NETZWERKARCHITEKTUREN, ZIELFUNKTIONEN UND KETTENREGEL

Habilitationsschrift
vorgelegt von
Jürgen Schmidhuber

zur Erlangung des Grades *Dr. rer. nat. habil.*
Technische Universität München
15. April 1993

Inhaltsverzeichnis

1 EINFÜHRUNG **1**

1.1 GRADIENTENBASIERTE NETZWERKE 3
1.2 ZIELE DER ARBEIT 5
1.3 SCHEMATISCHE VORGEHENSWEISE 6
1.4 NÄHERER BLICK AUF DAS GRUNDSCHEMA 7
1.4.1 GRUNDBAUSTEINE DES NETZWERKARCHITEKTEN 7
1.4.2 SINNVOLLE ZIELFUNKTIONEN 9
1.4.3 DIE KETTENREGEL 10
1.5 GLIEDERUNG DER ARBEIT 12
1.6 WIEDERKEHRENDE NOTATION 15
1.7 DANKSAGUNG 16

2 VOLLSTÄNDIGE RÜCKKOPPLUNG **19**

2.1 ARCHITEKTUR UND AKTIVIERUNGSDYNAMIK . . . 20
2.2 PERFORMANZMASS 22
2.3 ERSTE METHODE: *'BACK-PROPAGATION THROUGH TIME' (BPTT)* 24
2.4 ZWEITE METHODE: *'REAL-TIME RECURRENT LEARNING' (RTRL)* 26
2.5 DRITTE METHODE: EIN $O(n^3)$ VERFAHREN 28
2.6 EXPERIMENTE 33
2.6.1 EXPERIMENTE MIT KLAMMERBALANCIERENDER TURINGMASCHINE 33
2.6.2 EXPERIMENTE ZUM ERLERNEN REGULÄRER GRAMMATIKEN 35
2.6.3 SCHWACHSTELLEN REKURRENTER NETZE . 38
2.6.4 EXPERIMENTE: PRAKTISCHE GRENZEN REKURRENTER NETZE 39
2.6.5 ZWEI PROBLEME DER OBIGEN ALGORITHMEN 41

2.7 SCHLUSSBEMERKUNGEN 42

3 DYNAMISCHE VERBINDUNGEN 43
3.1 ARCHITEKTUR UND NETZWERKDYNAMIK 44
3.1.1 ARCHITEKTUR 1 46
3.1.2 ARCHITEKTUR 2 46
3.2 PERFORMANZMASS 48
3.3 ABLEITUNG DER ALGORITHMEN 48
3.3.1 *On-Line* VERSUS *Off-Line* 51
3.3.2 ALTERNATIVE VERWENDUNGEN DER KETTENREGEL 52
3.3.3 BESCHRÄNKUNGEN UND ERWEITERUNGEN . 52
3.4 EXPERIMENTE 52
3.4.1 EIN EXPERIMENT MIT UNBEKANNTEN ZEITLICHEN VERZÖGERUNGEN 52
3.4.2 EXPERIMENTE ZUR TEMPORÄREN VARIABLENBINDUNG 54
3.5 SCHLUSSBEMERKUNGEN 56

4 DISTANZIERTE PERFORMANZMASSE 59
4.1 WELTMODELLBAUER 60
4.1.1 ZWEI PERFORMANZMASSE 60
4.1.2 ARCHITEKTUR 61
4.1.3 ALGORITHMUS 62
4.2 ADAPTIVE SUBZIELGENERIERUNG 65
4.2.1 ARCHITEKTUR 66
4.2.2 PERFORMANZMASS 70
4.2.3 DIE ALGORITHMEN 70
4.3 EXPERIMENTE ZUR HINDERNISVERMEIDUNG 73
4.3.1 EXPERIMENTE MIT NICHT ADAPTIVEM E . . 73
4.3.2 EXPERIMENTE MIT ADAPTIVEM E 76
4.3.3 SCHRANKEN DER SUBZIELGENERATOREN . . 82

5 UNÜBERWACHTES LERNEN: ZIELFUNKTIONEN 83
5.1 UNÜBERWACHTES LERNEN: WOZU? 84
5.2 DAS INFOMAX-PRINZIP 86
5.2.1 MEHRDIMENSIONALE AUSGABEN 87
5.3 UNÜBERWACHTE BILDUNG LOKALER REPRÄSENTATIONEN 88
5.4 UNÜBERWACHTES LERNEN UND STATISTISCHE UNABHÄNGIGKEIT 90
5.4.1 G-MAX 90

5.4.2 FAKTORIELLE CODES 91
5.5 EXTRAKTION VORHERSAGBARER KONZEPTE . . . 94
5.5.1 METHODEN ZUR DEFINITION VON D_l 98
5.5.2 BEZUG ZU FRÜHEREN ARBEITEN: IMAX . . . 101
5.5.3 EXPERIMENTE ZUR VORHERSAGBARKEITSMAXIMIERUNG 102
5.5.4 SCHLUSSBEMERKUNGEN ZU 5.5 104

6 VORHERSAGBARKEITSMINIMIERUNG 107
6.1 PROBLEMFORMULIERUNG 109
6.2 GRUNDPRINZIP UND GRUNDLEGENDE ARCHITEKTUR 110
6.3 PERFORMANZMASSE FÜR DIE DREI KRITERIEN . . 112
6.3.1 EINE ZIELFUNKTION FÜR DAS UNABHÄNGIGKEITSKRITERIUM 112
6.3.2 EINE ZIELFUNKTION FÜR DAS BINÄRKRITERIUM 113
6.3.3 EINE ZIELFUNKTION FÜR DAS REVERSIBILITÄTSKRITERIUM 113
6.3.4 ZIELFUNKTIONSKOMBINATIONEN 115
6.3.5 REELLWERTIGE CODES 115
6.3.6 ENTFERNUNG DES GLOBALEN INVERTIBILITÄTSTERMS 115
6.3.7 EIN NACHTEIL OBIGER VERFAHREN 116
6.4 LOKALE BEDINGTE VARIANZMAXIMIERUNG 116
6.5 'NEURONALE' IMPLEMENTIERUNG 117
6.6 EXPERIMENTE 119
6.6.1 EXPERIMENTE MIT GLEICHVERTEILTEN EINGABEN 120
6.6.2 EXPERIMENTE MIT 'OCCAMS RASIERMESSER' 121
6.6.3 EXPERIMENTE MIT UNGLEICH VERTEILTEN EINGABEN 123
6.6.4 EXPERIMENTE MIT BUCHSTABENBILDERN MIT UNTERSCHIEDLICHEN AUFTRETENSWAHRSCHEINLICHKEITEN 124
6.6.5 EXPERIMENTE ZUR KOMBINATION VON VORHERSAGBARKEITSMAXIMIERUNG MIT VORHERSAGBARKEITSMINIMIERUNG 128
6.7 VORHERSAGBARKEITSMINIMIERUNG UND ZEIT . . 129
6.7.1 EXPERIMENTE MIT SEQUENZEN 131
6.8 ABSCHLIESSENDE BEMERKUNGEN 132
6.9 APPENDIX 133

6.9.1 EINE NICHT PREDIKTORBASIERTE ZIELFUNKTION ZUR AUFFINDUNG FAKTORIELLER CODES 133
6.9.2 ZUR VERMUTUNG 6.4.1 134
6.9.3 ÄQUIVALENZ VON V_C UND $V - H$ 136

7 UNÜBERWACHTE GESCHICHTSKOMPRESSION 141
7.1 NACHTEILE DER REINEN GRADIENTENABSTIEGSVERFAHREN 143
7.2 DAS PRINZIP DER GESCHICHTSKOMPRESSION . . . 144
7.3 EINE SELBSTORGANISIERENDE PREDIKTORENHIERARCHIE 146
7.3.1 PRINZIP 147
7.3.2 DETAILS: EINE SERIE VON ZIELFUNKTIONEN 149
7.4 EXPERIMENTE MIT PREDIKTORENHIERARCHIEN . 150
7.4.1 EXPERIMENT: VERZÖGERUNGEN MIT MEHR ALS 1000 ZEITSCHRITTEN 150
7.4.2 EXPERIMENTE ZUR TEXTKOMPRESSION . . . 153
7.5 KOLLAPS DER HIERARCHIE 156
7.5.1 PRINZIP 157
7.5.2 ARCHITEKTURDETAILS 159
7.5.3 EXPERIMENT MIT DER KOLLAPS-ARCHITEKTUR163
7.5.4 SCHRANKEN DER KOLLAPS-ARCHITEKTUR . 165
7.6 KONTINUIERLICHE GESCHICHTSKOMPRESSION . . 165
7.6.1 KONTINUIERLICHE GESCHICHTSKOMPRESSION MIT RAAMs 166
7.6.2 ARCHITEKTUR UND ZIELFUNKTIONEN 170
7.6.3 EXPERIMENT MIT DER RAAM-ARCHITEKTUR 171
7.7 SCHLUSSBEMERKUNGEN 174

8 AUSBLICK: 'SELBSTREFERENTIELLE' NEURONALE NETZE 175
8.1 EIN 'SELBSTREFERENTIELLES' REKURRENTES NETZ 177
8.1.1 'KONVENTIONELLE' ASPEKTE DER ARCHITEKTUR 179
8.1.2 'SELBSTREFERENTIELLE' ASPEKTE DER ARCHITEKTUR 180
8.1.3 MÄCHTIGKEIT DER KLASSE DER VOM NETZ BERECHENBAREN FUNKTIONEN 183
8.2 ALGORITHMUS FÜR ÜBERWACHTES LERNEN 187
8.2.1 PERFORMANZMASS 188
8.2.2 HERLEITUNG DES ALGORITHMUS 188

8.3 ABSCHLIESSENDE BEMERKUNGEN ZU KAPITEL 8 . 191
8.4 SCHLUSSBEMERKUNGEN ZUR ARBEIT 193

Kapitel 1

EINFÜHRUNG

Das Forschungsgebiet der *'künstlichen neuronalen Netzwerke'* (KNN) hat in den letzten Jahren bedeutende Fortschritte gemacht.

Zum einen wuchs die Zahl der Anwendungsbeispiele, bei denen KNN-Ansätze zu besseren Ergebnissen führen als traditionelle Standardverfahren: Das gegenwärtig erfolgreichste Verfahren zur Proteinstrukturvorhersage verwendet KNN [25][132]. Das zur Zeit beste System zur Erkennung handgeschriebener Ziffern basiert auf KNN [47](siehe auch [137]). KNN erwiesen sich bei gewissen Zeitreihenvorhersageproblemen gegenüber konventionellen Verfahren als überlegen (e.g. Sonnenfleckenvorhersage, [142]). Die komplexen Probleme bei der Walzwerksteuerung im Stahlwerk ließen sich durch eine KNN-Methode deutlich besser in den Griff bekommen als durch die klassische *'state of the art'*-Lösung – Steuerungsexperten bewerteten die durch das neuartige Verfahren gewonnenen Resultate als 'höchst überzeugend' [82]. In gewissen Fällen demonstrieren KNN im Vergleich zu traditionellen Klassifikationsmethoden aus der Statistik überlegene Performanz [136], was mit ein Grund dafür ist, daß sich viele Statistiker für KNN interessieren (e.g. [135], [145]). Oft zeigt sich auch, daß KNN in sinnvoller Weise in herkömmliche Systeme eingebettet werden können – in der Sprachverarbeitung beispielsweise erwies sich eine Kombination von *'Hidden Markov Models' (HMM)* und KNN als funktionstüchtiger als der klassische reine HMM-Ansatz [15]. Das mit Abstand beste *'Backgammon'* spielende Rechnerprogramm ist ein KNN (Gewinner der *'Backgammon Computer Olympiad'*, [134]). Letzteres ist überhaupt das erste maschinelle Lernsystem, welches je ein nennenswertes Brettspielturnier gewonnen hat – es hat inzwischen eine dem Niveau menschlicher Experten vergleichbare Spielstärke erreicht und fährt fort, sich zu verbessern, indem es gegen sich selbst spielt (R. Sutton, persönliche Kommunikation).

Zum anderen festigten sich die theoretischen Grundlagen des Feldes. Statistiker sowie Forscher aus dem der Statistik verwandten Bereich der Lerntheorie (insbesondere dem Bereich 'PAC-Learning' – 'PAC' steht für *'Probably Almost Correct'*) wandten ihre Konzepte erfolgreich auf KNN an, was Licht auf die theoretischen Möglichkeiten und Schranken gewisser Netzwerktypen warf (e.g. [137], [13], [32], [58], [39]). So wurden beispielsweise Begriffe wie 'Generalisierungsfähigkeit' im mathematisch statistischen Sinne präzise definiert, was für gegebene Verteilungen von zu klassifizierenden Mustern die Grundlage für asymptotische Abschätzungen der hinreichenden Zahl von Trainingsbeispielen bzw. Netzgrößen für gewisse KNN-Typen und Aufgaben schuf ([10], [52], [58], [130]). Die Ansicht, daß umgekehrt auch das Feld der Statistik wesentlich von KNN-Methoden profitieren kann, hat sich mittlerweile durchgesetzt (e.g. [145], [136]).

Die meisten (und sämtliche der oben aufgeführten) erfolgreichen Anwendungsbeispiele beinhalten als wesentliche Komponente Varianten des (aus später auszuführenden Gründen) sogenannten azyklischen *back-propagation (BP)*-Netzwerkes (auch die Mehrheit der theoretischen Untersuchungen befaßt sich mit BP-Netzwerken). BP-Netze sind im wesentlichen Funktionsapproximatoren, welche Eingabemuster auf Ausgabemuster abbilden. In Einführungen zu KNN werden schwerpunktmäßig zumeist ihr Potential für massiv parallele Informationsverarbeitung und ihre Eignung zur assoziativen Musterklassifikation hervorgehoben (wir werden im Rahmen dieser Einführung später noch detailliert auf BP eingehen). BP-Netze liefern das Standardbeispiel für die Vorgehensweise des nicht biologisch orientierten Neuroinformatikers: Statt (wie manche der ersten biologisch orientierten NN-Forscher) einen Lernalgorithmus zu postulieren und auszuprobieren, ob er zu sinnvollen Ergebnissen führt, leitet der formal orientierte Neuroinformatiker den Lernalgorithmus aus einem sinnvollen Optimalitätskriterium her.

Diese formal orientierte Vorgehensweise soll im folgenden genauer erläutert werden. Ich werde mich dabei *nicht* auf die Klasse der azyklischen BP-Funktionspproximatoren beschränken. Der Grund hierfür ist: Es gibt relativ neuartige KNN, deren Fähigkeiten beträchtlich über simple Musterassoziation hinausgehen. Gewisse KNN gestatten im Prinzip beliebiges *dynamisches* Verhalten sowie die Berechnung beliebiger auch auf einem herkömmlichen sequentiell arbeitenden Digitalrechner berechenbarer Funktionen. Derartige über einfach BP-Netze hinausweisende KNN werden einen Schwerpunkt dieser Arbeit bilden.

1.1 GRADIENTENBASIERTE NETZWERKE

Architektur und Aktivierungsdynamik eines KNNs entsprechen der unveränderlichen *'Hardware'* eines konventionellen Rechners. Die *'Software'* (das Programm) des KNNs wird durch gewisse reellwertige Parameter verkörpert, die *Gewichte* genannt werden.

Das im Vergleich zu herkömmlichen Ansätzen (beispiesweise aus dem Bereich der künstlichen Intelligenz und des maschinellen Lernens) hervorstechendste Merkmal der in der vorliegenden Arbeit zu untersuchenden Systemen liegt nicht etwa in ihrer massiven Parallelität o.ä. Der zentrale Unterschied zwischen den Programmen konventioneller Rechner und den Programmen der in den folgenden Kapiteln zu besprechenden Architekturen besteht vielmehr in dieser grundlegenden Tatsache: Die Funktionen, die die Ausgaben unserer Architekturen liefern, *lassen sich (mathematisch) bezüglich der 'Software' differenzieren.*

Definiert man ein *Performanzmaß* (im folgenden häufig auch *Zielfunktion* genannt), das auf irgendeine sinnvolle Weise *gewünschtes* Programmverhalten zum Ausdruck bringt und seinerseits bezüglich der die KNN-Ausgaben berechnenden Funktion differenzierbar ist, so läßt sich dank der *Kettenregel* auch das Performanzmaß bezüglich der 'Software' differenzieren.

