiden Größen zu erfassen, ist, die mittlere quadratische Abweichung

$$\int_0^1 \cdots \int_0^1 \Delta^2 \, \mathrm{d}^Z p$$

der beiden Funktionen zu berechnen. Diese Rechnung ist für beliebige Z analytisch schwierig, für $Z = 2$ ist sie im nächsten Abschnitt durchgeführt.

A.2.5 Stochastische Quellen als Beispiele

A.2.5.1 Binäre Quelle

Gegeben sei eine binäre stochastische Quelle, die zwei Symbole bernoulliverteilt mit den Wahrscheinlichkeiten $p_0 = p$ und $p_1 = 1 - p$ emittiert. Damit ist $a = Z^{Z/(Z-1)} = 2^{2/(2-1)} = 4$. Die Entropie und die heiße relative Diversität sind (siehe Abbildung A.1):

$$\begin{aligned} H_4 &= -\sum_z p_z \log_4(p_z) = -(p_0 \log_4(p_0) + p_1 \log_4(p_1)) \\ &= -(p \log_4(p) + (1-p) \log_4(1-p)) \end{aligned}$$

$$\begin{aligned} d_\mathrm{h} &= 1 - \sum_z p_z^2 = 1 - p_0^2 - p_1^2 \\ &= 1 - p^2 - (1-p)^2 \end{aligned}$$

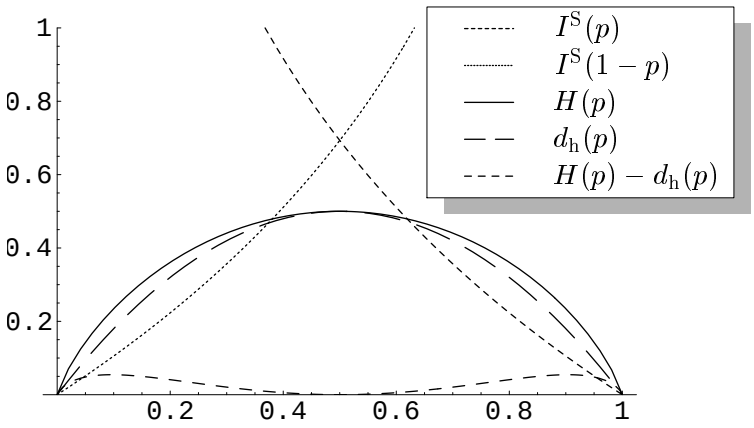


Abbildung A.1: Die Shannon'sche Information I^S für p und $1-p$, Entropy H, heiße relative Diversität d_h und die Differenz $\Delta = H - d_\mathrm{h}$. Wir sehen, daß die heiße relative Diversität die Entropie nach unten beschränkt.

Die Fläche zwischen den beiden Kurven ist

$$\begin{aligned} F &= \int_0^1 \Delta \, \mathrm{d}p = \int_0^1 -(p \log_4(p) + (1-p)\log_4(1-p)) - (1 - p^2 - (1-p)^2) \, \mathrm{d}p \\ &= \left[\frac{2p^3}{3} - \frac{\ln(p)p^2}{2\ln(4)} - p^2 + \frac{(p-2)\ln(1-p)p}{2\ln(4)} + \frac{p}{2\ln(4)} + \frac{\ln(p-1)}{2\ln(4)} \right]_0^1 \\ &= \frac{1}{\ln(16)} - \frac{1}{3} = 0.0273404 \end{aligned}$$

Teilen wir die Fläche F durch die Integrationsstrecke, erhalten wir die mittlere 'Höhe' der Fläche zwischen den beiden Größen. Hier ist also die heiße relative Diversität im Mittel $F/1 = 0.0273404$ kleiner als die Entropie.

A.2.5.2 Ternäre Quelle

Um sich ein besseres Bild vom unterschiedlichen Verhalten der Entropie und der heißen relativen Diversität machen zu können, sind hier Entropie und heiße relative Diversität für eine ternäre Quelle mit den Wahrscheinlichkeiten p_0, p_1 und p_2 in Abbildung A.2 und A.3 dargestellt.

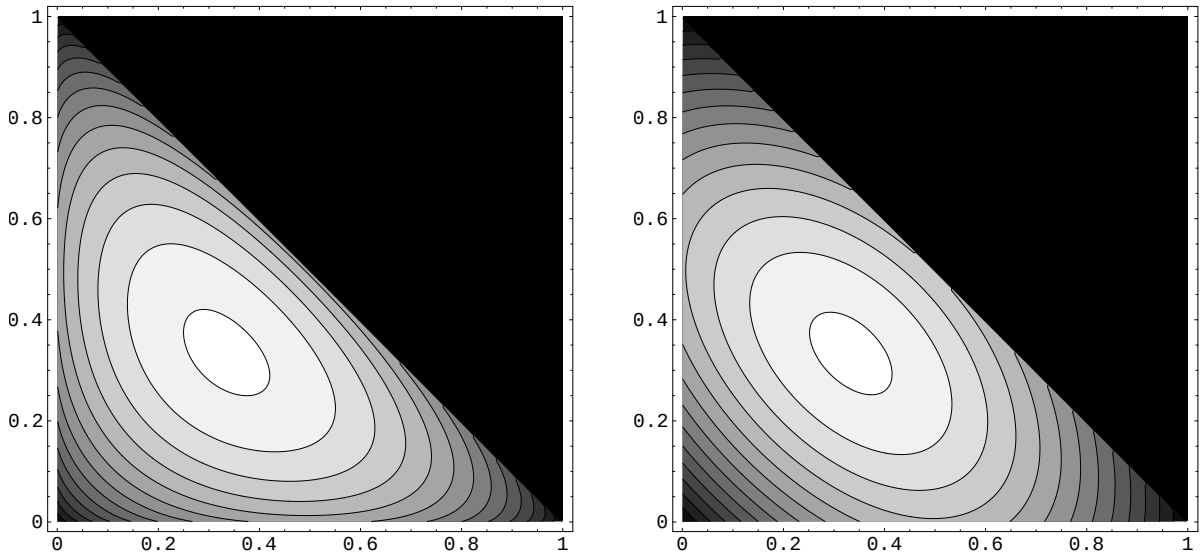


Abbildung A.2: Links die Konturgraphik der Entropie, rechts die Konturgraphik der heißen relativen Diversität. Die beiden Achsen entsprechen den Wahrscheinlichkeiten p_0 und p_1. Die Funktionen sind nur im Bereich $p_0 + p_1 \leq 1$ definiert. Die Werte der Funktionen steigen zu helleren Grauwerten an. Man sieht, daß die beiden Größen leicht verschiedene Formen haben.