Mit der mathematisch ableitbaren Zielfunktion spezifizieren wir unsere Ansprüche an eine zunächst unbekannte gesuchte Rechenvorschrift. (Dies stellt einen entscheidenden Unterschied zu herkömmlichen Ansätzen zur formalen Lösungsspezifikation dar.) Die Zielfunktion entspricht dem vorgegebenen Optimalitätskriterium: Nehmen wir an, der Wert der Zielfunktion sei maximal, falls ein bestimmter Algorithmus die gewünschten Effekte zeitigt, und um so kleiner, je 'ungünstiger' sich der Algorithmus verhält. Partielle Differenzierung der Zielfunktion bezüglich aller Gewichtsparameter liefert uns einen Gradienten im Raum der von der Netzwerkarchitektur erlaubten Algorithmen. Dieser Gradient gibt uns entweder Aufschluß darüber, in welcher Richtung im Algorithmenraum Programme zu finden sind, die bessere Performanz erwarten lassen, oder er teilt uns mit, daß sich in der 'Umgebung' der gegenwärtig im KNN implementierten Rechenvorschrift keine Algorithmen höherer Güte befinden.

Die Gradienteninformation erlaubt uns die Konstruktion von *Meta*-Algorithmen zum Auffinden günstiger Algorithmen. Im einfachsten Fall führt solch ein Meta-Algorithmus nach zufälliger Anfangsinitialisierung aller Gewichte einen auf sukzessiven Iterationen basierenden Gradientenanstieg bezüglich der zu maximierenden Zielfunktion durch. Ein Iterations-

schritt setzt sich dabei zusammen aus (a) der Auswertung des Zielfunktionsgradienten für die gegenwärtigen Gewichtsparameter und (b) der Änderung des Gewichtsvektors in Proportion zum Gradienten[1] (der Proportionalitätsfaktor selbst wird häufig als *'Lernrate'* bezeichnet). Der Meta-Algorithmus hinterläßt dabei im durch die möglichen Gewichte definierten Raum aller erlaubten Rechenvorschriften eine Trajektorie, deren Endpunkt (ein sub-optimales Programm) gewöhnlich einem lokalen Maximum der Zielfunktion entspricht. Hat man einmal einen derartigen gradientenbasierten Meta-Algorithmus hergeleitet, so hat man automatisch auch den Beweis dafür, daß seine Anwendung im asymptotischen Fall (bei gegen Null gehender Lernrate) nur zur Verbesserung der gegenwärtigen Gewichtsmatrix führen kann – nicht zur Verschlechterung (Experimente sind allerdings vonnöten, um akzeptable positive Lernraten für praktische Anwendungen zu finden).

Da der Raum möglicher Programme gewöhnlich alle Sorten von Nichtlinearitäten erlaubt, kann nicht garantiert werden, daß BP oder die in dieser Schrift vorgestellten Meta-Algorithmen für ein gegebenes Problem stets im ersten Anlauf das *beste* aller Programme finden werden. Gerade bei NP-vollständigen Problemklassen ist es höchst unwahrscheinlich, daß Gradientenanstieg in einer geeigneten Zielfunktion immer zur *optimalen* Lösung führt. Das soll uns nicht allzusehr betrüben – sogar wir Menschen geben uns meist mit sub-optimalen Verhaltensweisen zufrieden. Wir werden das Problem lokaler Maxima im folgenden (falls überhaupt nötig) durch die einfachste Standardlösung angreifen, welche darin besteht, ausgehend von verschiedenen zufällig gewählten Initialprogrammen solange ein oft wiederholtes, lokales, gradientenbasiertes Suchen im Raum der von der Architektur gestatteten Rechenvorschriften durchzuführen, bis ein Programm mit zufriedenstellender Performanz gefunden worden ist. Selbst bei Zielfunktionen mit vielen lokalen Minima stellt dies i.a. eine wesentlich befriedigendere Lösung dar als beispielsweise eine ungerichtete Zufallssuche.

Ein gradientenbasierter Meta-Algorithmus (wie oben umrissen) wird im folgenden stets als *Lern*algorithmus bezeichnet. Das Konzept der mathematisch durch partielle Differentation herleitbaren Lernalgorithmen ist ein selten betontes Markenzeichen eines umfassenden Teilbereichs der KNN-Forschung und hebt gradientenbasierte KNN deutlich vom Hintergrund sonstiger KI-Ansätze zum maschinellen Lernen ab. Differenzierbare Netzwerkarchitekturen bilden das Schwergewicht dieser Arbeit.

[1] Die Numerik kennt raffiniertere gradientenbasierte Verfahren als den einfachen Gradientenanstieg. Im Rahmen dieser Arbeit wird uns jedoch statt der Frage, wie man einmal gewonnene Gradienten am effizientesten zur Zielfunktionsmaximierung einsetzt, vor allem der Entwurf angemessener problemspezifischer Architekturen sowie unterschiedliche Ansätze zur Gradientenberechnung selbst interessieren.

1.2 ZIELE DER ARBEIT

I. Behandlung theoretischer Aspekte. Unterschiedliche Problemstellungen erfordern unterschiedliche Performanzmaße und (häufig modulare) Architekturen. Die vorliegende Schrift will an verschiedenen, großenteils neuartigen Architekturen detailliert aufzeigen, wie sich bedeutsame wiederkehrende Probleme des überwachten Lernens, des sogenannten *'Reinforcement'*-Lernens und des unüberwachten Lernens in der Sprache der Zielfunktionen formulieren lassen, und wie es mit Hilfe der Kettenregel möglich ist, bei verschiedensten gegebenen Architekturen aus der (durch Zielfunktionen gegebenen) formalen Spezifikation des gewünschten Programmverhaltens in mathematisch zu rechtfertigender Weise gradientenbasierte Lernalgorithmen herzuleiten.

Dabei soll u.a. demonstriert werden, daß der bereits erwähnte (und später noch detailliert zu beschreibende) neuronale Standardalgorithmus BP das Potential der Kettenregel zum Entwurf adaptiver Systeme nicht annähernd ausschöpft. Die Lektüre dieser Schrift soll dem Leser einerseits ein Gefühl für die mannigfaltigen Anwendungsmöglichkeiten der in Abschnitt 1.3 darzustellenden schematischen Entwurfssystematik vermitteln und ihn andererseits mit einer Reihe von zur Entwicklung von Lernverfahren brauchbaren Werkzeugen und Konzepten ausstatten. Es sollte dabei klar werden, daß die formale Spezifikation gewünschter *'Software'*-Eigenschaften durch Zielfunktionen, die bezüglich der *'Software'* differenzierbar sind, eine sehr allgemeine und interessante Alternative zu herkömmlichen Lösungsspezifikationsansätzen darstellt.

II. Experimentelle Untersuchungen. Obwohl großmaßstäbliche Anwendungen der abgeleiteten Verfahren im Rahmen dieser Arbeit nicht von Interesse sind, sollen die Arbeitsweise und die Schranken neuartiger Algorithmen im folgenden durch die Beschreibung zahlreicher Experimente verdeutlicht werden.

III. Fragen außerhalb des Schwergewichts der Arbeit. *1.* Wir beschäftigen uns *nicht* mit Lernverfahren für nicht-differenzierbare oder nicht-deterministische KNN ([146], [147], [38], [36], [8], [62], [77], [9], [140]). Das soll nicht etwa heißen, daß derartige KNN nicht ebenfalls ihre Berechtigung haben – in der Tat hat sich der Autor selbst längere Zeit damit beschäftigt, solche Systeme zu untersuchen und Lernalgorithmen für sie zu entwerfen (z.B. [90], [91], [100], [97], [95], [51], [92], [99], [103], [93]).

2. Auch auf Methoden zur *'Regularisierung'* nicht vom Lernalgorithmus adjustierbarer Parameter (wie z.B. 'Lernraten', Zahl der parallel arbeitenden neuronenartigen Prozessoren, relative Gewichtung von Zielfunktionstermen zur Erzwingung *'einfacher'* Lösungen [18] [142], etc.) werden wir aufgrund ihrer Orthogonalität zum Ziel der Arbeit keine Rücksicht neh-

men. Der interessierte Leser sei statt dessen auf MacKays Untersuchungen sowie auf die von ihm angeführten Referenzen verwiesen [53][54].

3. Die folgenden Kapitel konzentrieren sich auf die einfachste Art gradientenbasierter Suche, nämlich den Gradientenanstieg oder -abstieg. Zwar stellt uns die Numerik trickreichere und häufig effizientere gradientenbasierte Standardverfahren zur Verfügung (z.B. konjugierte Gradientenverfahren, *'line search'*, Methoden zweiter Ordnung, effiziente Approximationen von Methoden zweiter Ordnung, etc., siehe auch Arbeiten zu beschleunigten Gradientenabstiegsverfahren in e.g. [24][89][67][127]). Derartige Modifizierungen haben allerdings nichts mit der Essenz vorliegender Schrift zu tun und werden von ihr daher ignoriert. Der Schwerpunkt dieser Arbeit liegt auf der Erstellung geeigneter 'differenzierbarer' Architekturen für verschiedenartige Problemklassen sowie der zugehörigen kettenregelbasierten Gradientenberechnung, nicht auf der Frage, wie man den Gradienten schließlich am effizientesten zur Zielfunktionsmaximierung einsetzt.

1.3 SCHEMATISCHE VORGEHENSWEISE

Die allen in dieser Arbeit vorgestellten Methoden zugrundeliegende Philosophie ist folgende: Lernverfahren sollten nicht einfach 'erfunden' werden und sich nicht nur durch intuitive Überlegungen rechtfertigen lassen. Statt dessen sollten sie mathematisch sauber aus einer gegebenen Spezifikation ableitbar sein.

Zum Entwurf von Lernalgorithmen werden wir uns stets der folgenden drei Schritte umfassenden schematischen Vorgehensweise bedienen.

(1) Verschaffe Klarheit darüber, welche Menge erlaubter Rechenvorschriften dem Lernalgorithmus als Basis für die Auswahl günstiger Programme dienen soll. Definiere eine geeignete Netzwerkarchitektur mit bezüglich gewisser Gewichtsparameter differenzierbaren Netzausgaben, so daß jedes Element der Menge erlaubter Rechenvorschriften in der Netzarchitektur implementiert werden kann.

(2) Formuliere das Problem durch Spezifikation geeigneter differenzierbarer Zielfunktionen, die maximiert oder minimiert werden sollen.

(3) Leite mittels Kettenregel aus Zielfunktionen und Architektur einen gradientenbasierten Lernalgorithmus her, von dem sich beweisen läßt, daß seine Anwendung im asymptotischen Fall (bei gegen Null gehender Lernrate) zur Auffindung einer suboptimalen, im Netz implementierten Rechenvorschrift führt.

Vor allem die Punkte (1) und (2) bieten genügend Spielraum zur Entfal-

tung menschlichen Einfallsreichtums. Trotz der Tatsache, daß obiges Grundschema zum Entwurf von Lernregeln viel Platz für Kreativität läßt, birgt es doch eine erstrebenswerte Systematik, die sich in weiten Bereichen der KNN-Forschung mittlerweile durchgesetzt hat.

1.4 NÄHERER BLICK AUF DAS GRUNDSCHEMA

Es gibt keine allgemeine Methode für das Entwerfen geeigneter Architekturen und Zielfunktionen zur Lösung gegebener Probleme – hier ist die Phantasie des Netzwerkarchitekten gefragt! Wie jedoch die folgenden Kapitel aufzeigen werden, erweist es sich bei der Lösung häufig auftretender Unterprobleme oft als zweckmäßig, auf gewisse grundlegende, immer wieder von neuem anwendbare Konzepte zurückzugreifen.

Der dritte Punkt des Grundschemas (die formale Ableitung des Lernalgorithmus) ist am ehesten bis zu einem gewissen Grade automatisierbar. Wie sich jedoch in Kapitel 2 herausstellen wird, lassen sich selbst hier auf unterschiedlichen formalen Wegen mehr oder weniger effiziente Algorithmen (mit ansonsten identischem Verhalten) herleiten.

1.4.1 GRUNDBAUSTEINE DES NETZWERKARCHITEKTEN

Welche Netzwerkarchitekturen sind sinnvoll? Eine 'vernünftige' Netzarchitektur spiegelt gewöhnlich die beim Lösungsansatz für eine gegebene Problemklasse auftretenden Unterprobleme wider. Oft erhält man die geeignete Gesamtarchitektur durch Zusammenschaltung verschiedener Untermodule mit jeweils eigener Zielfunktion (Kapitel 3, 4, 6, 7). Ausgaben des einen adaptiven Moduls werden häufig zu Eingaben eines anderen adaptiven Moduls – im Rahmen dieser Arbeit muß allerdings dafür Sorge getragen werden, daß durch die Hintereinanderschaltung verschiedener KNN das Differenzierbarkeitskriterium nicht verletzt wird.

Gewisse Grundbausteine erweisen sich in höchst unterschiedlichen Kontexten als sinnvoll und werden daher in unseren Netzarchitekturen immer wieder auftreten. Alle in den folgenden Kapiteln untersuchten neuronalen Netze bestehen aus einem Paar (K, V): K ist eine Menge von zu biologischen Neuronen korrespondierenden *Knoten* , $V \subseteq K \times K$ ist eine Menge von zu biologischen Nervenfasern korrespondierenden gerichteten *Kanten* oder *Verbindungen* zwischen den Knoten. Zu jedem Knoten gehört zu jedem Zeitpunkt eine der Feuerrate eines Neurons entsprechende reelle *Aktivation*. Zu jeder Kante gehört ein zu einer Synapsenstärke korrespondierendes

reelles *Gewicht*. Eine Teilmenge von K wird als die Menge der Eingabeknoten ausgezeichnet. Eingabeknoten werden zu jedem Zeitpunkt durch sensorische Wahrnehmung von Eingabevektoren aus der Umgebung aktiviert. Alle anderen Knoten (die sogenannten Nichteingabeknoten) berechnen ihre Aktivationen durch differenzierbare Funktionen der Aktivationen von Knoten, von denen sie Verbindungen erhalten. Häufig wird eine Teilmenge der Nichteingabeknoten als die Menge der Ausgabeknoten ausgezeichnet. Ausgabeknoten dienen in der Regel zur Übermittlung der vom Netzwerk berechneten Ergebnisse an die Umgebung – sie stellen die Schnittstelle zur Außenwelt dar. Zielfunktionen sind häufig Funktionen der Aktivationen der Ausgabeknoten.

Azyklische Netze. Häufig lassen sich Netzwerke ohne rückkoppelnde Verbindungen in sinnvoller Weise als Untermodule in umfangreichere adaptive Systeme einbetten (siehe Kapitel 3, 4, 6, 7). Wir definieren, was bei einem azyklischen Netz oder Netzmodul unter einer *Lage* zu verstehen ist. Die erste Lage ist die Menge aller Eingabeknoten. Die n-te Lage ist die Menge aller Knoten, zu denen ausgehend von der ersten Lage mindestens ein Kantenpfad der Länge $n-1$ führt, aber kein Kantenpfad der Länge $m \geq n$. Ein Kantenpfad der Länge $n-1$ ist hierbei eine Liste

$$((k_1,k_2),(k_2,k_3),\ldots,(k_{n-1},k_n)) \ mit \ k_i \in K, i=1,\ldots,n \ und \ (k_i,k_{i+1}) \in V, i=1,\ldots,n-1.$$

Der Sinn dieser Definition besteht darin, auch direkte Verbindungen ('Abkürzungen') zwischen nicht aufeinanderfolgenden Lagen in angemessener Weise zu berücksichtigen. Die Lage mit der höchsten Nummer wird häufig als Ausgabelage benutzt. Es läßt sich zeigen, daß jede beliebige stetige Funktion durch ein geeignetes hinreichend umfangreiches Netzwerk mit 4 Lagen beliebig genau approximiert werden kann (e.g. [45]).