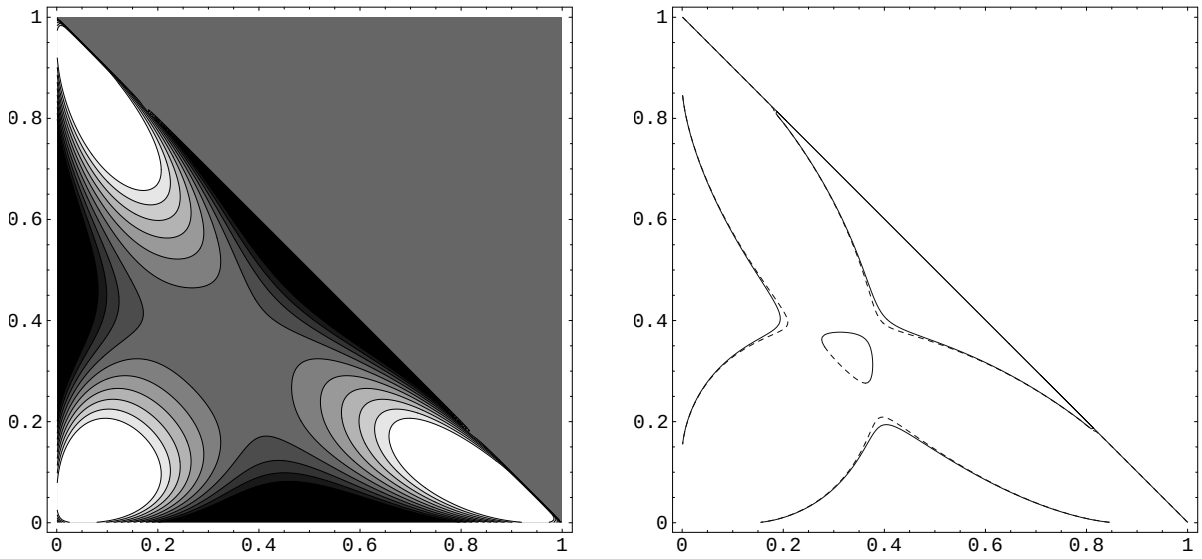


Abbildung A.3: Links die Konturgraphik der Differenz $\Delta = H - d_{\mathrm{h}}$, rechts die Linien $\Delta = 0$ (durchgezogen) und $\Delta = 0.0003$ (gestrichelt). Δ hat bei $p_z = 1/3\ \forall z$ ein lokales Minimum. Die Gerade $p_1 = 1 - p_0$ im rechten Bild trennt die beiden Bereiche $p_0 + p_1 \leq 1$ und $p_0 + p_1 > 1$. Wir wie anhand der Nullstellenlinie von Δ sehen, durchdringen sich H undd_{h}.

A.3 Die Nettoinformation bei gleichen Rückschlußwahrscheinlichkeiten pro Zustand

Gegeben sei ein Problem mit K Klassen, ein Knoten mit Z Zuständen auf seinen rückgekoppelten Leitungen und gleichverteilter Statistik, d. h. die relativen Häufigkeiten der Klassen pro Zustand sind in allen Zuständen gleich. Im folgenden laufen die Summen über i immer von 0 bis $Z - 1$, und die Summen über j immer von 0 bis $K - 1$. Mit den Bezeichnungen

$$\begin{aligned}
\text{Statistikmatrixelement}: &\quad s_{ij} \\
\text{Zeilensumme}: &\quad z_i = \sum_j s_{ij} \\
\text{relative Häufigkeit in der Spalte } j: &\quad r_j = \frac{s_{ij}}{z_i} \\
\text{Zahl der Beobachtungen}: &\quad N = \sum_i z_i
\end{aligned}$$

sehen wir aus folgender Rechnung, daß die heiße Nettoinformation null ist:

$$
\begin{aligned}
I_{\mathrm{nh}} &= D_{\mathrm{h}} - \frac{D_{\mathrm{t}}}{R_{\mathrm{t}}} R_{\mathrm{h}} = D_{\mathrm{t}} - D_{\mathrm{k}} - \frac{D_{\mathrm{t}}}{R_{\mathrm{t}}} \left(R_{\mathrm{t}} - R_{\mathrm{k}}\right) \\
&= \frac{D_{\mathrm{t}}}{R_{\mathrm{t}}} R_{\mathrm{k}} - D_{\mathrm{k}} = \frac{\left(\sum_i z_i\right)^2}{\sum_j \left(\sum_i z_i r_j\right)^2} \cdot \left(\sum_i \sum_j (z_i r_j)^2 \right) - \sum_i z_i^2 \\
&= \frac{\left(\sum_i z_i\right)^2}{\sum_j \left[r_j^2 \left(\sum_i z_i\right)^2 \right]} \cdot \left(\sum_i \left[z_i^2 \sum_j r_j^2 \right] \right) - \sum_i z_i^2 \\
&= \frac{1}{\sum_j r_j^2} \cdot \left(\sum_i \left[z_i^2 \sum_j r_j^2 \right] \right) - \sum_i z_i^2 \\
&= \sum_i z_i^2 - \sum_i z_i^2 = 0
\end{aligned}
$$

A.4 Analytische kontinuierliche Optimierung

A.4.1 Minimierung ohne Nebenbedingungen

Wir beginnen mit der analytischen Minimierung einer Funktion einer Variablen und „steigern“ uns dann zur Minimierung einer Funktion mehrerer Variablen unter Gleichungs- und Ungleichungsnebenbedingungen (für weitergehende Literatur und die Herleitungen der Bedingungen für ein lokales Minimum siehe Papageorgiou, 1991). Da sich jedes Maximierungsproblem mühelos in ein Minimierungsproblem umwandelt läßt, ist die Beschränkung auf Minimierungsprobleme keine Einschränkung der Allgemeinheit. Am Ende dieses Abschnitts werden wir ein einfaches Beispiel besprechen.