Rekurrente Netze. Ist es möglich, in einem gegebenen Netzwerk wenigstens einen Knoten k zu finden, von dem aus ein Kantenpfad auf k selbst führt, so sprechen wir von einem *zyklischen* oder *rekurrenten* Netzwerk oder einem Netzwerk mit zyklischen oder rekurrenten Verbindungen. Viele der zu besprechenden Netze werden zyklische Verbindungen aufweisen. Der Grund dafür ist: In rekurrenten Netzwerken vermögen zu einem gegebenen Zeitpunkt stattfindende Eingabeereignisse einen Einfluß darauf auszuüben, wie *spätere* Eingaben verarbeitet werden. Somit eröffnet sich ein Potential für ein Kurzzeitgedächtnis in Form wandernder Aktivationen. Im allgemeinen können beliebige zeitliche Verzögerungen zwischen irgendwelchen Eingaben und ihren späteren Konsequenzen auftreten. Rückkoppelnde Verbindungen erlauben sequenzverarbeitende Algorithmen, die über simple Eingabe/Ausgabe-Assoziation hinausgehen.

1.4.2 SINNVOLLE ZIELFUNKTIONEN

Wodurch zeichnet sich ein sinnvolles Performanzmaß aus? Um dieser Frage auf den Grund zu gehen, vollziehen wir die gemeinhin gemachte Unterscheidung zwischen drei verschiedenen Arten des Lernens. In allen drei Fällen reagiert ein lernendes KNN mit (in der Regel zeitlich variierenden) Ausgaben auf (in der Regel zeitlich variierende) Eingaben.

(1) Überwachtes Lernen. Ein externer Lehrer definiert zu bestimmten Zeitpunkten *gewünschte* Ausgaben. Eine geeignete zu minimierende differenzierbare Zielfunktion besteht in der Summe aller von den verschiedenen Eingabesequenzen eines 'Trainingensembles' verursachten Abweichungen zwischen erwünschten und tatsächlichen Ausgaben (mit diesem Fall beschäftigen sich Kapitel 2 und 3).

(2) 'Reinforcement-Lernen' (mangels aussagekräftiger deutscher Bezeichnung ab jetzt *R-Lernen* genannt). Kein externer Lehrer kennt zu irgendeinem Zeitpunkt die gewünschte Ausgabe bereits im voraus. Die Umgebungsdynamik übersetzt die (als Steuersignale für Muskelmotorik vorstellbaren) Ausgaben jedoch in Änderungen des Umgebungszustandes – eine (oft sehr primitive) Evaluierungsfunktion liefert dem lernenden System eine Rückmeldung (das *'Reinforcement'*, z.B. ein Schmerzsignal) über die 'Güte' des erreichten Zustandes. Kapitel 4 zeigt u.a., wie sich unter der Annahme, daß die Abbildung von Steuersignalen auf *'Reinforcement'* differenzierbar ist, eine Hilfszielfunktion für ein Hilfsmodul (das sogenannte *Umgebungsmodell*) definieren läßt, welches das eigentlich interessante Performanzmaß, definiert durch eine Summe von *Reinforcement*signalen, bezüglich der Steuersignale differenzierbar macht.

Überwachtes Lernen und R-Lernen werden unter dem Oberbegriff *zielgerichtetes Lernen* zusammengefaßt, wobei die Ziele von externen Prozessen vorgegeben werden.

(3) Unüberwachtes Lernen. Die Motivation unüberwachten Lernens erwächst aus dem Wunsch, Repräsentationen der Umgebungseingaben zu finden, die *zielgerichtetes* Lernen vereinfachen oder beschleunigen, *ohne* daß die Ziele von vornherein genau bekannt sind. Oft erweisen sich z.B. unüberwachte Performanzmaße (basierend auf informationstheoretischen Erwägungen) für die Ausfilterung statistischer Redundanz in den Eingaben zur Erstellung kompakter Eingabekodierungen und zur Beschleunigung zielgerichteter Lernalgorithmen als zweckmäßig. Unüberwachte Performanzmaße haben in der jüngsten Literatur viel Aufmerksamkeit erfahren und stellen einen bedeutenden Schwerpunkt dieser Arbeit dar (Kapitel 5, 6, 7).

1.4.3 DIE KETTENREGEL

Bei gegebener Architektur und gegebenen Zielfunktionen verbleibt die formale Aufgabe der Ableitung des Lernalgorithmus. Hilfsmittel hierzu ist die Kettenregel. Sie sei für Funktionen mehrerer Veränderlicher an dieser Stelle in der für unser Anliegen zweckmäßigsten Weise formal niedergeschrieben:

Gegeben sei eine bezüglich all ihrer Parameter differenzierbare Funktion $f : R^n \to R$

$$f(\phi_1(w), \phi_2(w), \ldots, \phi_n(w)) \tag{1.1}$$

wobei alle $\phi_i : R \to R$ mit $i = 1 \ldots n$ ihrerseits nach w ableitbar sind. Dann gilt

$$\frac{\partial f(\phi_1(w), \phi_2(w), \ldots, \phi_n(w))}{\partial w} = \sum_{i=1}^{n} \frac{\partial f}{\partial \phi_i} \frac{\partial \phi_i}{\partial w}. \tag{1.2}$$

f stellt bei unseren Anwendungen häufig ein Performanzmaß oder aber auch bloß die Ausgabe eines adaptiven Untermoduls dar, ϕ_i bezeichnet in vielen Fällen die Aktivation eines Knotens (möglicherweise aber auch die Ausgabe eines weiteren Untermoduls), und w steht oft für einen Gewichtsparameter (vielleicht aber auch für die Ausgabe eines zu einem früheren Zeitpunkt aktiven Knotens oder Untermoduls).

DAS STANDARDBEISPIEL: 'BACK-PROPAGATION'.

Der wohl bekannteste durch einfache Anwendung der Kettenregel abgeleitete Lernalgorithmus ist der einführend bereits erwähnte 'back-propagation' Algorithmus (im folgenden stets BP genannt). BP wurde 1974 erstmals von Werbos im Kontext der Sozialpsychologie beschrieben [143], fand jedoch seinerzeit wenig Beachtung und fiel daraufhin bis etwa Mitte der 80er Jahre der Vergessenheit anheim. Das Verfahren wurde verschiedentlich wiederentdeckt [46][66] und erfreut sich seit Rumelharts, Hintons und Williams' Veröffentlichung [85] des Rufs des beliebtesten Lernverfahrens für KNN. Die Herleitung des BP-Algorithmus sei in dieser Einführung aus zwei Gründen vorgeführt: (1) In den Kapiteln 3, 4, 5, und 6, werden uns BP-Netze in der Gestalt von Untermodulen umfangreicherer Netzwerkarchitekturen begegnen. (2) BP liefert ein besonders einfaches illustratives Beispiel dafür, daß man mit der Kettenregel im Kontext maschinellen Lernens etwas Vernünftiges anfangen kann.

Ein BP-Netz ist ein azyklisches Netz. Die Anzahl der Lagen im Netz sei mit l bezeicnet. Alle Knoten im Netz seien beliebig durchnumeriert, die Aktivation des k-ten Knotens in Antwort auf einen an der Eingabelage anliegende Eingabevektor x (mit i-ter Komponente x_i) heiße ϕ_k, wobei

$\phi_i = x_i$, falls i Eingabeknoten ist. In der r-ten Lage ($r > 1$) berechnet sich ϕ_k wie folgt:

$$net_k = \sum_{l \in \ Lagen \ <r} w_{kl}\phi_l, \quad \phi_k = f(net_k), \tag{1.3}$$

wobei w_{kl} das Gewicht der gerichteten Verbindung vom Knoten l zum Knoten k ist, und f eine reellwertige differenzierbare Aktivierungsfunktion darstellt.

Die Zielfunktion J ist in vielen Fällen eine zu minimierende *Fehlerfunktion*. Häufig wird der durch ein einzelnes Muster x verursachte Fehler J als Summe der Fehlerquadrate (SFQ)

$$J = \sum_{i \ in \ Ausgabelage} (\phi_i - d_i)^2$$

gewählt, wobei d_i die gewünschte Ausgabe des i-ten Ausgabeknotens bezeichnet. SFQ-Minimierung eignet sich sowohl zur Funktionsapproximation (fast alle BP-Netze werden gewöhnlich zu diesem Zweck eingesetzt) als auch zur Modellierung des *Erwartungswertes* nicht-deterministischer, teilweise durch die gegenwärtige Eingabe bedingter Ereignisse. Letztere selten ausgeschöpfte Möglichkeit wird im 6. Kapitel eine entscheidende Rolle spielen. Es sei erwähnt, daß entropiebasierte Zielfunktionen wie z.B. das Kreuzentropiemaß (e.g. [44], [131]) eine informationstheoretisch begründete Alternative zu SFQ darstellen.

Was immer die genaue Form der Zielfunktion J auch sein mag – wir nehmen an, daß sie in differenzierbarer Form von der Ausgabe des BP-Netzes abhängt. Den Bemerkungen aus der Einleitung folgend, interessiert uns nun für alle Gewichte w_{ij} der Gradient

$$-\frac{\partial J}{\partial w_{ij}} = \delta_i \frac{\partial net_i}{\partial w_{ij}} = \delta_i \phi_j \tag{1.4}$$

mit

$$\delta_i = -\frac{\partial J}{\partial \phi_i}\frac{\partial \phi_i}{\partial net_i} = -\frac{\partial J}{\partial \phi_i} f'(net_i). \tag{1.5}$$

Ist i ein Ausgabeknoten, so gewinnt man den ersten Faktor der rechten Seite von (1.5) sofort durch partielle Ableitung von J nach ϕ_i. Ist i kein Ausgabeknoten und befindet i sich in Lage r, so berechnet sich dieser Faktor mit der Kettenregel (1.2) zu

$$\sum_{k \ in \ Lagen \ >r} w_{ki}\delta_k.$$

Zur Durchführung eines Iterationsschrittes des Gradientenabstiegsprozesses wird jedes Gewicht w_{ij} gemäß

$$w_{ij} = w_{ij} + \triangle w_{ij} = w_{ij} - \alpha \delta_i \phi_j$$

geändert, wobei α eine positive Konstante, die sogenannte Lernrate, bezeichnet. Normalerweise ist mehr als ein Eingabemuster zu berücksichtigen – in diesem Fall hat man w_{ij} einfach proportional zur Summe der entsprechenden Gradienten zu ändern (der Gradient einer Summe ist die Summe der Gradienten). Die sogenannte *off-line* Version von BP spart sich dabei die Ausführung der Gesamtgewichtsänderung bis nach der Präsentation aller Trainingsmuster auf. Im praktischen Einsatz wird findet allerdings häufig eine sogenannte *on-line*-Version Verwendung, bei der Gewichtsänderungen sofort nach der Präsentation jedes Trainingsmusters durchgeführt werden. Sowohl *on-line* als auch *off-line* Version führen bei gegen Null gehendem α einen exakten Gradientenabstieg in J durch. In Anwendungen (bei Lernraten in der Größenordnung von 0.5) erweist sich die *on-line* Version jedoch häufig als günstiger (siehe [26] für eine theoretische Begründung dieses Sachverhalts).

Einige der zahlreichen in der Fachliteratur anzutreffenden BP-Anwendungen führten in gewissen Domänen zu *'state-of-the-art performance'* (siehe Einführung). Wie der Rest dieser Schrift jedoch verdeutlichen wird, gehen die Anwendungsmöglichkeiten der Kettenregel weit über BP in azyklischen Netzen hinaus.

1.5 GLIEDERUNG DER ARBEIT

Kapitel 2 beginnt mit der Untersuchung einer potentiell sehr mächtigen (im Prinzip Turingmaschinen-äquivalenten) Klasse von Netzwerkarchitekturen – den dynamischen, vollständig rückgekoppelten sequenzverarbeitenden Netzen. Nach Definition eines geeigneten Performanzmaßes für überwachtes Lernen beliebiger Mustersequenzen werden mittels Kettenregel drei verschiedene Lernalgorithmen abgeleitet, die allerdings alle exakt denselben Gradienten berechnen und sich lediglich in ihrer Komplexität unterscheiden (was nahelegt, daß auch der dritte Schritt der grundlegenden Vorgehensweise (Abschnitt 1.3) nicht notwendigerweise rein automatisch abläuft, da man die Kettenregel offenbar mehr oder weniger geschickt zum Einsatz bringen kann). Der originäre Beitrag dieses Kapitels besteht aus dem dritten Verfahren (mit Komplexität $O(n^3)$), welches das bisher häufig verwendete zweite Verfahren (mit Komplexität $O(n^4)$) in Proportion zur Anzahl n der Knoten im Netz beschleunigt. Experimente zur Erlernung des

Zustandsübergangsdiagramms einer klammerbalanzierenden Turingmaschine sowie zur Extraktion regulärer Grammatiken aus zeitlich variierenden Eingabeströmen verdeutlichen sowohl die Funktionstüchtigkeit als auch die praktischen Schranken der Algorithmen.

Kapitel 3 (ein weiterer originärer Beitrag) behält das allgemeine Performanzmaß aus Kapitel 1 bei, verwendet jedoch eine Architektur, die auf zwei Netzwerken *ohne* zyklische Verbindungen beruht. Um im Eingabestrom auftretende Ereignisse temporär abspeichern zu können, verwendet eines der Netze die unkonventionellen von v. d. Malsburg vorgeschlagenen *dynamischen Verbindungen*, deren Gewichte sich innerhalb kürzester Zeit dramatisch ändern können. Die Gewichtsänderungen selbst werden durch die Ausgaben des zweiten Netzwerks gesteuert, dessen Lernalgorithmus wieder mit der Kettenregel hergeleitet wird. Das resultierende System zeigt, daß rekurrente Netze nicht die einzige Möglichkeit zur 'neuronalen' Sequenzverarbeitung darstellen. Wie auch experimentell demonstriert wird, weist die Methode weiterhin ein natürliches Potential für das Erlernen temporärer Variablenbindungen auf (Kritiker haben den KNN in der Vergangenheit (wie man sieht, fälschlicherweise) die Fähigkeit zur Variablenbindung abgesprochen).

Kapitel 4 beschäftigt sich mit Performanzmaßen, die keinen expliziten Lehrer erfordern, der die gewünschten Netzausgaben (hier *Steuersignale* genannt) schon im voraus kennt. Statt dessen bewertet eine Evaluierungsfunktion die 'Güte' der (von den Steuersignalen mitverursachten) Umgebungszustände. Der erste Beitrag des Kapitels beschreibt ein Standardverfahren, welches demonstriert, wie ein Hilfsnetzwerk mit separatem Performanzmaß dazu veranlaßt werden kann, ein differenzierbares Modell der Abbildung von 'Steuersignalen' auf Evaluationen von Umgebungszuständen zu bilden, und wie sich dieses Hilfsnetzwerk dergestalt mit dem eigentlich interessierenden Steuernetzwerk zusammenschalten läßt, so daß mittels Kettenregel ein Performanzgradient für die Programme des Steuernetzwerkes herleitbar wird. Der zweite (originäre) Beitrag vertieft die Analyse differenzierbarer Modelle der Effekte von Steuerprogrammen und beschreibt sowohl azyklische als auch rekurrente neuronale 'Subzielgeneratoren'. Letztere sind Netzwerke, die mittels Kettenregel lernen, als Antwort auf eine Kombination des gegenwärtigen Umgebungszustandes und eines gewünschten Zielzustandes eine Liste geeigneter Subziele auszugeben. Mit Hilfe adaptiver Subzielgeneratoren werden experimentell Pfadfindungsprobleme in zweidimensionalen Welten mit Hindernissen gelöst.

Nachdem Kapitel 2, 3, und 4 sich mit zielgerichtetem Lernen beschäftigten, konzentriert sich *Kapitel 5* auf Performanzmaße für *unüberwachtes* Lernen. Beschrieben werden verschiedene in der Literatur aufgetauchte Lernregeln für stationäre Umgebungen (z.B. zur Maximierung der Informations-

transmission zwischen Eingaben und ihren internen Repräsentationen, zur Dekorrelation der Repräsentationskomponenten etc.). Dabei werden wieder nur solche Lernverfahren berücksichtigt, die aus einem sinnvollen Performanzmaß herleitbar sind. Gerechtfertigt werden Methoden für unüberwachtes Lernen durch ihr Potential zur Unterstützung zielgerichteten Lernens. An geeigneter Stelle werden verschiedene eigene Beiträge eingeflochten. Experimente schließen die unüberwachte Extraktion von Stereoinformation aus einfachen Zufallsstereogrammen mit ein – eine neuartige Methode erweist sich hierbei für diese Aufgabenstellung als günstiger als ein älteres, informationstheoretisch begründetes Verfahren.

Kapitel 6 (ein weiterer originärer Beitrag) beschäftigt sich mit dem vielleicht ambitioniertesten Ziel *unüberwachten* Lernens – der Entdeckung 'faktorieller' Codierungen der Umgebungseingaben. Das sind höchst vorteilhafte Repräsentationen, die sich dadurch auszeichnen, daß ihre einzelnen Komponenten statistisch voneinander unabhängig sind. Erstmalig wird eine aus zwei 'miteinander streitenden' Arten von Untermodulen bestehende Netzarchitektur samt zugehörigen Performanzmaßen präsentiert, die die kettenregelmäßige Herleitung von Algorithmen zum Finden binärer faktorieller Codes ermöglicht. Zahlreiche Experimente (unter anderem zur unüberwachten Kodierung von Buchstaben mit unterschiedlichen Auftretenswahrscheinlichkeiten) belegen die Anwendbarkeit der Verfahren zur Redundanzreduktion in Eingabemustern und Eingabemustersequenzen.

Kapitel 7 (ein weiterer originärer Beitrag) untersucht die enormen Vorteile, die sich aus unüberwachtem Lernen in *dynamischen* Umgebungen ergeben können. Grundlage des Kapitels ist das durch theoretische Überlegungen zu rechtfertigende einfache und dennoch neuartige *Prinzip der Geschichtskompression*, welches im wesentlichen besagt, daß nur unerwartete Ereignisse im Eingabestrom nicht-redundante Information tragen. Erwartete Eingaben dürfen im wesentlichen ignoriert werden. Aufbauend auf dem Prinzip der Geschichtskompression werden selbstorganisierende hierarchische Architekturen samt zugehörigen Zielfunktionen entworfen, die kausale Abhängigkeiten im Eingabestrom widerspiegeln und eindeutige *reduzierte* Sequenzbeschreibungen ermöglichen. Letztere können die Aufgaben zusätzlicher zielgerichteter Lerner gewaltig erleichtern. Experimente zum Erlernen regulärer Grammatiken zeigen, daß auf dem Prinzip der Geschichtskompression basierende Systeme in gewissen Fällen mindestens 1000 mal schneller lernen können als die in Kapitel 2 beschriebenen, ohne zusätzliche unüberwachte Performanzmaße auskommenden, sequenzverarbeitenden Algorithmen. Weitere Experimente mit natürlichsprachlichen Texten weisen Schranken der Methode auf. Ein Nebenprodukt der experimentellen Untersuchungen ist ein neuartiges Textkompressionsverfahren, welches bei gewissen Zeitungstexten bereits zu besserer Datenkompression führte als

der weitverbreitete, in einem gewissen informationstheoretischen Sinne optimale Lempel-Ziv Algorithmus, der u.a. auch die Grundlage der *compress*-Funktion des Betriebssytems *UNIX* bildet.

Kapitel 8 (der letzte originäre Beitrag) schließlich liefert (im Rahmen eines Gedankenexperiments) einen theoretischen Ausblick auf 'introspektives' maschinelles Lernen. Ein wesentlicher Unterschied zwischen 'menschlichem' Lernen und existierenden Methoden für maschinelles Lernen besteht darin, daß Menschen über ihr eigenes Lernverhalten reflektieren und ihre Lernstrategien gegebenenfalls ändern können, um sie den von der Umgebung typischerweise gestellten Lernaufgaben anzupassen. Kapitel 8 geht daher über alle vorangegangenen Kapitel hinaus, in dem es fragt, ob sich Gewichtsänderungsalgorithmen selbst so repräsentieren lassen, daß sie sinnvoller Manipulation zugänglich sind. Um die Frage konstruktiv zu beantworten, wird das erste 'selbst-referentielle' rekurrente neuronale Netzwerk präsentiert. Dieses verfügt zum einen über die Fähigkeit, seine eigenen Erfolge und Mißerfolge zu beobachten. Zum anderen besitzt es spezielle Eigenschaften, die es ihm erlauben, *alle* eigenen Gewichtsparameter zu analysieren und explizit zu manipulieren, und zwar einschließlich derjenigen Parameter, die für die Analysier- und Manipulierprozesse zuständig sind. Dank der Allgemeinheit der Architektur existieren keine theoretischen Begrenzungen für die Komplexität der im Netzwerk implementierbaren, zur Selbstmodifikation fähigen Gewichtsänderungsalgorithmen (abgesehen von unvermeidbaren durch die Endlichkeit der Architektur verursachten Zeit und Speicherbegrenzungen). Theoretisch kann das Netz nicht nur seinen eigenen Gewichtsänderungsalgorithmus ändern, sondern auch die Art und Weise, in der es seinen eigenen Gewichtsänderungsalgorithmus ändert, sowie die Art und Weise, in der es den Algorithmus ändert, der seinen eigenen Gewichtsänderungsalgorithmus ändert, und so fort *ad infinitum*. Mittels Kettenregel wird ein *'vorverdrahteter'* Gewichtsänderungsalgorithmus zur Auffindung 'sinnvoller' *selbst-modifizierender* Gewichtsänderungsalgorithmen hergeleitet.

1.6 WIEDERKEHRENDE NOTATION

Soweit nötig, wird jedes Kapitel seine eigenen formalen Bezeichner einführen. Gewisse Symbole werden in den verschiedenen Kapiteln jedoch gleiche oder vergleichbare Bedeutung tragen.

Reelle Vektoren werden durch kursive Kleinbuchstaben dargestellt. Das Superskript p des Vektors v^p zeigt eine Verbindung zum p-ten Eingabemuster an.

Sind dynamische Abläufe und zeitlich gedehnte Sequenzen im Spiel, so stellt $v^p(t)$ gewöhnlich den t-ten Vektor einer zur p-ten Eingabesequenz

gehörigen Reihe von Vektoren dar (wir werden stets mit Digitalrechnern angepaßten diskreten Zeitschritten statt mit kontinuierlicher Zeit arbeiten).

Die i-te Komponente des Vektors v wird stets mit v_i bezeichnet. $v_i^p(t)$ ist demzufolge die i-te Komponente von $v^p(t)$.

Symbole wie x, x^p, $x^p(t)$ denotieren normalerweise Eingabevektoren, Symbole wie y, y^p, $y^p(t)$ stehen gewöhnlich für Ausgabevektoren.

v^T (bzw. Q^T) repräsentiert die Transponierte des Vektors v (bzw. der Matrix Q),

$dim(v)$ bezeichnet die Dimension von v,

$\|v\| = \sqrt{v^T v}$ stellt die Länge von v dar,

'$\circ$' steht für den Konkatenationsoperator: $v^1 \circ v^2$ ist die Konkatenation der Vektoren v^1 und v^2,

$Det(Q)$ repräsentiert die Determinante der Matrix Q,

$P(A)$ bezeichnet die Wahrscheinlichkeit des Ereignisses A,

$P(A \mid B)$ denotiert die bedingte Wahrscheinlichkeit von A unter der Voraussetzung B,

$E(Y)$ stellt den Erwartungswert der Zufallsvariable Y dar,

$E(Y \mid X)$ steht bei gegebenem X für den bedingten Erwartungswert von Y,

$VAR(Y) = E(Y - E(Y))^2$ repräsentiert die Varianz von Y,

$COV(X,Y) = E((Y - E(Y))(X - E(X)))$ bezeichnet die Covarianz der Zufallsvariablen X und Y.

1.7 DANKSAGUNG

Ohne die vielfältige Unterstützung durch meinen Betreuer Wilfried Brauer wäre mir die Anfertigung dieser Schrift nicht möglich gewesen.

Fruchtbare Diskussionen mit den folgenden Personen haben zur Entstehung dieser Arbeit beigetragen: Mike Mozer, Paul Smolensky, Josef Hochreiter, Daniel Prelinger, Peter Dayan, Richard Zemel, David Rumelhart, Radford Neal, Luis Almeida, Sue Becker, Martin Eldracher, Jost Bernasch, Clayton McMillan, und Gerhard Weiss.

Karolin Schmidhuber, meine Mutter, kam zahlreichen syntaktischen Schnitzern auf die Spur.

Besonderer Dank gebührt auch meinen Mitarbeitern und Studenten, die im Rahmen von Forschungsprojekten, Diplomarbeiten, Fortgeschrittenenpraktika, oder *'masters thesises'* viele der in dieser Arbeit vorgestellten Algorithmen implementierten und experimentell testeten: Jeff Rink und Stefanie Lindstädt von der *University of Colorado at Boulder* sowie Martin Eldracher, Josef Hochreiter, Daniel Prelinger, Klaus Bergner, Reiner

Wahnsiedler, Boris Baginski und Stefan Heil von der *Technischen Universität München*.

Ohne den Zugang zu Rechnern beider oben genannten Universitäten, insbesondere ohne den Zugang zur elektronischen Post (*'email'*), zur von der *Carnegie Mellon University* verwalteten *'connectionists mailing list'*, sowie zur von der *Ohio State University* verwalteten öffentlichen elektronischen KNN-Datenbank (*'neuroprose'*) wäre es mir unmöglich gewesen, stets rechtzeitig die neuesten Informationen über relevante Entwicklungen zu beschaffen.

Der vorliegende Text sowie die zugehörige Literatursammlung wurden mit Hilfe des Textverarbeitungsprogramms LATEX formatiert. Die meisten Abbildungen wurden mit dem Graphiksystem *IDRAW* erstellt.

Die Arbeit wurde im Rahmen eines DFG-Habilitationsstipendiums angefertigt. Einige Vorarbeiten wurden durch ein DFG-Postdoktorandensstipendium sowie durch *NSF award IRI–9058450, grant 90–21* der James S. McDonnell Foundation und durch *DEC external research grant 1250* an Mike Mozer unterstützt. Folgenden Professoren möchte ich für Empfehlungsbriefe (zur Unterstützung meiner Bewerbungen um DFG-Stipendien) danken: Mike Mozer, Paul Smolensky, Ronald Williams, Wilfried Brauer, Luis Almeida.

Kapitel 2

VOLLSTÄNDIGE RÜCKKOPPLUNG

Wir beginnen mit einer allgemeinen Standardarchitektur, die (im selben Sinne wie ein herkömmlicher Rechner) Turingmaschinen-äquivalent ist. Sie eignet sich zur Berechnung beliebiger, auf sequentiell arbeitenden konventionellen Maschinen berechenbarer Funktionen und weist damit über die Schranken einfacher statischer BP-Musterassoziatoren hinaus: Im Prinzip gestattet das System beispielsweise die Extraktion der Regeln einer unbekannten Grammatik aus einem sequentiellen Eingabestrom.

Abschnitt 2.1 definiert dabei zunächst die Netzwerkstruktur und Aktivierungsdynamik, welche beschreiben, wie Eingabesequenzen verarbeitet werden. Daraufhin geben wir eine geeignete Zielfunktion an, die formal unser Ziel spezifiziert, welches darin bestehen soll, beliebige diskrete Ausgabesequenzen mit beliebigen diskreten Eingabesequenzen zu assoziieren. Schließlich leiten wir mit der Kettenregel eine Reihe nicht-lokaler Algorithmen zum Erlernen von Ein-/Ausgabesequenzen ab, die alle denselben exakten Performanzgradienten berechnen, sich aber in ihrer Komplexität unterscheiden. Daraus ist bereits ersichtlich, daß die Kettenregel gelegentlich mehr als einen Lernalgorithmus gestattet und offenbar in mehr oder weniger raffinierter Weise eingesetzt werden kann. Der erste Algorithmus braucht für beliebig lange Traininssequenzen beliebig viel Speicherplatz. Der zweite Algorithmus kommt mit fixem Speicherplatz aus, hat jedoch den Nachteil hoher Zeitkomplexität pro Iterationsschritt. Der dritte neuartige Algorithmus (der originäre Beitrag dieses Kapitels) benötigt nicht mehr Speicherplatz als der zweite, ist jedoch um den Faktor $O(n)$ schneller (wobei n die Anzahl der Knoten im Netzwerk bezeichnet).

Experimente mit klammerbalanzierenden Turingmaschinen und regulären Grammatiken belegen die praktische Anwendbarkeit der Verfahren. Sie demonstrieren aber auch gewisse praktische Schranken rekurrenter Netze für den Fall langer zeitlicher Verzögerungen zwischen Ereignissen und korrelierten gewünschten Ausgaben. Dies wird Motivation für Kapitel 7 liefern.

2.1 ARCHITEKTUR UND AKTIVIERUNGSDYNAMIK

Zu unseren Netzwerken gehören Eingabeknoten und 'interne' Knoten (auch Nichteingabeknoten genannt). Wie im folgenden zu sehen sein wird, werden Eingabeknoten von Eingabemustern aus der Umgebung aktiviert, während interne Knoten ihre Aktivationen gemäß einfacher Regeln aus den Aktivationen der übrigen Knoten berechnen. Die Bezeichnung 'vollständige Rückkopplung' soll besagen, daß jeder Eingabeknoten genau eine gerichtete Verbindung zu jedem internen Knoten aufweist, und daß jedem internen Knoten ebenfalls genau eine gerichtete Verbindung zu jedem internen Knoten (einschließlich sich selbst, siehe Abbildung 2.1) entspringt.

Bei der Wahl formaler Symbole halten wir uns relativ eng an die Notation von Williams und Peng [149]. Um Indizes zu sparen, betrachten wir der Einfachheit halber eine einzelne, s diskrete Zeitschritte umfassende geordnete Sequenz von vektorwertigen Eingabemustern. Zu jedem Zeitpunkt $0 < t \leq s$ ist der reellwertige Vektor der Aktivationen der Eingabeknoten gleich dem t-ten Eingabemuster. Alle Knoten seien in beliebiger Weise eindeutig durchnumeriert. U bezeichne die Menge von Indizes k mit der Eigenschaft, daß $x_k(t)$ die Ausgabe des internen Knotens mit der Nummer k zum Zeitpunkt t ist. I stehe für die Menge von Indizes k mit der Eigenschaft, daß $x_k(t)$ die Eingabe des Eingabeknotens mit der Nummer k zum Zeitpunkt t darstellt. Es sei

$$| I | = m, | U | = n, m = O(n).$$

Das reellwertige skalare Gewicht der gerichteten Verbindung vom Knoten j zum Knoten i wird durch w_{ij} denotiert. Während der Verarbeitung der Eingabesequenz ändern sich die Gewichte nicht. Um jedoch zwischen verschiedenen Instanzen von w_{ij} zu verschiedenen Zeitpunkten unterscheiden zu können, verwenden wir die Notation $w_{ij}(t)$ für das Gewicht der Verbindung vom Knoten j zum Knoten i zum Zeitpunkt t. Dies hat rein formale Gründe: $\forall t: \quad w_{ij}(t) = w_{ij}$.