Zunächst besteht die Aufgabe darin, eine Funktion $f(x)$ zu minimieren, wofür wir kurz

$$\min_{x\in\mathbb{R}} f(x) \tag{A.5}$$

schreiben. Wir bezeichnen einen Punkt, der $f(x)$ lokal minimiert, mit $\hat{x}$. Die Bedingungen für ein lokales Minimum sind (wie aus der Schulmathematik bekannt):

$$\text{Notwendige Bedingung 1. Ordnung} : f'(\hat{x}) = 0 \tag{A.6}$$

$$\text{Notwendige Bedingung 2. Ordnung} : f'(\hat{x}) = 0 \text{ und } f''(\hat{x}) \geq 0 \tag{A.7}$$

$$\text{Hinreichende Bedingung} : f'(\hat{x}) = 0 \text{ und } f''(\hat{x}) > 0 \tag{A.8}$$

$$\tag{A.9}$$

$f'(x)$ bezeichnet die erste und $f''(x)$ die zweite Ableitung von $f(x)$ nach x. In Spezialfällen ist es möglich für ein lokales Minimum zu zeigen, daß es ein globales Minimum ist.

Erweitern wir nun das Problem auf Funktionen $f(\boldsymbol{x})$ von n Variablen, geht (A.5) in

$$\min_{\boldsymbol{x}\in\mathbb{R}^n} f(\boldsymbol{x}) \tag{A.10}$$

über. Genau wie im eindimensionalen Fall gelten für ein lokales Minimum notwendige und hinreichende Bedingungen, und zwar:

$$\text{Notwendige Bedingung 1. Ordnung} : f_{\boldsymbol{x}}(\hat{\boldsymbol{x}}) = 0$$

$$\text{Notwendige Bedingung 2. Ordnung} : f_{\boldsymbol{x}}(\hat{\boldsymbol{x}}) = 0 \text{ und } f_{\boldsymbol{xx}}(\hat{\boldsymbol{x}}) \geq 0$$

$$\text{Hinreichende Bedingung} : f_{\boldsymbol{x}}(\hat{\boldsymbol{x}}) = 0 \text{ und } f_{\boldsymbol{xx}}(\hat{\boldsymbol{x}}) > 0$$

$f_{\boldsymbol{x}}(\boldsymbol{x})$ mit $[f_{\boldsymbol{x}}(\boldsymbol{x})]_i = \partial_{x_i} f(\boldsymbol{x})$ bezeichnet den Gradienten und $f_{\boldsymbol{xx}}(\boldsymbol{x})$ mit $[f_{\boldsymbol{xx}}(\boldsymbol{x})]_{ij} = \partial_{x_i}\partial_{x_j} f(\boldsymbol{x})$ die Hesse-Matrix von $f(\boldsymbol{x})$. Die Notation $\boldsymbol{M} > 0$ bezeichnet die Definitheit einer Matrix $\boldsymbol{M}$:

$$\begin{aligned}
\boldsymbol{M} > 0 &: \boldsymbol{M} \text{ ist positiv definit} \\
\boldsymbol{M} \geq 0 &: \boldsymbol{M} \text{ ist positiv semidefinit} \\
\boldsymbol{M} = 0 &: \boldsymbol{M} \text{ ist indefinit} \\
\boldsymbol{M} \leq 0 &: \boldsymbol{M} \text{ ist negativ semidefinit} \\
\boldsymbol{M} < 0 &: \boldsymbol{M} \text{ ist negativ definit}
\end{aligned}$$

A.4.2 Gleichungsnebenbedingungen

Nun nehmen wir m Gleichungsnebenbedingungen hinzu, so daß (A.10) in

$$\min_{\boldsymbol{x}\in\mathcal{X}} f(\boldsymbol{x}), \qquad \text{mit } \mathcal{X} = \{\boldsymbol{x} \,|\, \boldsymbol{c}(\boldsymbol{x}) = 0\} \tag{A.11}$$

übergeht. Die m Gleichungsnebenbedingungen sind in der vektorwertigen Funktion $\boldsymbol{c}(\boldsymbol{x})$ zusammengefaßt. Ein Punkt, der die Nebenbedingungen erfüllt, nennt man zulässigen Punkt. Wir definieren nun zunächst die *Lagrange-Funktion*, die die Formulierung der Bedingungen für ein lokales Minimum erleichtert:

$$L(\boldsymbol{x}, \boldsymbol{\lambda}) = f(\boldsymbol{x}) + \boldsymbol{\lambda}^{\mathrm{T}}\, \boldsymbol{c}(\boldsymbol{x}) \tag{A.12}$$

$\boldsymbol{\lambda} \in \mathbb{R}^m$ ist der Vektor der sogenannten *Lagrange-Multiplikatoren*. Die Bedingungen für ein lokales Minimum lauten nun: Es existiert ein $\hat{\boldsymbol{\lambda}} \in \mathbb{R}^m$, so daß gilt

Notwendige Bedingung 1. Ordnung :	$L_{\boldsymbol{x}}(\hat{\boldsymbol{x}}, \hat{\boldsymbol{\lambda}}) = f_{\boldsymbol{x}}(\hat{\boldsymbol{x}}) + \boldsymbol{c}_{\boldsymbol{x}}{}^{\mathrm{T}}(\hat{\boldsymbol{x}})\, \hat{\boldsymbol{\lambda}} = 0$ $L_{\boldsymbol{\lambda}}(\hat{\boldsymbol{x}}, \hat{\boldsymbol{\lambda}}) = \boldsymbol{c}(\hat{\boldsymbol{x}}) = 0$
Notwendige Bedingung 2. Ordnung :	Notwendige Bedingung 1. Ordnung und $L_{\boldsymbol{xx}}(\hat{\boldsymbol{x}}, \hat{\boldsymbol{\lambda}}) \geq 0$, mit der Restriktion $\mathcal{X} = \{\delta\boldsymbol{x} \,|\, \boldsymbol{c}_{\boldsymbol{x}}(\hat{\boldsymbol{x}})\, \delta\boldsymbol{x} = 0\}$
Hinreichende Bedingung :	Notwendige Bedingung 1. Ordnung und $L_{\boldsymbol{xx}}(\hat{\boldsymbol{x}}, \hat{\boldsymbol{\lambda}}) > 0$, mit der Restriktion $\mathcal{X}$

Eine Matrix unter einer Restriktion wie $\mathcal{X}$ auf Definitheit zu prüfen, erfordert spezielle Kriterien. Diese übergehen wir hier, da wir keine Definitheitsprüfung unter Restriktion vornehmen werden. Mit der notwendigen Bedingung 1. Ordnung erhalten wir also ein Gleichungssystem mit $n + m$ Variablen und $n + m$ Gleichungen, welches wir lösen müssen, um die Kandidaten für lokale Minima zu finden.

A.4.3 Ungleichungsnebenbedingungen

Nun kommen wir zur eigentlichen Aufgabe, nämlich zur Minimierung mit Gleichungs- und Ungleichungsnebenbedingungen. Die l Ungleichungsnebenbedingungen werden, ähnlich wie die Gleichungsnebenbedingungen, in einem Vektor $\boldsymbol{h}(\boldsymbol{x})$ zusammengefaßt. In einem zulässigen Punkt teilt man nun die l Ungleichungsnebenbedingungen in l^{a} *aktive Ungleichungsnebenbedingungen* $\boldsymbol{h}^{\mathrm{a}}(\boldsymbol{x})$ mit $\boldsymbol{h}(\boldsymbol{x})^{\mathrm{a}} = 0$ und l^{i} *inaktive Ungleichungsnebenbedingungen* $\boldsymbol{h}^{\mathrm{i}}(\boldsymbol{x})$ mit $\boldsymbol{h}(\boldsymbol{x})^{\mathrm{i}} < 0$ ein. Die folgenden Bedingungen gelten nur für Probleme, die die sogenannte Qualifikationsbedingung erfüllen. Diese ist unter anderem dann erfüllt, wenn $\boldsymbol{c}(\boldsymbol{x})$ und $\boldsymbol{h}^{\mathrm{a}}(\boldsymbol{x})$ linear sind. Dies bei unserer Problemstellung der Fall. Wir definieren nun die verallgemeinerte Lagrange-Funktion:

$$L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = f(\boldsymbol{x}) + \boldsymbol{\lambda}^{\mathrm{T}}\, \boldsymbol{c}(\boldsymbol{x}) + \boldsymbol{\mu}^{\mathrm{T}}\, \boldsymbol{h}(\boldsymbol{x}) \tag{A.13}$$

$\boldsymbol{\mu} \in \mathbb{R}^l$ ist der Vektor der sogenannten *Kuhn-Tucker-Multiplikatoren*. Die Bedingungen für ein lokales Minimum lauten nun: Es existieren $\hat{\boldsymbol{\lambda}} \in \mathbb{R}^m$ und $\hat{\boldsymbol{\mu}} \in \mathbb{R}^l$, so daß gilt

$$L_{\boldsymbol{x}}(\hat{\boldsymbol{x}}, \hat{\boldsymbol{\lambda}}, \hat{\boldsymbol{\mu}}) = f_{\boldsymbol{x}}(\hat{\boldsymbol{x}}) + \boldsymbol{c_x}^{\mathrm{T}}(\hat{\boldsymbol{x}})\,\hat{\boldsymbol{\lambda}} + \boldsymbol{h_x}^{\mathrm{T}}(\hat{\boldsymbol{x}})\,\hat{\boldsymbol{\mu}} = 0 \tag{A.14a}$$

$$L_{\boldsymbol{\lambda}}(\hat{\boldsymbol{x}}, \hat{\boldsymbol{\lambda}}, \hat{\boldsymbol{\mu}}) = \boldsymbol{c}(\hat{\boldsymbol{x}}) = 0 \tag{A.14b}$$

Notwendige Bedingung 1. Ordnung :

$$\boldsymbol{h}(\hat{\boldsymbol{x}}) \le 0 \tag{A.14c}$$

$$\boldsymbol{h}^{\mathrm{T}}(\hat{\boldsymbol{x}})\,\hat{\boldsymbol{\mu}} = 0 \tag{A.14d}$$

$$\hat{\boldsymbol{\mu}} \ge 0 \tag{A.14e}$$

Notwendige Bedingung 2. Ordnung : Notwendige Bedingung 1. Ordnung und

$$L_{\boldsymbol{xx}}(\hat{\boldsymbol{x}}, \hat{\boldsymbol{\lambda}}, \hat{\boldsymbol{\mu}}) \ge 0, \quad \text{mit der Restriktion } \mathcal{X} = \tag{A.14f}$$

$$\{\delta\mathcal{X} \mid \boldsymbol{c_x}(\hat{\boldsymbol{x}})\,\delta\boldsymbol{x} = 0, \boldsymbol{h}^{\mathrm{a}}_{\boldsymbol{x}}(\hat{\boldsymbol{x}})\,\delta\boldsymbol{x} = 0, \boldsymbol{h}^{\mathrm{i}}_{\boldsymbol{x}}(\hat{\boldsymbol{x}})\,\delta\boldsymbol{x} \le 0\}$$