Die Ablaufdynamik definieren wir für $k \in U$ wie folgt:

$$net_k(0) = 0, \quad \forall t \geq 0 : x_k(t) = f_k(net_k(t)),$$

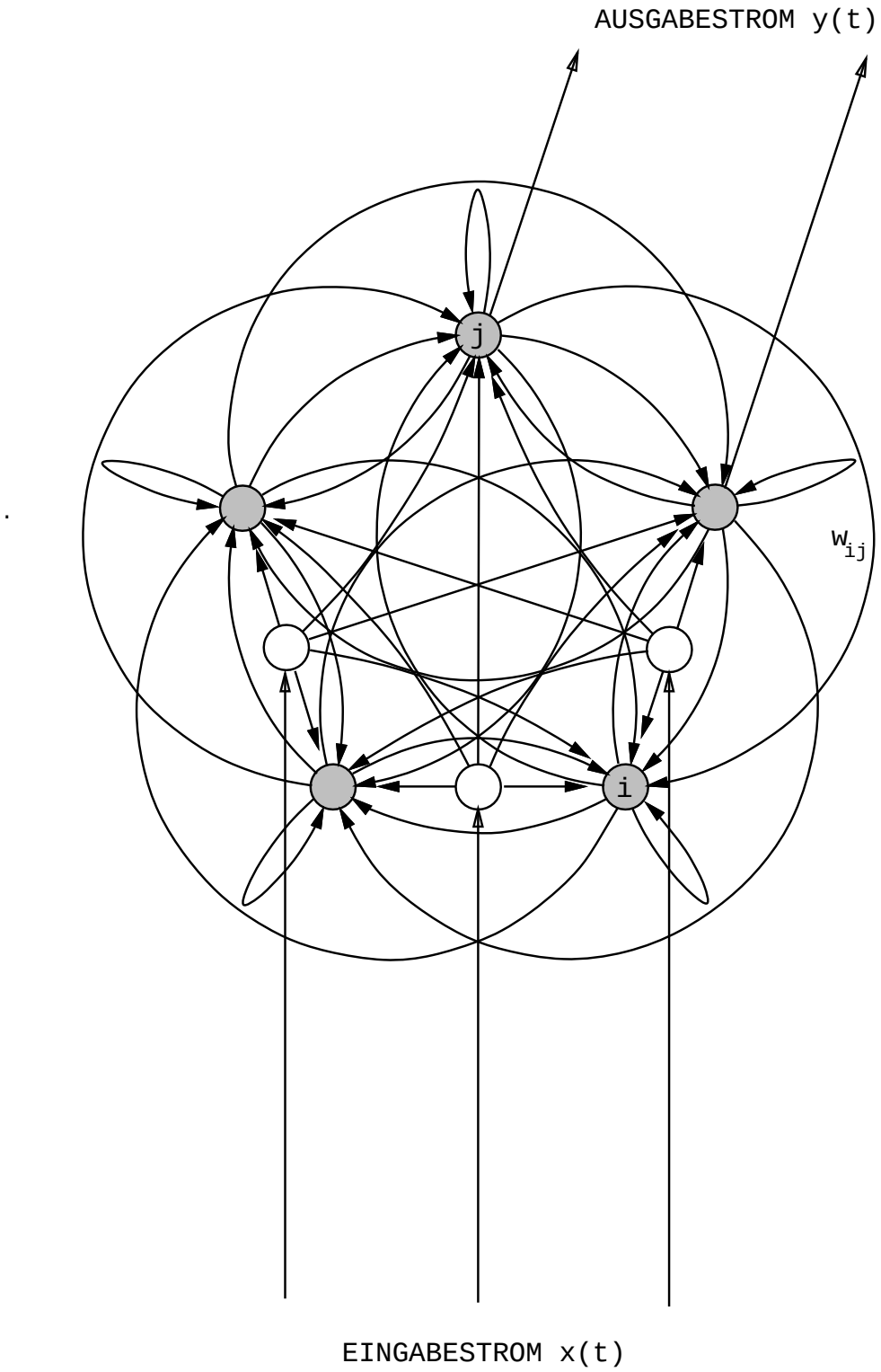

Abbildung 2.1: Ein rekurrentes Netz mit 3 Eingabeknoten (weiße Kreise) und 5 vollständig miteinander vernetzten internen Knoten (grau). 2 der internen Knoten dienen gleichzeitig als Ausgabeknoten. Das Gewicht der gerichteten Verbindung vom j-ten Knoten zum i-ten Knoten besitzt den Wert w_{ij}.

$$\forall t > 0: \quad net_k(t+1) = \sum_{l \in U \cup I} w_{kl}(t+1) x_l(t), \tag{2.1}$$

wobei f_k eine differenzierbare (gewöhnlich streng monoton wachsende) Aktivierungsfunktion bezeichnet, häufig die logistische Funktion

$$f(x) = \frac{1}{1+e^{-x}}. \tag{2.2}$$

Letztere läßt sich einfach ableiten:

$$f'(x) = f(x)(1-f(x)). \tag{2.3}$$

Turingäquivalenz. Obige zyklische Verbindungsstruktur weist zwar Ähnlichkeiten zur Topologie von Hopfieldnetzen [38], Boltzmannmaschinen [36], Schürmann-Netzen [123], und BP-Equilibriumsnetzen ([2], [81], [70], [71]) auf. Diese Ansätze sind allerdings *nur* für stationären Eingaben gedacht und für das Erlernen von Sequenzen ungeeignet – die Rekurrenz im Netz dient bei obigen Referenzen lediglich zur Ermittlung eines Aktivationsequilibriums. Die Aktivierungsdynamik (2.1) hingegen erlaubt zeitlich veränderliche Eingaben. Dieser wesentliche Unterschied bildet die Grundlage für ein System, das im Prinzip die Mächtigkeit einer Turingmaschine besitzt – und zwar im selben Sinne, in dem jeder konventionelle Rechner einer Turingmaschine gleichkommt. Dies ist unschwer einzusehen: Mit entsprechenden Gewichten kann man in einem hinreichend großen Netzwerk jede Boolesche Funktion und damit auch jede Kombination Boolescher Funktionen verdrahten. Damit läßt sich bei geeigneter Gewichtsbelegung die sequentielle Arbeitsweise eines herkömmlichen seriell arbeitenden Digitalrechners exakt emulieren – das einzige Hemmnis auf dem Weg zur 'wahren' Turingäquivalenz ist, wie stets, der endliche Speicherplatz.

Natürlich hat dieses Argument nur theoretische Bedeutung: Niemand wollte ernsthaft mittels eines zyklischen Netzes einen gewöhnlichen Digitalrechner nachbauen[1]. Vorteile erhofft man sich ja gerade von der Möglichkeit zumindest teilweise paralleler Informationsverarbeitung. Um unter den vielen Programmen, die ein zyklisches neuronales Netz ausführen könnte, bestimmte zweckmäßige Programme (= Gewichtsbelegungen) auszuzeichnen, definieren wir nun ein geeignetes

2.2 PERFORMANZMASS

$T(t)$ bezeichne die Menge von Indizes $k \in U$, für die ein von einem externen Lehrer definierter Zielwert $d_k(t)$ zum Zeitpunkt t existiert. Die Menge

[1] Dazu müßte man ja die meisten Gewichte gleich Null setzen und ginge dabei der potentiell ausschöpfbaren Möglichkeiten massiv paralleler Algorithmen verlustig.

aller dieser Zielwerte zu allen diskreten Zeitpunkten $0 < t \leq s$ der Eingabesequenz dient zur Definition des Performanzmaßes als die Summe aller zu verschiedenen Zeitpunkten auftretenden Fehler (den Abweichungen der tatsächlichen Ausgaben des Netzes von den Zielwerten). Wir schreiben:

$$e_k(t) = d_k(t) - x_k(t), \quad falls\ k \in T(t),$$

$$e_k(t) = 0 \quad sonst,$$

$$E(t) = \frac{1}{2} \sum_{k \in U} (e_k(t))^2, \quad E^{total}(t', t) = \sum_{\tau = t'+1}^{t} E(\tau). \tag{2.4}$$

Bei einer Trainingssequenz mit s Zeitschritten wollen wir durch einen geeigneten Lernalgorithmus

$$E^{total}(0, s) \tag{2.5}$$

mimimieren. Man beachte, daß die Minimierung von $E^{total}(0, s)$ im allgemeinen die zeitweise Speicherung von vergangenen Ereignissen erfordert, und daß i.a. nicht von vornherein bekannt ist, für welche Zeitdauern das Netz vergangene Ereignisse mit Hilfe seiner zyklischen Verbindungen intern repräsentieren muß, um seine Aufgabe lösen zu können.

Die drei folgenden Lernalgorithmen beruhen auf dem Prinzip des einfachen Gradientenabstiegs. Die Trainingssequenz wird dabei wiederholt präsentiert. Nach jeder Präsentation bestimmt man für jedes Gewicht w_{ij} eine Gewichtsänderung

$$\triangle w_{ij}(s) = -\alpha \frac{\partial E^{total}(0, s)}{\partial w_{ij}}, \tag{2.6}$$

wobei α eine positive Konstante (die sogenannte Lernrate) bezeichnet. Die Anwendung gradientenbasierter Lernalgorithmen kann im asymptotischen Fall (bei gegen Null gehendem *alpha*) höchstens zur Verbesserung der gegenwärtigen Gewichtsmatrix führen – nicht zur Verschlechterung. In praktischen Anwendungen (siehe Abschnitt 2.6) erweisen sich oft Lernraten in der Größenordnung von 0.01 bis 1 als brauchbar.

Wie schon verschiedentlich erwähnt, kennt die Numerik trickreichere und häufig effizientere Standardverfahren zur Ausnützung der einmal gewonnenen Gradienteninformation (z.B. konjugierte Gradientenverfahren, *'line search'*, Methoden zweiter Ordnung, effiziente Approximationen von Methoden zweiter Ordnung, etc., siehe e.g. [24][89][67][127]). Derartige Modifizierungen sind allerdings orthogonal zum Ziel dieser Arbeit – wir werden daher in den Experimenten stets einfachen Gradientenabstieg verwenden. Auch zur Behandlung lokaler Minima der Zielfunktion machen wir nur von

der einfachsten Lösung Gebrauch, welche darin besteht, ausgehend von verschiedenen zufällig gewählten Gewichtsmatrizen solange ein wiederholtes, lokales, gradientenbasiertes Suchen im Raum der möglichen Gewichtsmatrizen durchzuführen, bis ein Programm mit zufriedenstellender Performanz gefunden worden ist.

In der Regel hat man es mit vielen Trainingssequenzen zu tun – in diesem Fall ergibt sich die Gewichtsänderung für jedes Gewicht bei jeder Iteration des Gradientenabstiegsprozesses zur Summe der Beiträge, die man durch einen der nachfolgenden gradientenbasierten Algorithmen für jede einzelne Trainingssequenz erhält. Gerechtfertigt ist dies durch den Hinweis auf die Tatsache, daß der Gradient der Summe aller Fehler gleich der Summe der entsprechenden Gradienten ist.

2.3 ERSTE METHODE: *'BACK-PROPAGATION THROUGH TIME' (BPTT)*

Die Grundidee des folgenden Verfahrens geht auf Minsky und Papert [56] zurück, die darauf hinwiesen, daß die Aktivationsausbreitungsphase eines zyklischen dynamischen Netzes durch die Aktivationsausbreitungsphase eines wesentlich größeren azyklischen statischen Netzes beschrieben werden kann.

Für jeden Zeitschritt der Aktivationsausbreitungsphase, die ein rekurrentes Netzwerk durchläuft, legt man eine neue Kopie aller Knoten samt ihren gegenwärtigen Aktivationen an. Die Topologie des zu konstruierenden azyklischen Netzes ergibt sich dadurch, daß jede Verbindung des ursprünglichen Netzes in eine 'virtuelle' Verbindung transformiert wird, die in dem azyklischen Netz die Kopien der entsprechenden Knoten zu aufeinanderfolgenden Zeitschritten verbindet [85][144] (siehe hierzu Abbildung 2.2).

Man kann sich die $w_{ij}(t)$ als Gewichte virtueller Verbindungen zur t-ten Lage eines azyklischen Netzes vorstellen. Die Kettenregel liefert uns einen iterativen Algorithmus zur Berechnung von (2.6). Wir schreiben zunächst

$$\delta_i(\tau) = -\frac{\partial E^{total}(0,s)}{\partial net_i(\tau)}.$$

$\delta_i(\tau)$ kann für alle $i \in U, 0 \leq \tau \leq s$ in einem einzigen Pass berechnet werden:

$$\delta_i(\tau) = f_i'(net_i(\tau))e_i(\tau) \quad falls \quad \tau = s$$

$$\delta_i(\tau) = f_i'(net_i(\tau))(e_i(\tau) + \sum_{l \in U} w_{li}\delta_l(\tau+1)) \quad falls \quad 1 \leq \tau < s.$$

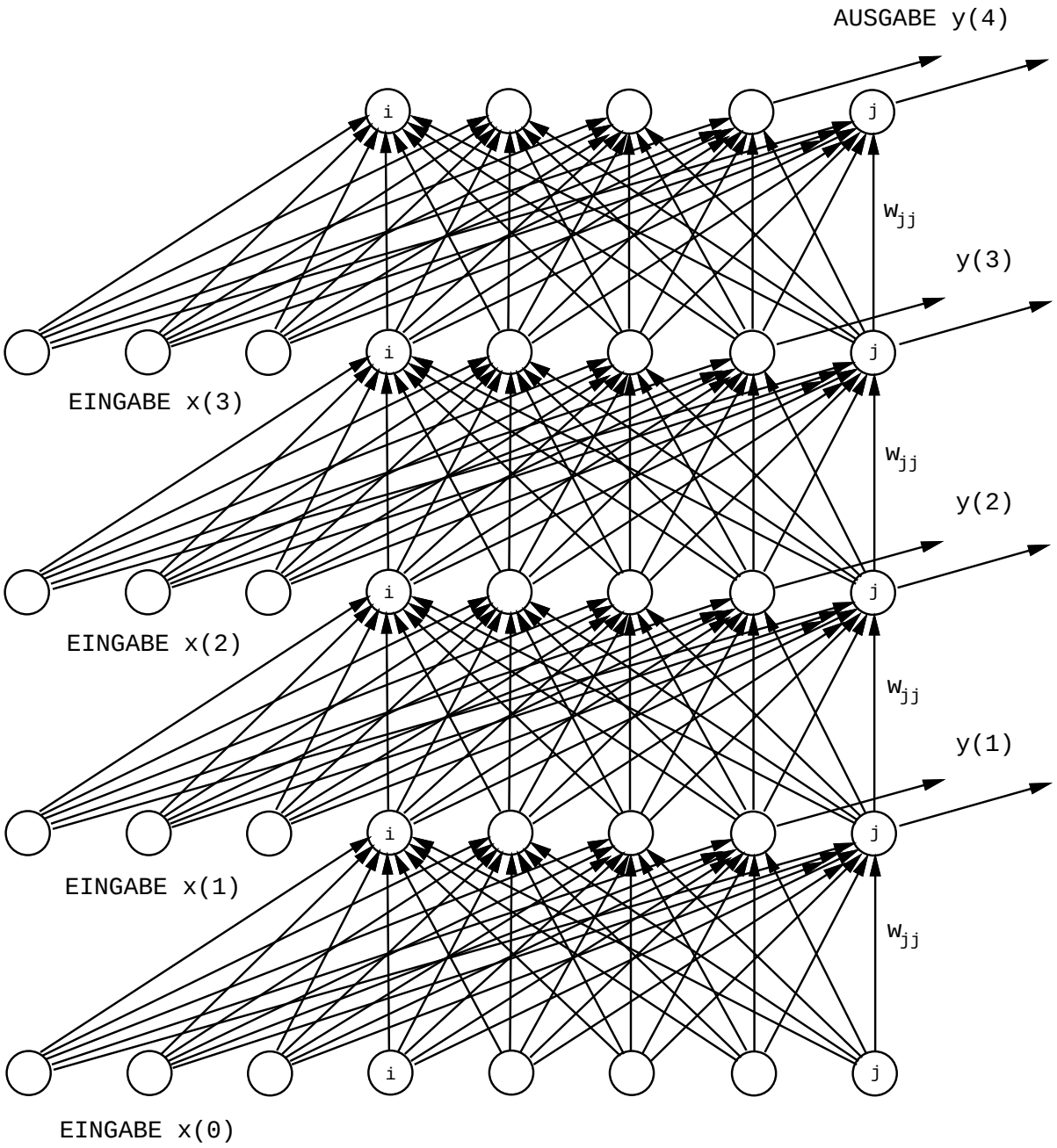

Abbildung 2.2: Eine 4 Zeitschritte umfassende Aktivationsausbreitungsphase des Netzes aus Abbildung 2.1 im 'zeitlich entfalteten' Zustand.

Die Berechnungskomplexität dieses Pass (die Anzahl der durchzuführenden Additionen und Multiplikationen) hat die Ordnung $O(sn^2)$.

Nun lassen sich die endgültigen Gewichtsänderungen bestimmen:

$$\triangle w_{ij}(s) = -\alpha \frac{\partial E^{total}(0,s)}{\partial w_{ij}} = -\sum_{\tau=1}^{s} \delta_i(\tau) x_j(\tau - 1).$$

Es ist offenbar nicht nötig, neben den zeitveränderlichen Aktivationen auch die Verbindungen und ihre Gewichte stets von neuem zu kopieren, da es sich ja für jeden Zeitschritt um dieselben Verbindungen handelt. Zweckmäßigerweise führt man für jeden Knoten des zyklischen Netzes einen Stapel ein, auf den die jeweiligen Aktivationen bei sukzessiven Zeitschritten der Aktivationsausbreitungsphase gestapelt werden. Bei der anschließenden Fehlerpropagierungsphase werden die Aktivationen nach dem *'last-in-first-out'*-Prinzip wieder vom Stapel geholt, um die Fehlersignale zu berechnen.

Komplexität. Egal, wie man den Algorithmus implementiert, in jedem Fall braucht man Speicherplatz unbekannter Größe für den Fall, daß kein Wissen um die Längen der Trainingssequenzen zur Verfügung gestellt wird. Wie wir jedoch später sehen werden, ist BPTT die Methode der Wahl, wenn von vornherein bekannt ist, daß alle Trainingssequenzen weniger als $O(n)$ Zeitschritte umfassen. Die Speicherkomplexität (die Anzahl der Variablen zur Speicherung von Gewichten und zeitlich variierenden Aktivationen) beträgt für einen Durchgang $O(sn + n^2)$. Die Zeitkomplexität (die Anzahl der erforderlichen Multiplikationen und Additionen) pro Durchgang beträgt $O(sn^2)$. Der Spitzenberechnungsaufwand pro Verbindung und Zeitschritt liegt bei $O(s)$. Das liegt daran, daß im jeweils letzten Zeitschritt eines Trainingsintervalls der gesamte Fehlerpropagierungs- und Gewichtsänderungsprozeß untergebracht werden muß.

'Abgeschnittenes BPTT'. Weiß man von vornherein, daß während der Trainingsphase zwischen Eingabeereignissen und korrelierten Fehlersignalen keine zeitlichen Verzögerungen der Länge $> h$ auftreten, so genügt es, die Fehlersignale zu jedem Zeitpunkt höchstens h Schritte 'in die Vergangenheit' zu senden, statt sie (wie bei BPTT vorgeschrieben) bis zum Anfang der Sequenz zurückzupropagieren [149].

2.4 ZWEITE METHODE: *'REAL-TIME RECURRENT LEARNING' (RTRL)*

Die Kettenregel erlaubt die Herleitung mehr als eines Algorithmus zur Berechnung ein und desselben Gradienten. Robinson und Fallside wiesen als

erste darauf hin, daß die Fehlerpropagierungsphase nicht unbedingt notwendig ist [79](siehe auch [78]). Es ist möglich, schon zur Laufzeit der Aktivierungsausbreitungsphase Information aufzusammeln, die sich bei der Beobachtung von späteren Fehlern sofort zur Berechnung eines Fehlergradienten heranziehen läßt. Der Vorteil dieses Verfahrens liegt darin, daß es bei fixer Netzgröße unabhängig von der Länge der zu erlernenden Sequenzen mit beschränktem Speicherplatz auskommt. Zu jedem Zeitpunkt einer Episode werden im wesentlichen die gleichen Operationen ausgeführt. Damit eignet sich die Methode für *'on-line'* Lernen, was ihr auch den Namen *'Real-Time Recurrent Learning'* (RTRL) eingetragen hat.

Wir lehnen uns wieder an Williams und Zipsers Beschreibung an [151] [150]. Verwandte Methoden finden sich in [68], [59], [3], [28] und [80].

Da der Gradient der Summe aller $E(t)$ gleich der Summe der entsprechenden Gradienten ist, genügt es, für jedes Gewicht w_{ij} den Wert

$$\triangle w_{ij}(t) = -\alpha \frac{\partial E(t)}{\partial w_{ij}} = -\alpha \sum_{k \in T(t)} (d_k(t) - x_k(t)) \frac{\partial x_k(t)}{\partial w_{ij}}$$

zu bestimmen. $\sum_t \triangle w_{ij}(t)$ liefert dann die Gesamtgewichtsänderung für w_{ij} am Ende des Trainingsintervalls. Nach der Kettenregel gilt für alle $k \in U$ und alle Zeitschritte t außer dem ersten

$$\frac{\partial x_k(t+1)}{\partial w_{ij}} = f'_k(net_k(t)) \left[\sum_{l \in U} w_{kl} \frac{\partial x_l(t)}{\partial w_{ij}} + \delta_{ik} x_j(t) \right].$$

δ_{ik} bezeichnet hier das Kroneckersche Delta und ist 1 für $i = k$ und 0 sonst.

Für den ersten Zeitschritt ist die entsprechende Ableitung 0. Daher lassen sich mit 0 initialisierte Variablen $p^k_{ij_{new}}$ und $p^k_{ij_{old}}$ zur inkrementellen Berechnung der $\frac{\partial x_k(t)}{\partial w_{ij}}$ einführen. Zu jedem Zeitpunkt t werden diese Variablen gemäß

$$\forall i,j,k: p^k_{ij_{new}} \leftarrow f'_k(net_k(t)) \left[\sum_{l \in U} w_{kl} p^k_{ij_{old}} + \delta_{ik} x_j(t) \right] \; ; \; \forall i,j,k: p^k_{ij_{old}} \leftarrow p^k_{ij_{new}}$$

aktualisiert und anschließend, wie oben ausgeführt, zur Berechnung von $\triangle w_{ij}(t)$ herangezogen.

On-Line versus *Off-Line*. Die *off-line* Version des Algorithmus spart sich die Ausführung der Gesamtgewichtsänderung (die Summe aller durch einzelne Zeitschritte verursachten Gewichtsänderungen) bis Abschluß der Präsentation aller Trainingssequenzen auf. Die entsprechende *on-line*-Version ändert die Gewichte zu jedem Zeitschritt jeder Trainingssequenz. Eine interessante Eigenschaft der *on-line*-Version ist, daß die Notwendigkeit zur Spezifikation von Trainingssequenzbegrenzungen entfällt (*'all episodes blend*

into each other' [150]). Die Lernrate α muß dabei natürlich klein genug sein, um kein Potential für Instabilitäten aufkommen zu lassen: Nur im asymptotischen Fall (bei gegen Null gehender Lernrate) vollzieht man exakten Gradientenabstieg in der Fehlersumme. Akzeptable Lernraten lassen sich gegenwärtig nur durch Experimente finden. Die später zu beschreibenden Experimente legen sowohl für die *on-line* Version als auch für die *off-line* Version für α Werte zwischen $\alpha = 0.02$ und $\alpha = 1.0$ nahe.

Komplexität. Die Speicherkomplexität von RTRL beträgt $O(n^3)$, die Berechnungskomplexität pro Zeitschritt beträgt $O(n^4)$.

Es sollte erwähnt werden, daß es auch RTRL-Versionen für nicht-diskrete Zeit gibt [68] [28].

Vor allem die Zeitkomplexität macht das Verfahren für großmaßstäbliche Anwendungen unbrauchbar. Im folgenden Abschnitt lernen wir eine Methode kennen, die exakt denselben Gradienten berechnet, bei gleichem Speicheraufwand aber n mal schneller ist.

2.5 DRITTE METHODE: EIN $O(n^3)$ VERFAHREN

Dieser Abschnitt präsentiert den ersten originären Beitrag der vorliegenden Arbeit. Wie oben gesehen, beträgt RTRLs Berechnungskomplexität[2] pro Zeitschritt $O(n^4)$. Im folgenden werden wir die Gradientenkalkulation so umarrangieren, daß die Speicherkomplexität des neuen Verfahrens dieselbe Ordnung wie die von RTRL aufweist, die Berechnungskomplexität aber in Proportion zur Zahl der Knoten im Netzwerk sinkt. Im Gegensatz zu der in [156] vorgestellten ad-hoc Methode bestimmt das neuartige Verfahren den exakten Gradienten, nicht etwa nur eine Approximation desselben [109].

Alle $O(n)$ Zeitschritte benötigt die neue Methode $O(n^4)$ Operationen, bei allen dazwischenliegenden Zeitschritten sind allerdings nur $O(n^2)$ Operationen erforderlich. Dies drückt die durchschnittliche Zeitkomplexität pro Zeitschritt auf $O(n^3)$.

Eine Trainingssequenz mit $s + 1$ Zeitschritten startet wieder zum Zeitschritt 0 und endet zum Zeitschritt s. Der nachfolgende Algorithmus ist von Interesse, falls $s >> n$ (sonst sei BPTT empfohlen).

Der Algorithmus[3] ist ein Hybrid zwischen BPTT und RTRL [109].

[2] Pineda hat einen weiteren Algorithmus für rekurrente Netze beschrieben, der, wie er selbst feststellt, *'einige der schlechtesten Eigenschaften beider Algorithmen (BPTT und RTRL) besitzt'* [72]. Sein Verfahren benötigt $\geq O(n^4)$ Speicherkomplexität und $\geq O(n^4)$ Zeitkomplexität, falls die Zahl der Zeitschritte n übersteigt.

[3] Kürzlich erfuhr ich, daß derselbe Algorithmus auch von Williams in einem nicht öffentlich angekündigten Report [148] beschrieben wurde, dessen Inhalt in [152] erschei-

Die folgende Beschreibung enthält sowohl die Herleitung als auch einige Kommentare zur Komplexität der einzelnen Schritte. Die wesentliche Idee ist: Zerlege die Gradientenberechnung in mehrerere Blöcke, von denen jeder $O(n)$ Zeitschritte umfaßt. Führe für jeden Block $n+1$ BPTT-ähnliche Berechnungsphasen durch. Eine davon dient zur Bestimmung gewisser Fehlergradienten, die restlichen n Phasen dienen zur Berechnung der partiellen Ableitungen der Netzeingaben jedes internen Knotens am Ende jedes Blocks. Führe schließlich $n+1$ RTRL-ähnliche Berechnungsphasen durch, um die Resultate der BPTT-Phasen mit den aus früher abgearbeiteten Blocks gewonnenen Resultaten zu verrechnen. Siehe Abbildung 2.3.

Für alle w_{ij} und für alle $l \in U, t \geq 0$ definieren wir

$$q_{ij}^l(t) = \frac{\partial net_l(t)}{\partial w_{ij}} = \sum_{\tau=1}^{t} \frac{\partial net_l(t)}{\partial w_{ij}(\tau)}.$$

Zu Beginn des Algorithmus setzen wir die Variable $t_0 \leftarrow 0$. t_0 bezeichnet den ersten Zeitschritt des gegenwärtigen Blocks. Man beachte, daß für alle in Frage kommenden l, w_{ij} gilt:

$$q_{ij}^l(0) = 0, \ \frac{\partial E^{total}(0,0)}{\partial w_{ij}} = 0.$$

Die Hauptschleife des Algorithmus umfaßt 5 Schritte.

1. SCHRITT: Setze $h \leftarrow O(n)$ (es sei empfohlen: $h \leftarrow n$).

Die Größe $\frac{\partial E^{total}(0,t_0)}{\partial w_{ij}}$ ist für alle w_{ij} bereits bekannt. Dasselbe gilt für $q_{ij}^l(t_0)$ für alle in Frage kommenden l, i, j. Gesucht ist eine effiziente Methode zur Bestimmung des Beitrags von $E^{total}(0, t_0+h)$ zur Änderung von w_{ij},

$$\triangle w_{ij}(t_0+h) = -\alpha \frac{\partial E^{total}(0,t_0+h)}{\partial w_{ij}} = -\alpha \sum_{\tau=1}^{t_0+h} \frac{\partial E^{total}(0,t_0+h)}{\partial w_{ij}(\tau)},$$

wobei α wieder eine positive konstante Lernrate bezeichnet.

2. SCHRITT: Führe eine Aktivationsausbreitungsphase für die Zeitschritte t_0 bis einschließlich t_0+h gemäß der in Gleichung (2.1) spezifizierten Aktivierungsdynamik durch. Stellt sich zur Laufzeit heraus, daß die gegenwärtige Trainingssequenz weniger als t_0+h Zeitschritte umfaßt (i.e., $h > s - t_0$), dann setze $h \leftarrow s - t_0$. Falls $h = 0$, terminiere den Algorithmus.

nen wird.

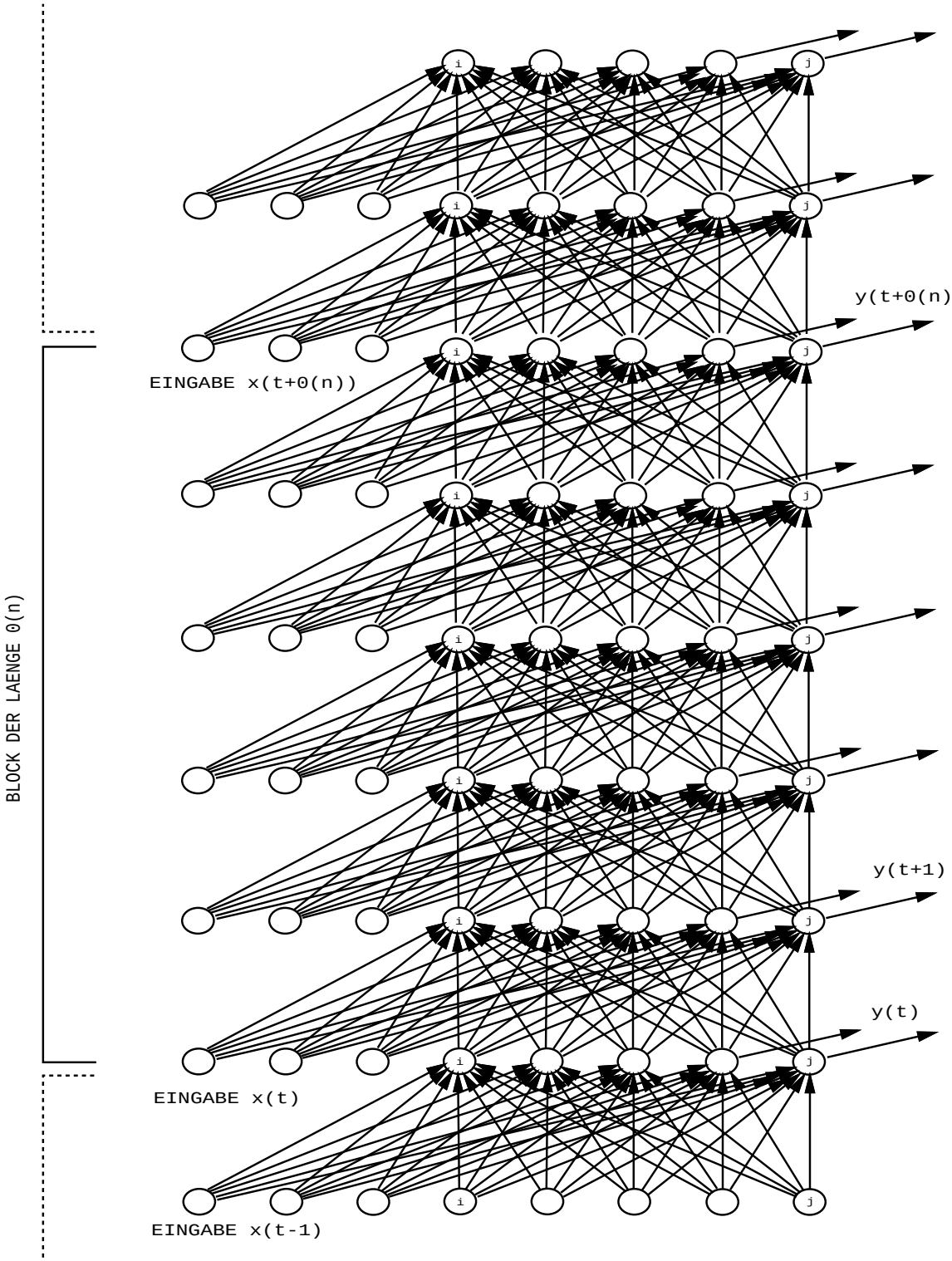

Abbildung 2.3: Zur Illustration des $O(n^3)$-Verfahrens. Die Aktivationsausbreitungsphase des rekurrenten Netzes aus Abbildung 2.1 wird in zeitliche Blöcke der Länge $O(n)$ zerlegt.