Hinreichende Bedingung : Notwendige Bedingung 1. Ordnung und

$$L_{\boldsymbol{xx}}(\hat{\boldsymbol{x}}, \hat{\boldsymbol{\lambda}}, \hat{\boldsymbol{\mu}}) > 0, \quad \text{mit der Restriktion } \mathcal{X} \tag{A.14g}$$

Wegen (A.14c) und (A.14e) ist $h_i(\hat{\boldsymbol{x}}) \le 0$ und $\hat{\mu}_i \ge 0$ und daraus folgt, daß $h_i(\hat{\boldsymbol{x}})\,\hat{\mu}_i \le 0\ \forall\, i$. Daher enthält (A.14d) ausschließlich nichtpositive Summanden und kann durch

$$h_i(\hat{\boldsymbol{x}})\,\hat{\mu}_i = 0\ \forall\, i \tag{A.15}$$

ersetzt werden. Wie wir sehen, werden die Bedingungen für ein lokales Minimum unter Gleichungs- und Ungleichungsnebenbedingungen recht kompliziert.

A.4.4 Beispiel

Um zu sehen, wie man nun praktisch ein Minimum bestimmt, berechnen wir ein einfaches Beispiel, das unserer Optimierungsaufgabe ähnlich ist. Das Problem sei wie folgt:

Minimiere die Funktion

$$f(\boldsymbol{x}) = f(x_1, x_2) = -x_1^2 - x_2^2$$

unter der Gleichungsnebenbedingung

$$x_1 + x_2 = 1$$

und den Ungleichungsnebenbedingungen

$$x_1 \ge 0 \quad \text{und} \quad x_2 \ge 0\,.$$

Da es nur eine Gleichungsnebenbedingung gibt, reduziert sich der Vektor $\boldsymbol{c}(\boldsymbol{x})$ zu einem Skalar:

$$c(x_1, x_2) = x_1 + x_2 - 1 = 0$$

Die Ungleichungsnebenbedingungen führen auf den Vektor

$$\boldsymbol{h}(x_1, x_2) = \begin{pmatrix} -x_1 \\ -x_2 \end{pmatrix} \le 0$$

Wir erhalten also die Lagrange-Funktion

$$\begin{aligned} L(\boldsymbol{x}, \lambda, \boldsymbol{\mu}) &= f(\boldsymbol{x}) + \lambda\, c(\boldsymbol{x}) + \boldsymbol{\mu}^{\mathrm{T}}\, \boldsymbol{h}(\boldsymbol{x}) \\ &= -x_1^2 - x_2^2 + \lambda(x_1 + x_2 - 1) - \mu_1 x_1 - \mu_2 x_2 \end{aligned} \tag{A.16}$$

Aus der Nebenbedigung 1. Ordnung erhalten wir die Gleichungen

$$\frac{\partial L(\boldsymbol{x}, \lambda, \boldsymbol{\mu})}{\partial x_1} = -2x_1 + \lambda - \mu_1 = 0 \tag{A.17a}$$

$$\frac{\partial L(\boldsymbol{x}, \lambda, \boldsymbol{\mu})}{\partial x_2} = -2x_2 + \lambda - \mu_2 = 0 \tag{A.17b}$$

$$\frac{\partial L(\boldsymbol{x}, \lambda, \boldsymbol{\mu})}{\partial \lambda} = x_1 + x_2 - 1 = 0\,. \tag{A.17c}$$

Addition von (A.17a) und (A.17b) ergibt

$$\begin{aligned} &-2\ \underbrace{(x_1 + x_2)}_{=1 \text{ wg. (A.17c)}}\ +2\lambda - \mu_1 - \mu_2 = 0 \\ \rightarrow \quad &\lambda = \frac{\mu_1 + \mu_2}{2} + 1\,. \end{aligned}$$

Wir erhalten als Bestimmungsgleichungen für x_1 und x_2

$$\begin{aligned} -2x_1 + \frac{\mu_2 - \mu_1}{2} + 1 = 0 \\ -2x_2 + \frac{\mu_1 - \mu_2}{2} + 1 = 0\,. \end{aligned} \tag{A.18}$$

Nun untersuchen wir die Fälle, in denen Ungleichungsnebenbedingungen aktiv bzw. inaktiv sind.

Fall 1. : Beide Ungleichungsnebenbedingungen seien inaktiv, d. h. $h_1(\hat{\boldsymbol{x}}) < 0$ und $h_2(\hat{\boldsymbol{x}}) < 0$, daraus folgt wegen (A.15) $\mu_1 = \mu_2 = 0$. Dann erhalten wir für (A.18)

$$\begin{aligned} -2x_1 + 1 = 0 \quad &\rightarrow \quad x_1 = \frac{1}{2} \\ -2x_2 + 1 = 0 \quad &\rightarrow \quad x_2 = \frac{1}{2} \end{aligned} \tag{A.19}$$

Fall 2. : Die erste Ungleichungsnebenbedingung sei aktiv, die zweite inaktiv, d. h. $h_1(\hat{\boldsymbol{x}}) = -x_1 = 0$ und $h_2(\hat{\boldsymbol{x}}) < 0$, daraus folgt wegen (A.15) $\mu_1 \geq 0$ und $\mu_2 = 0$. Dann erhalten wir für (A.18)

$$\begin{aligned} -\frac{\mu_1}{2} + 1 = 0 \quad &\rightarrow \quad \mu_1 = 2 \\ -2x_2 + \frac{\mu_1}{2} + 1 = 0 \quad &\rightarrow \quad x_2 = 1 \end{aligned} \tag{A.20}$$