3. SCHRITT: Führe zur Berechnung von Fehlerableitungen folgende Kombination einer BPTT-ähnlichen Berechnungsphase mit einer RTRL-ähnlichen Berechnungsphase durch. Wir schreiben

$$\frac{\partial E^{total}(0,t_0+h)}{\partial w_{ij}} = \frac{\partial E^{total}(0,t_0)}{\partial w_{ij}} + \frac{\partial E^{total}(t_0,t_0+h)}{\partial w_{ij}}$$

$$= \frac{\partial E^{total}(0,t_0)}{\partial w_{ij}} + \sum_{\tau=1}^{t_0+h} \frac{\partial E^{total}(t_0,t_0+h)}{\partial w_{ij}(\tau)}$$

$$= \frac{\partial E^{total}(0,t_0)}{\partial w_{ij}} + \sum_{\tau=1}^{t_0} \frac{\partial E^{total}(t_0,t_0+h)}{\partial w_{ij}(\tau)} + \sum_{\tau=t_0+1}^{t_0+h} \frac{\partial E^{total}(t_0,t_0+h)}{\partial w_{ij}(\tau)} \quad (2.7)$$

Wir kennen bereits den ersten Term des Ausdrucks (2.7). Man konzentriere sich auf den dritten Term:

$$\sum_{\tau=t_0+1}^{t_0+h} \frac{\partial E^{total}(t_0,t_0+h)}{\partial w_{ij}(\tau)} = -\sum_{\tau=t_0+1}^{t_0+h} \delta_i(\tau) x_j(\tau-1),$$

wobei

$$\delta_i(\tau) = -\frac{\partial E^{total}(t_0,t_0+h)}{\partial net_i(\tau)}.$$

Für gegebenes t_0 kann $\delta_i(\tau)$ für alle $i \in U, t_0 \leq \tau \leq t_0+h$ mittels einer einzigen BPTT-Phase der Länge h und der Ordnung $O(hn^2)$ bestimmt werden:

$$\delta_i(\tau) = f_i'(net_i(\tau))e_i(\tau) \quad falls \quad \tau = t_0+h$$

$$\delta_i(\tau) = f_i'(net_i(\tau))(e_i(\tau) + \sum_{l\in U} w_{li}\delta_l(\tau+1)) \quad falls \quad t_0 \leq \tau < t_0+h$$

Was bleibt, ist die Berechnung des zweiten Terms in (2.7) für alle w_{ij}, was $O(n^3)$ Operationen kostet:

$$\sum_{\tau=1}^{t_0} \frac{\partial E^{total}(t_0,t_0+h)}{\partial w_{ij}(\tau)} = \sum_{\tau=1}^{t_0} \sum_{k\in U} \frac{\partial E^{total}(t_0,t_0+h)}{\partial net_k(t_0)} \frac{\partial net_k(t_0)}{\partial w_{ij}(\tau)}$$

$$= \sum_{k\in U} -\delta_k(t_0) \sum_{\tau=1}^{t_0} \frac{\partial net_k(t_0)}{\partial w_{ij}(\tau)} = -\sum_{k\in U} \delta_k(t_0) q_{ij}^k(t_0).$$

4. SCHRITT: Um $q_{ij}^l(t_0+h)$ für alle möglichen l,i,j zu berechnen, führe n BPTT/RTRL-Kombinationen (eine solche Kombination für jedes l) aus:

$$q_{ij}^l(t_0+h) = \frac{\partial net_l(t_0+h)}{\partial w_{ij}} = \sum_{\tau=1}^{t_0+h} \frac{\partial net_l(t_0+h)}{\partial w_{ij}(\tau)}$$

$$= \sum_{\tau=1}^{t_0} \frac{\partial net_l(t_0+h)}{\partial w_{ij}(\tau)} + \sum_{\tau=t_0+1}^{t_0+h} \frac{\partial net_l(t_0+h)}{\partial w_{ij}(\tau)}$$

$$= \sum_{\tau=1}^{t_0} \sum_{k\in U} \frac{\partial net_l(t_0+h)}{\partial net_k(t_0)} \frac{\partial net_k(t_0)}{\partial w_{ij}(\tau)} + \sum_{\tau=t_0+1}^{t_0+h} \frac{\partial net_l(t_0+h)}{\partial net_i(\tau)} \frac{\partial net_i(\tau)}{\partial w_{ij}(\tau)}$$

$$= \sum_{k\in U} \gamma_{lk}(t_0) h_{ij}^k(t_0) + \sum_{\tau=t_0+1}^{t_0+h} \gamma_{li}(\tau) x_j(\tau-1), \tag{2.8}$$

wobei

$$\gamma_{lk}(\tau) = \frac{\partial net_l(t_0+h)}{\partial net_k(\tau)}.$$

Für gegebenes t_0 und gegebenes $l \in U$ läßt sich $\gamma_{li}(\tau)$ für alle $i \in U, t_0 \leq \tau \leq t_0+h$ mittels einer einzigen BPTT-Phase der Länge h und der Ordnung $O(hn^2)$ bestimmen:

$$falls \;\; \tau = t_0+h: \;\; falls \;\; l=i, \;\; dann \;\; \gamma_{li}(\tau)=1, \;\; sonst \;\; \gamma_{li}(\tau)=0$$

$$falls \;\; t_0 \leq \tau < t_0+h: \;\; \gamma_{li}(\tau) = f_i'(net_i(\tau)) \sum_{a\in U} w_{ai}\gamma_{la}(\tau+1)$$

Für gegebenes l kostet die Berechnung von (2.8) für alle w_{ij} $O(n^3+hn^2)$ Operationen. Der 3. und der 4. SCHRITT brauchen zusammen also $(n+1)O(hn^2+n^3)$ Operationen, die sich allerdings über h Zeitschritte verteilen. Da wir zu Beginn $h = O(n)$ gesetzt haben, verstreuen sich demzufolge $O(n^4)$ Operationen über $O(n)$ Zeitschritte. *Dies zieht eine durchschnittliche Berechnungskomplexität der Ordnung $O(n^3)$ nach sich.*

Der 5. Schritt schließt die Hauptschleife des Algorithmus ab:

5. SCHRITT: Setze $t_0 \leftarrow t_0+h$ und springe zum 1. SCHRITT.

So wie er oben formuliert wurde, führt der Algorithmus bei jedem n-ten Zeitschritt zu einen Spitzenberechnungsaufwand von $O(n^4)$ Operationen. Nichts hält uns jedoch davon ab, diese $O(n^4)$ Berechnungen gleichmäßig über n Zeitschritte zu verteilen. Man könnte zum Beispiel zu jedem Zeitschritt des k-ten Blocks eine der n BPTT-Phasen des 4. SCHRITTs des $(k-1)$-ten Blocks durchführen. Dies würde auch den Spitzenberechnungsaufwand pro Zeitschritt auf $O(n^3)$ drücken.

Die *off-line* Version des Algorithmus führt die Gesamtgewichtsänderung nach Abschluß der Präsentation aller Trainingssequenzen aus.

Die korrespondierende *on-line*-Version ändert die Gewichte jeweils nach dem 4. SCHRITT (also jeweils nach der Abarbeitung eines Blockes der Zeitdauer n). Die im nächsten Abschnitt (sowie in den Arbeiten der anderen in diesem Kapitel aufgeführten Autoren) beschriebenen Experimente verwenden für alle Versionen aller drei oben betrachteten Algorithmen gewöhnlich Lernraten zwischen 0.01 und 1.0.

2.6 EXPERIMENTE

2.6.1 EXPERIMENTE MIT KLAMMERBALANCIERENDER TURINGMASCHINE

Das folgende Problem wurde erstmals von Williams und Zipser beschrieben [150]. Es geht darum, einem rekurrenten Netz beizubringen, das Zustandsübergangsdiagramm einer Turingmaschine zu simulieren, deren Aufgabe es ist, balancierte Zeichenreihen bestehend aus öffnenden und schließenden Klammern von unbalancierten Klammerfolgen zu unterscheiden.

Auf jedem Feld des Bandes der Turingmaschine kann eines von 4 möglichen Symbolen stehen: '(', ')', '*' und ' ' ('Blank'). Die entsprechenden Eingabevektoren für das Netzwerk sind $(0,1)^T$, $(1,0)^T$, $(1,1)^T$ und $(0,0)^T$. Der endliche Automat, der das Zustandsübergangsdiagramm der Turingmaschine beschreibt, ist durch Tabelle 2.1 definiert. Er besitzt 4 Zustände: 0, 1, 2 und 3. Aus einem gegebenen Zustand geht der Automat nach dem Lesen des Zeichens unter dem Lese- und Schreibkopf der Turingmaschine gemäß der Zustandsübergangstabelle in einen neuen Zustand über und schreibt gleichzeitig vor, ob der Lese- und Schreibkopf nach rechts, nach links, oder überhaupt nicht bewegt werden soll ('nix'). Die entsprechenden gewünschten Ausgabevektoren für zwei der Ausgabeknoten des Netzwerkes sind $(0,1)^T$, $(1,0)^T$ und $(0,0)^T$. Weiterhin wird eine von vier Aktionen ausgeführt: 1. Schreibe '*', 2. Zeige an, daß die Klammerstruktur balanciert ist, 3. Zeige an, daß die Klammerstruktur nicht balanciert ist, 4. Tue nichts ('nix'). Die entsprechenden gewünschten Ausgabevektoren für zwei zusätzliche Ausgabeknoten des Netzwerkes sind $(1,1)^T$, $(0,1)^T$, $(1,0)^T$ und $(0,0)^T$. Dem Netzwerk wird zu keinem Zeitpunkt Information über den gegenwärtigen Zustand der Turingmaschine geliefert. Es muß daher eigene geeignete interne Zustände 'erfinden'.

Während der Trainingsphase wurde sowohl zu Beginn als auch nach jeder Abarbeitung einer aus maximal 30 Klammern bestehenden Zeichenreihe eine neue Klammernfolge gemäß einer exponentiellen Bandlängenverteilung

Zustand	Eingabe	Neuer Zustand	Aktion	Bewegung
0	Blank	1	nix	nix
0	(	1	nix	nix
0	) (unmöglich)			
0	* (unmöglich)			
1	Blank	3	nix	links
1	(	1	nix	rechts
1	)	2	schreibe '*'	links
1	*	1	nix	rechts
2	Blank	0	unbalanziert	nix
2	(	1	schreibe '*'	rechts
2	)	2	nix	links
2	*	2	nix	links
3	Blank	0	balanziert	nix
3	(	0	unbalanziert	nix
3	)	3	nix	links
3	*	3	nix	links

Tabelle 2.1: *Zustandsübergangsdiagramm einer Turingmaschine, die balancierte Klammerfolgen von unbalancierten Klammerfolgen unterscheidet.*

generiert und die Anfangsposition des Kopfes der Turingmaschine auf das erste Zeichen des frisch generierten Bandes zurückgesetzt. Ein frisches Trainingsband wurde dabei wie folgt erstellt: Zunächst wurde auf zufällige Weise eine balancierte Klammernfolge generiert. In zwei Dritteln aller Fälle wurde diese durch zufällige 'Mutationen' modifiziert, wobei die Wahrscheinlichkeit dafür, daß k Klammern durch ihr Gegenstück ersetzt wurden, 0.5^k betrug.

Dem rekurrenten Netzwerk wurden jeweils zwei Zeitschritte zur Verarbeitung einer neuen Eingabe zugestanden. Zwischen dem Ende der Abarbeitung eines Bandes und dem Beginn der Abarbeitung des neuen Bandes wurde das Netz nicht von neuem initialisiert – Ereignisse, die während der Verarbeitung alter Bänder auftraten, konnten somit im Prinzip einen störenden Einfluß auf die Verarbeitung eines neuen Bandes nehmen. Die Trainingsphase wurde nach 100000 Turingmaschinenzyklen beendet. Eine anschließende Testphase galt als erfolgreich beendet (das Problem galt als gelernt), wenn das Netzwerk für die Zeitdauer von 50000 Turingmaschinenzyklen keinen Fehler machte, wobei ein Fehler durch eine 0.5 übersteigende Differenz zwischen der gewünschten und der tatsächlichen Ausgabe eines beliebigen Ausgabeknotens definiert war.

Bei 5 Testläufen mit RTRL lernte ein rekurrente Netz mit 15 versteckten Knoten und mit zwischen -1.0 und 1.0 initialisierten Gewichten bei einer Lernrate von 0.5 dabei stets, die Turingmaschine korrekt zu simulieren. Diese Resultate sind mit den in [150] berichteten Ergebnissen verträglich.

Bei 5 Testläufen mit dem neuartigen $O(n^3)$-Verfahren aus Abschnitt 2.5 lernte dasselbe rekurrente Netz bei gleicher Gewichtsinitialisierung ebenfalls in allen Fällen, die Aufgabe zu lösen. Dies ist nicht verwunderlich, da ja beide Verfahren den gleichen Gradienten berechnen (numerische Probleme durch die unterschiedliche Art der Gradientenberechnung traten nicht auf). Das neuartige Verfahren benötigte allerdings nur etwa ein Achtel der Rechenzeit des RTRL-Verfahrens. Natürlich stellt der Beschleunigungsfaktor 8 hier keine obere Schranke für die Überlegenheit der $O(n^3)$-Methode dar – je größer das Netzwerk, desto stärker fällt der Proportionalitätsfaktor $O(n)$ ins Gewicht.

2.6.2 EXPERIMENTE ZUM ERLERNEN REGULÄRER GRAMMATIKEN

Ein weiteres Beispiel für die Anwendung gradientenbasierter rekurrenter Netze ist das Erlernen regulärer Grammatiken aus einem kontinuierlichen Eingabestrom, der nach den Regeln der zu lernenden Grammatiken generiert wird (e.g. [129], [141]). Man betrachte hierzu Abbildung 2.4, die einen Automaten zur Bildung von Sätzen aus dem Sprachschatz einer sogenannten *'Reber-Grammatik'* darstellt. Abbildung 2.4 ist wie folgt zu le-

sen: Alle legalen Zeichenreihen beginnen mit dem Symbol a, welches in den Zustand 1 führt. Aus Zustand 1 kommt man mit dem Eingabesymbol a in den Zustand 2, dabei wird a zur bisher beobachteten Zeichenreihe hinzugefügt, etc.. Alle alternativen Pfade im Automaten werden mit gleicher Wahrscheinlichkeit eingeschlagen. Das Netz sieht als Eingabe zu jedem Zeitpunkt ein neues, mit Hilfe der durch den Automaten definierten Regeln produziertes, lokal durch einen 7-dimensionalen Binärvektor der Länge 1 repräsentiertes Zeichen: Das Symbol a wird durch $(1,0,0,0,0,0,0)^T$ repräsentiert, b durch $(0,1,0,0,0,0,0)^T$, c durch $(0,0,1,0,0,0,0)^T$, d durch $(0,0,0,1,0,0,0)^T$, e durch $(0,0,0,0,1,0,0)^T$, f durch $(0,0,0,0,0,1,0)^T$, und g durch $(0,0,0,0,0,0,1)^T$. Das Ziel des Netzes besteht zu jedem Zeitpunkt in der korrekten Vorhersage möglicher nächster Eingabesymbole. Eine Vorhersage gilt dabei als korrekt, wenn die Aktivationen der Ausgabeknoten, die den legalen Nachfolgern der bisher beobachteten Zeichenreihe entsprechen, größer als der Schwellwert 0.3 sind, und gleichzeitig die Aktivationen aller anderen Ausgabeknoten unterhalb dieses Schwellwerts liegen.

Den getesteten rekurrenten Netzwerke wurden wieder jeweils zwei Zeitschritte zur Verarbeitung einer neuen Eingabe zugestanden. Zwischen dem Ende eines Durchlaufs durch den Automaten und dem Beginn eines neuen Durchlaufs wurden die Netze (analog zum Experiment mit der Turingmaschine) nicht von neuem initialisiert.

Bei 10 Testläufen mit RTRL lernte ein rekurrentes Netz mit 9 Nichteingabeknoten, mit zwischen -0.5 und 0.5 initialisierten Gewichten und mit einer Lernrate von 0.5 innerhalb von 100000 Zeichenreihenpräsentationen stets, ausschließlich korrekte Vorhersagen zu produzieren. Dies verträgt sich mit den in [129] berichteten Ergebnissen[4].

Bei 10 Testläufen mit dem neuartigen $O(n^3)$-Verfahren aus Abschnitt 2.5 lernte dasselbe rekurrente Netz bei gleicher Parametereinstellung ebenfalls in allen Fällen, die Aufgabe zu lösen. Allerdings benötigte das neuartige Verfahren aber dank seiner geschickteren Gradientenkalkulation nur etwa ein Fünftel der Rechenzeit des RTRL-Verfahrens.

In weiteren Experimenten wurden unter anderem auch Simulationen des Verhaltens tatsächlich beobachteter biologischer Neuronennetzwerke mittels RTRL durchgeführt (z.B. [157]).

[4] Giles, Miller, Chen and Chen, Sun und Lee extrahieren aus trainierten Netzen (mit Verbindungen zweiter Ordnung) auch noch den zugehörigen minimalen endlichen Automaten [29].