Fall 3. : Die erste Ungleichungsnebenbedingung sei inaktiv, die zweite aktiv, d. h. $h_1(\hat{\boldsymbol{x}}) < 0$ und $h_2(\hat{\boldsymbol{x}}) = -x_2 = 0$, daraus folgt wegen (A.15) $\mu_1 = 0$ und $\mu_2 \geq 0$. Dann erhalten wir für (A.18)

$$\begin{aligned} -\frac{\mu_2}{2} + 1 = 0 \quad &\rightarrow \quad \mu_2 = 2 \\ -2x_1 + \frac{\mu_2}{2} + 1 = 0 \quad &\rightarrow \quad x_1 = 1 \end{aligned} \tag{A.21}$$

Fall 4. : Beide Ungleichungsnebenbedingungen seien aktiv, d. h. $h_1(\hat{\boldsymbol{x}}) = -x_1 = 0$ und $h_2(\hat{\boldsymbol{x}}) = -x_2 = 0$. Dies verletzt aber die Nebenbedingung (A.17c), weswegen dieser Fall nicht zulässig ist.

Wir haben nun drei Kandidaten, die die Bedingung 1. Ordnung erfüllen. Sie haben die Funktionswerte

$$\begin{aligned} f(1/2, 1/2) &= -1/2 \\ f(0, 1) &= -1 \\ f(1, 0) &= -1 \ . \end{aligned}$$

Da es keine weiteren Kandidaten gibt, sind die Punkte $(0, 1)$ und $(1, 0)$ globale Minima der Funktion $f(\boldsymbol{x})$ unter den gegebenen Nebenbedingungen. Die Untersuchung der Hesse-Matrix, die wir hier übergehen, ergibt, daß der Punkt $(1/2, 1/2)$ ein lokales (und damit eindeutiges globales) Maximum ist.

Um die Kandidaten für mögliche Minima zu finden, muß man also alle Kombinationen von aktiven und inaktive Ungleichungsnebenbedingungen überprüfen. Die Zahl der möglichen Kombinationen wächst aber schon bei moderaten Problemgrößen so schnell an, daß dann eine Überprüfung aller Kombinationen nicht mehr möglich ist. Betrachten wir als Beispiel eine 8×8 Hyperpermutationsmatrix $\boldsymbol{H}$, die einem 3-Bit Knoten entspricht. Sie hat 64 Ungleichungsnebenbedingungen $h_{ij} \geq 0$. Es gäbe also allein für diese Ungleichungsnebenbedingungen 2^{64} Fälle wie im obigen Beispiel zu überprüfen, da jede Ungleichungsnebenbedingungen aktiv oder inaktiv sein kann. Solche Probleme sind also auf diese Weise nicht zu lösen.

Tabellenverzeichnis

Abbildungsverzeichnis

Index

Literaturverzeichnis

R. Al-Alawi und T. J. Stonham. A training strategy and functional analysis of digital multilayer neural networks. *Journal of Intelligent Systems*, 2(1-4):53–93, 1992. Special Issue.

I. Aleksander. Memory networks for practical vision systems. In O. J. Braddick and A. C. Sleight, editors, *Physical and Biological Processing of Images*, pages 244–259. Springer-verlag, 1982.

I. Aleksander, R. Evans, und W. Penny. Magnus: An iconic neural state machine. In *Proc. Weightless Neural Net. Conf.*, York, 1993.

I. Aleksander, W. V. Thomas, und P. A. Bowden. Wisard: A radical step forward in image recognition. *Sensor Review*, 3(4):120–124, 1984.

Igor Aleksander. From wisard to magnus: A family of weightless virtual neural machines. In Austin (1998), pages 18–30.

Igor Aleksander und T. J. Stonham. Guide to pattern recognition using random-access memories. *Computers and Digital Techniques*, 2(1):29–40, feb 1979. SLN: Single Layer Net Diversity.

I. Alexander. Wisard: A component for image unterstanding. *IEEE Proc. 135*, E(5), 1985.

Igor Alexander, editor. *Neural Computing Architectures: The Design of Brain–Like Machines*. MIT Press, 1989. ISBN 0-262-01110-7.

Luís B. Almeida. *Neural Computing Architectures*, chapter Backpropagation in non-feedforward networks. In , Alexander (1989), 1989. ISBN 0-262-01110-7.

J. A. Anderson, A. Pelionisz, and E. Rosenfeld, editors. *Neurocomputing 2: Directions of Research*. MIT Press, 1990.

James A. Anderson and Edward Rosenfeld, editors. *Neurocomputing: Foundations of Research*. MIT Press, Cambridge, Massachusetts, 1988. ISBN 0-262-01097-6.

J. Austin. A review of ram based neural networks. In *Proc. of the Fourth International Conference on Microelectronics for Neural Networks and Fuzzy Systems*, pages 58–66. IEEE Computer Society Press, 1994.

J. Austin und S. Buckle. The practical application of binary neural networks. In *Proceedings of the UNICON Seminar on Adaptive Computing and Information*, 1994. URL `http://www.cs.york.ac.uk/arch/neural/publications/1994/001.ps.Z`.

J. Austin und T. J. Stonham. An associative memory for use in image recognition and occlusion analysis. *Image and Vision Computing*, 5(4):251–261, nov 1987.

James Austin, editor. *RAM-based Neural Networks*, volume 9 of *Progress in neural Processing*. World Scientific Publishing, 1998.

W. W. Bledsoe und I. Browning. Pattern recognition and reading by machine. In *Proc. Joint Comp. Conference*, pages 255–232, 1959.

W. W. Bledsoe und I. Browning. Pattern recognition and reading by machine. In Leonard Uhr, editor, *Pattern Recognition*, pages 301–316. John Wiley & Sons, Inc., 1966.

Heinrich Braun. Optimierung neuronaler Netze durch Lernen und Evolution. Im Rahmen des Interdisziplinären Kollegs 1998, Günne am Möhnesee, 1998. URL `http://www.tzi.uni-bremen.de/ik98/prog/kursunterlagen/d5/Ik98.html`.

Volker Breuer. *Zeitreihenanalyse mit Stochastischen Automaten: Klassifikation und Modellbildung*. PhD thesis, Universität Kiel, 1995.

Igor Aleksander Catherine Myers. Learning algorithms for probabilistic neural nets. In *INNS Annual Meeting, Boston Mass.*, page 205, 1988.

R. Chin und C. Dyer. Model-based recognition in robot vision. *Computing Surveys*, 18(1): 67–108, 1986.

J. D. Cowan und D. H. Sharp. Neural nets and artificial intelligence. *Dædalus, Journal of the American Academy of Arts and Sciences*, 117:85–121, 1988.

Marc Davio. Kronecker products and shuffle algebra. *IEEE Transactions on Computers*, C-30 (2):116–125, February 1981.

Ulrich Derigs. Anwendung von Standardmodellen der kombinatorischen Optimierung auf das Problem der optimalen Sensorreduktion. Unveröffentlichte Studie, June 1995.

Scott E. Fahlman. An empirical study of learning speed in back-propagation networks. In D. Touretzky, G. Hintor, and T. Sejnowski, editors, *Proceedings of the 1988 Connectionist Models Summer School*. Carnegie Mellon University, 1989.

Filho, Bisset, und Fairhurst. A goal seeking neuron for boolean neural networks. In *Proc. International Neural Networks Conference*, volume 2, pages 894–897, Paris, France, 1990. IEEE.

Filho, Bisset, und Fairhurst. Adaptive pattern recognition using goal seeking neurons. *Pattern Recognition Letters*, 12, mar 1991.

Filho, Bisset, und Fairhurst. Architectures for goal-seeking neurons. *Journal of Intelligent Systems*, 2(1-4):95–119, 1992. Special Issue.

Edward Fredkin und Tommaso Toffoli. Conservative logic. *International Journal of Theoretical Physics*, 21(3/4):219–253, 1982.

N. Goddard, K.J. Lynne, T. Mintz, und L.Bukys. The rochester connectionist simulator. Technical Report 233, Univ. of Rochester, NY, oct 1989.

D. Gorse und G.J. Taylor. An analysis of noisy ram and neural nets. *Physica D*, 34:90–114, 1989.

Martin Grötschel und Manfred Padberg. Die optimierte Odyssee. *Spektrum der Wissenschaft*, 4:76–83, 1999.

K. N. Gurney. Weightet nodes and ram-nets: A unified approach. *Journal of Intelligent Systems*, 2(1-4):155–185, 1992a.

K. N. Gurney und M. J. Wright. Digital nets and intelligent systems. *Journal of Intelligent Systems*, 2(1-4):1–9, 1992. Special Issue.

Kevin Gurney. Training nets of hardware realizable sigma-pi units. *Neural Networks*, 5:289–303, 1992b.

Kevin Gurney. Training nets of stochastic units using system identification. *Neural Networks*, 6:133–145, 1993.

D.O. Hebb. *The Organization of Behavior*. John Wiley & Sons, New York, 1949.

Robert Hecht-Nielsen. *Neurocomputing*. Addison–Wesley, 1989.

Werner Heise und Pasquale Quattrocchi. *Informations- und Codierungstheorie*. Springer, 1995. ISBN 3-540-57477-8.

G. Howells, D. L. Bisset, und M. C. Fairhurst. A new paradigm for ram-based neural networks. In Austin (1998), pages 130–139.

H Jaeger. A short introduction to observable operator models for stochastic processes. In R. Trappl, editor, *Proceedings of the Cybernetics and Systems 98 conference*, volume 1, pages 38–43. Austrian Society for Cybernetic Studies, 1998.

William James. *Principles of Psychology*. Henry Holt, New York, 1890.

S.A Kauffman. Metabolic stability and epigenisis in randomly constructed genetic nets. *J. Theoretical Biology*, 22:437–467, 1969.

E. Kienzle. Vergleichende Klassifikation von Sitzbelegungsdaten. Technical report, Daimler-Chrysler, 1998.

T Kohonen. *Self-Organization and Associative Memory*. Springer-Verlag, Berlin, Heidelberg, 1984.

Y. Lamdan und H.J. Wolfson. Geometric hashing: A general and efficient model-based recognition scheme. In *International Conference on Computer Vision*, pages 238–249, 1988.

Kevin J. Lang und Michael J. Witbrock. Learning to tell two spirals apart. In *Proceedings of the 1988 Connectionist Models Summer School, Morgan Kaufmann*, 1988.

S. Lin und B.W. Kernighan. An effective heuristic for the traveling- salesman problem. *OR*, 21:498–516, 1973.

D. Martland. Auto-associative pattern storage using synchronous boolean networks. In *Proceedings of the first IEEE Conference on Neural Networks*, pages 355–366, San Diego, 1987.

W. S. McCullough und W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bull. Math. Biophysics*, 5:115–133, 1943.

John R. McDonnel und Don Waagen. Evolving neural network connectivity. In *IEEE Int. Conf. on Neural Networks*, pages 863–868, 1993.

Marvin L. Minsky und Seymour A. Papert. *Perceptrons*. MIT Press, Cambridge, Massachusetts, expanded edition, 1988. ISBN 0-262-63111-3 (pbk.).

M. Morciniec und R. Rohwer. Benchmarking n-tuple classifier with statlog datasets. In Austin (1998), pages 53–60.

S. Muroga, T.Tsuboi, und C. R. Bough. Enumeration of threshold functions of eight variables. *IEEE TC*, C-19(9):818–825, 1970.

Bruce A. Murtagh und Michael A. Saunders. Minos 5.4 users guide. Technical Report Technical Report SOL 83-20R, Stanford University, 1995.

Catherine E. Myers. *Delay Learning in Artificial Neural Networks*, chapter RAM-based nodes and networks. Neural Computing Series. Chapman & Hall, 1992.

J. A. Nelder und R. Mead. A simplex method for function minimization. *Computer Journal*, 7:308–313, 1965.

G. L. Nemhauser, A. H. G. Rinnooy Kan, and M. J. Todd, editors. *Optimization*, volume 1 of *Handbooks in operations reseach and management science*. North-Holland, 1989.

Matthias Oberländer. Hyperpermutationsnetzwerke dritten Grades sind vollständig – das Lokalisierungstheorem. 1995.

Matthias Oberländer. Stochastische Tabellenoptimierung — Sensorreduktion mit der PAP-Heuristik. Technical report, DaimlerChrysler, 1997.

Matthias Oberländer. Hyperpermutation networks – a discrete approach to machined perception. In *Third Weightless Neural Networks Workshop*. University of York, mar 1999.

David D. Olmsted. History and principles of neural networks. WWW, 1998. URL: neurocomputing.org/history.htm.

Markos Papageorgiou. *Optimierung*. R. Oldenbourg Verlag GmbH, München, 1991.

Frank Pasemann. Characterization of periodic attractors in neural ring networks. *Neural Network*, 8(3):421–429, 1995.

L. Personnaz and G. Dreyfus, editors. *Neural Networks from Models to Applications*, 1988. I.D.S.E.T. Paris.

M. J. D. Powell. A direct search optimization method that models the objective and constraint functions by linear interpolation. In *Advances in Optimization and Numerical Analysis, Proceedings of the Sixth workshop on Optimization and Numerical Analysis, Oaxaca, Mexico*, volume 275 of *Mathematics and its Applications*, pages 51–67. Kluwer Academic Publishers, 1994.

M. J. D. Powell. Direct search algorithms for optimization calculations. URL `http://www.damtp.cam.ac.uk/user/na/NA_papers/NA1998_04.ps.gz`. To be published in Acta Numerica, Vol. 7, 1998.

W. H. Press, S. A. Teukolsky, W. T. Vetterling, und B. P. Flannery. *Numerical Recipes in C*. Cambridge University Press, 1992.

Lawrence R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–285, February 1989.

N Rashevsky. *Mathematical Biophysics*. University of Chicago Press, 1938.

Raúl Rojas. *Theorie der neuronalen Netze*. Springer-Verlag, 1996.

F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65:386–408, 1958.

David E. Rumelhart, James L. McClelland, und PDP Research Group. *Parallel Distributed Processing*, volume 1. MIT Press, Cambridge, Massachusetts, 1986. ISBN 0-262-18120-7 (vol.1), 0-262-13218-4 (vol.2), ISBN 0-262-18123-1 (set).

Jürgen Schürmann. *Pattern Classification*. John Wiley & Sons, Inc., New York, 1996.

E.H. Simpson. Measurement of diversity. *Nature*, 163:688, 1949.

G. Smith und J. Austin. Analysing aerial photographs with adam. In *International Joint Conference on Neural Networks*, volume 3, pages 49–54, jun 1992.

Tommaso Toffoli. Reversible computing. Technical Report MIT/LCS/TM-151, MIT, Lab. for Computer Science, feb 1980a.

Tommaso Toffoli. Reversible computing. In De Bakker and Van Leeuwen, editors, *Automata, Languages, and Programming*, pages 632–644. Springer-Verlag, jul 1980b. 7th Colloquium, Noordwijkerhout, NL.

W. Utschick und W. Weichselberger. A posteriori codes for multiclass discrimination. In *European Conference of Circuit Theory and Design*, 1999.

Vladimir N. Vapnik. *The nature of statistical learning theory*. Springer-Verlag, New York, 1995. ISBN 0-387-94559-8.

Francisco J. Varela. *Eine Skizze aktueller Perspektiven*. Suhrkamp, 1990.

C. C. Walker. Attractor dominance patterns in sparsely connected boolean nets. *Physica D*, 45:441–451, 1990.

Peter Zagorski. Entwicklung evolutionärer Algorithmen zur Optimierung der Topologie und des Generalisierungsverhaltens von Multilayer Perceptrons. Master's thesis, Universität Karlsruhe, 1994. URL `ftp://springbank.ira.uka.de/pub/neuro/braun/pz_diplom.ps.Z`.

Andreas Zell. *Simulation Neuronaler Netze*. Addison Wesley, Germany, 1994.

Danksagung

An dieser Stelle bedanke ich mich bei denjenigen, die zum Gelingen dieser Arbeit beigetragen haben:

- bei Professor Schuster, der mir einerseits Freiheit bei der Verfassung dieser Arbeit gelassen hat, andererseits aber genau artikuliert hat, was ihm bei dieser Arbeit wichtig ist.
- bei Matthias Oberländer, der durch die Entwicklung der Hyperpermutationsnetzwerke den Grundbaustein für die Frage nach der Optimierung rückgekoppelter Hyperpermutationsnetzwerke gelegt hat. Seine Bereitschaft, sich jederzeit kritisch mit meinen Gedanken auseinanderzusetzen, war für mich sehr wertvoll.
- bei meinen Kollegen für die gute Arbeitsatmosphäre.
- bei Anke für das Korrekturlesen.
- bei meinen Eltern für die Unterstützung während des Studiums.

Eidesstattliche Erklärung

Hiermit versichere ich an Eides Statt, daß vorliegende Abhandlung – abgesehen von der Beratung durch meinen akademischen Lehrer – nach Inhalt und Form meine eigene Arbeit ist. Sie hat weder ganz noch zum Teil bereits einer anderen Stelle im Rahmen eines Prüfungsverfahrens vorgelegen.

(Tilo Schwarz)