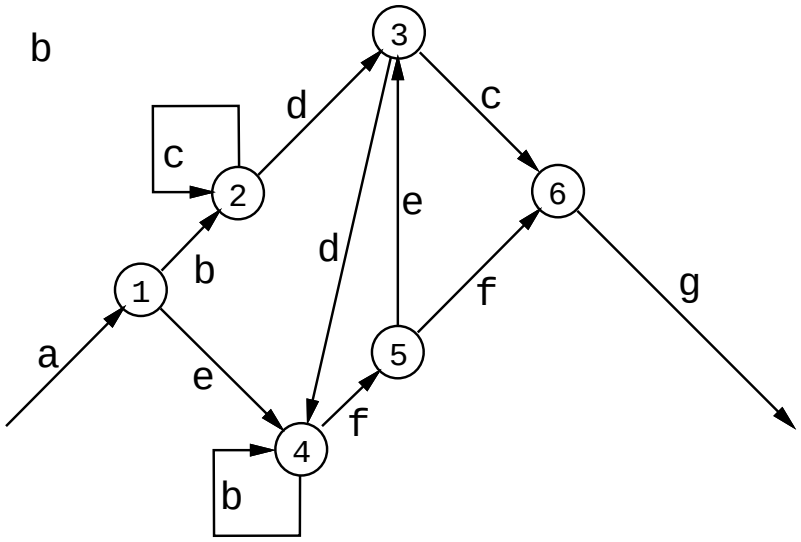

Abbildung 2.4: Ein Automat zur Generierung von Sätzen einer Reber-Grammatik. Alle legalen Zeichenreihen beginnen mit dem Symbol a, welches in den Zustand 1 führt. Aus Zustand 1 kommt man mit dem Eingabesymbol b in den Zustand 2, dabei wird a zur bisher beobachteten Zeichenreihe hinzugefügt, etc.. Alle alternativen Pfade im Automaten werden mit gleicher Wahrscheinlichkeit eingeschlagen.

2.6.3 SCHWACHSTELLEN REKURRENTER NETZE

Ich möchte die Aufmerksamkeit des Lesers nun zunächst auf die Tatsache lenken, daß beispielsweise die 'lernbare' Reber-Grammatik unter anderem auch sehr *kurze* (nur 5 Zeitschritte umfassende) Trainingssequenzen erlaubt. Damit ist gemeint, daß durch die zufällige Auswahl von Trainingssequenzen immer wieder welche vorkommen, bei denen die zeitlichen Abstände zwischen korrelierten Ereignissen so kurz werden, daß das rekurrente Netz lediglich ein paar Zeitschritte 'in die Vergangenheit blicken' muß, um den entsprechenden Zusammenhang zu entdecken. Bei einer solch kleinen Zahl von Zeitschritten ist es für einen gradientenbasierten Algorithmus gewöhnlich relativ einfach, die Gewichte dergestalt zu adjustieren, daß das Netz relevante Ereignisse solange abspeichern kann, bis sie etwas später für die korrekte Ausgabe zum richtigen Zeitpunkt benötigt werden. Hat das Netz einmal gelernt, sich Eingaben aus kurzen Trainingssequenzen in geeigneter Weise zu merken, fällt es ihm im allgemeinen leicht, auf lange Ereignissequenzen derselben Art (d.h. mit denselben relevanten Ereignissen, nun aber durch größere zeitliche Lücken getrennt) zu 'generalisieren'. Alle mir bekannten Experimente, so beeindruckend sie auf den ersten Blick auch erscheinen mögen, profitieren solcherart von kurzen Trainingssequenzen. Dies soll im folgenden detaillierter ausgeführt werden.

Bei gegebener Grammatik G mag die Aufgabe des Systems z.B. darin bestehen, einen Strom von Terminalsymbolen (ein Symbol pro Zeitschritt) zu beobachten und schließlich zu beurteilen, ob die beobachtete Zeichenreihe ein Satz von G ist oder nicht. Um das System zu trainieren, definiert man zweckmäßigerweise eine zweite Grammatik T, die sogenannte Trainingsgrammatik. T dient dazu, Beispiele und Gegenbeispiele für von G produzierte Zeichenreihen zu liefern. T definiert damit die Lernumgebung des lernenden Systems.

Im folgenden bezeichnen Großbuchstaben wie S und B Nichtterminalsymbole, während Kleinbuchstaben wie $a, x, b_1, \ldots, b_{100}$ Terminalzeichen denotieren. S wird stets als Startsymbol verwendet. Die folgende sehr einfache reguläre Grammatik G_1 produziert nur einen einzigen Satz. Damit ist sie von der Komplexität her wesentlich einfacher als die Reber-Grammatik, dennoch reicht sie aus, um eine fundamentale Schwierigkeit zu illustrieren:

$$G_1: \quad S \to aB, B \to b_1 b_2 b_3 \ldots b_{100}.$$

Definieren wir nun die Trainingsgrammatik T_1 als

$$T_1: \quad S \to aB, S \to xB, B \to b_1 b_2 b_3 \ldots b_{100}.$$

T_1 produziert nur 2 Traininssequenzen, nämlich $ab_1b_2b_3 \ldots b_{100}$ und $xb_1b_2b_3 \ldots b_{100}$. In Experimenten stellt sich nun heraus, daß es den gradientenbasierten Algorithmen für rekurrente Netze bei typischer zufälliger Gewichtsinitialisierung (im Bereich $-0.5, \ldots, 0.5$) schlechterdings unmöglich ist, innerhalb von 10^6 Trainingssequenzen zu lernen, die erste legale Zeichenreihe zu akzeptieren und die zweite illegale Zeichenreihe abzulehnen.

Zwar könnte man nun die Aufgabe *vereinfachen*, indem man G_1 durch G_2 und T_1 durch T_2 ersetzt:

$$G_2: \quad S \to aB, B \to b_1b_2b_3 \ldots b_{100}, B \to b_{100},$$

$$T_2: \quad S \to aB, S \to xB, B \to b_1b_2b_3 \ldots b_{100}, B \to b_{100}.$$

Nun könnten die konventionellen Algorithmen dank der kurzen Trainingssequenzen ab_{100} und xb_{100} herausfinden, daß das Auftreten von a oder x bedeutsam ist und mittels geeigneter Gewichte rekurrenter Verbindungen abgespeichert werden sollte. Aus den kurzen Trainingssequenzen könnten die Algorithmen auf 'schwierige' Sequenzen wie $xb_1b_2 \ldots b_{100}$ generalisieren.

Im allgemeinen darf man jedoch nicht davon ausgehen, daß die Umgebung hilfreiche kurze Trainingssequenzen zur Verfügung stellt. Die folgenden Experimente erzählen uns etwas über die Grenzen der praktischen Anwendbarkeit rekurrenter Netze.

2.6.4 EXPERIMENTE: PRAKTISCHE GRENZEN REKURRENTER NETZE

Wir konstruieren eine mit dem soeben erwähnten Problem verwandte Lernaufgabe, die auf den ersten Blick wieder wesentlich einfacher aussieht als das Problem des Lernens der Reber-Grammatik (siehe Kapitel 2). Es geht erneut um Sequenzklassifikation, allerdings weisen *alle* Trainingssequenzen relativ *lange* zeitliche Verzögerungen zwischen korrelierten Ereignissen auf.

Da das Problem repräsentativ für die bei langen Trainingssequenzen auftretenden Schwierigkeiten ist, nennen wir es die *Standardaufgabe*. Im folgenden werden wir verschiedene Architekturen immer wieder an dieser Aufgabe testen.

Standardaufgabe: Die Schwierigkeit der Standardaufgabe liegt in der Überbrückung einer m Zeitschritte umfassenden zeitlichen Verzögerung zwischen relevanten Ereignissen. Es gibt $m+2$ mögliche Eingabesymbole $a, x, b_1, b_2, \ldots, b_m$. Jedes Eingabesymbol wird lokal durch einen $(m+2)$-dimensionalen Bitvektor der Länge 1 eindeutig repräsentiert. Dem lernenden System wird zu jedem Zeitschritt ein Eingabesymbol präsentiert. Alle Nichteingabeknoten verwenden die logistische Aktivierungsfunktion $f(x) = \frac{1}{1+e^{-x}}$. Jeder

Nichteingabeknoten erhält (als modifizierbaren 'Schwellwert') eine zusätzliche Verbindung von einem Knoten, dessen Aktivation zu jedem Zeitpunkt gleich 1 ist. Alle Gewichte werden zwischen -0.2 und 0.2 initialisiert. Es gibt lediglich 2 mögliche Eingabesequenzen: $ab_1 \ldots b_m$ und $xb_1 \ldots b_m$. Diese werden während der Trainingsphase in zufälliger Ordnung angeboten. Zu jedem Zeitpunkt besteht ein Ziel darin, die nächste Eingabe vorherzusagen (man beachte, daß das erste Symbol jeder Sequenz aufgrund des zufälligen Auftretens von x und a i.a. nicht vorhersagbar ist). Das zweite (und bedeutend schwierigere) Ziel besteht in der Aktivierung eines speziellen Ausgabeknotens (des Zielknotens), wann immer die letzten $m+1$ verarbeiteten Eingabesymbole $a, b_1, \ldots, b_m$ waren. Der Zielknoten soll deaktiviert werden, wann immer der letzte sich über (m+1) Zeitschritte erstreckende Eingabesequenzprefix $xb_1 \ldots b_m$ war. Es sind also $m+2+1$ Ausgabeknoten vonnöten. Episodengrenzen werden *nicht* angezeigt: Die Trainingssequenzen werden sukzessive präsentiert, ohne daß Information über ihre Anfänge und Enden beigesteuert wird. So entsteht ein ununterbrochener kontinuierlicher Eingabestrom. Die Aufgabe gilt als gelöst, wenn die lokalen Fehler aller Ausgabeknoten (einschließlich des Zielknotens) stets unterhalb von 0.3 liegen (mit Ausnahme von Fehlern, die durch das prinzipiell unvorhersagbare Auftreten von a und x provoziert werden).

Konventionelle rekurrente Netze wurden an der Standardaufgabe bei verschiedenen Anzahlen versteckter Knoten, verschiedenen Lernraten, und verschiedenen m getestet. Die Experimente wurden von Josef Hochreiter (Diplomand an der TUM) durchgeführt. Fehlersignale wurden u. a. durch 'abgeschnittenes BPTT' (siehe Abschnitt 2.3) zu jedem Zeitpunkt $(m+1)$ Schritte zurückpropagiert (das ist die minimale Anzahl von Zeitschritten, die theoretisch für Verzögerungen der Länge m erforderlich ist). Man beachte, daß zusätzliche Rückpropagierungsschritte nur Verwirrung stiften würden, statt wohltätige Effekte zu zeitigen. Insofern darf man sagen, daß hier externes Wissen über die Natur der Standardaufgabe einfloß.

Es stellte sich heraus, daß es rekurrenten Netzen unter obigen Bedingungen schon bei $m = 20$ schlechterdings unmöglich ist, die Standardaufgabe bei vertretbarem Zeitaufwand zu lösen. Natürlich lernten die Netzwerke sehr schnell, die Symbole $b_1, \ldots, b_m$ vorherzusagen, doch die 21 Zeitschritte umfassende Verzögerung stellte ein praktisch unüberwindliches Hindernis dar (die Testläufe wurden nach 10^6 Trainingssequenzen unterbrochen). (Es sollte jedoch erwähnt werden, daß beschränkte Rechnerzeit keine systematischen Tests aller möglichen Parameterkombinationen erlaubte.) Man beachte, daß 20 Zeitschritte im Kontext der Sprachverarbeitung einem alles andere als langen Zeitraum entsprechen.

Bei $m = 4$ (dies enspricht einer 5 Zeitschritte umfassenden Verzögerung) erwiesen sich bei 4 Testläufen mit einer Lernrate von 1.0 die folgenden Zah-

len von Trainingssequenzen als erforderlich, um eine befriedigende Lösung zu erzielen:

$1,9 \times 10^6$; $0,9 \times 10^6$; $3,5 \times 10^6$; $0,25 \times 10^6$.

2.6.5 ZWEI PROBLEME DER OBIGEN ALGORITHMEN

Das erste Problem der Algorithmen aus den Abschnitten 2.3 bis 2.5 ist das folgende: Je mehr Zeit zwischen dem Auftreten eines Ereignisses und der Generierung eines durch dieses Ereignis bedingten Fehlersignals verstreicht, desto stärker wird das Fehlersignal i.a. auf seiner 'Reise in die Vergangenheit' gestreut, und desto unsignifikanter werden die durch das Fehlersignal hervorgerufenen Gewichtsänderungen (Hochreiter [37] beschreibt für aus nur wenigen Gewichten bestehende Netzwerke im Rahmen einer detaillierteren Analyse u.a. den exponentiellen Schwund der Stärke von Fehlersignalen in Abhängigkeit von der Anzahl der Rückpropagierungsschritte). [37] attackierte dieses Problem durch Einführung von speziellen Knoten mit auf sich selbst rückgekoppelten Verbindungen mit fixen, im voraus berechneten geeigneten Gewichten. Mozer erzielte einen vergleichbaren Effekt durch die Einführung spezieller Aktivierungsfunktionen [60][61].

Keiner dieser Ansätze löst allerdings das folgende schwerwiegende zweite Problem, welches darin besteht, daß die Aktivation jedes Eingabeknotens von den gradientenbasierten Algorithmen zu jedem Zeitpunkt in unspezifischer Weise gleich stark berücksichtigt wird, um das Problem des *'temporal credit assignment'*s [55] zu lösen. Keiner der bisher behandelten Algorithmen versucht in irgendeiner Weise, die von einem Eingabesignal übermittelte *Information* zu messen. Keiner der bekannten (oder bisher in dieser Arbeit behandelten) Algorithmen lernt, unsignifikante (redundante) Eingaben von potentiell signifikanten Eingaben zu unterscheiden und sich selektiv auf informationstragende Ereignisse zu konzentrieren, um so Zeit und Ressourcen zu sparen.

Dies liefert die Motivation für Kapitel 7, welches neuartige sequenzverarbeitende Architekturen samt zugehörigen Zielfunktionen beschreibt, denen es keine Schwierigkeiten bereitet, Aufgaben wie obige Standardaufgabe zu lösen, bei denen die im vorliegenden Kapitel angegebenen Verfahren in der Praxis scheitern. Das heißt allerdings nicht, daß die oben vorgestellten Algorithmen überflüssig wären: In der Tat werden wir die gradientenbasierten Netze des vorliegenden Kapitels in Kapitel 7 als Untermodule eines umfangreicheren Systems einsetzen.

2.7 SCHLUSSBEMERKUNGEN

Wir haben drei Methoden zur Berechnung des exakten Fehlergradienten für vollständig rekurrente dynamische Netze kennengelernt. Bei Trainingssequenzen, deren Länge von vornherein als klein im Verhältnis zur Anzahl der Knoten im Netz bekannt ist, empfiehlt sich die erste Methode (BPTT). In Fällen, wo man bisher die zweite Methode (RTRL) bevorzugte, sollte man tatsächlich das neue dritte komplexere Verfahren einsetzen, da es bei gleichem Speicheraufwand $O(n)$ mal schneller ist.

Eine auffallende Eigenschaft der neuartigen dritten Methode ist, daß sie zwei Größen miteinander in sinnvolle Beziehung setzt, die auf den ersten Blick nichts miteinander zu tun haben: Die Anzahl der Knoten im Netzwerk und die Anzahl der Zeitschritte, die die Aktivationsausbreitungsphase eines Blocks umfaßt.

Experimente zeigten, daß rekurrente gradientenbasierte Netze sich zwar zur Lösung interessanter Probleme eignen, daß sie aber trotz ihrer theoretischen Mächtigkeit im praktischen Einsatz in Schwierigkeiten geraten können, wenn lange Zeitverzögerungen zwischen korrelierten Ereignissen zu überbrücken sind. Kapitel 7 wird zeigen, wie durch die Einführung zusätzlicher Performanzmaße für *unüberwachtes* Lernen die praktische Lösung des durch die allgemeine Zielfunktion (2.5) definierten Problems in bestimmten Fällen gewaltig erleichtert werden kann.

Kapitel 3

DYNAMISCHE VERBINDUNGEN

Im letzten Kapitel haben wir vollständig rückgekoppelte Netze sowie eine allgemeine Zielfunktion für überwachtes Lernen in dynamischen Umgebungen betrachtet. Stellen bei gleichbleibender Zielfunktion rückgekoppelte Netze die einzige Architekturklasse dar, die es erlaubt, über simple Musterassoziation hinausgehendes sequentielles Verhalten zu implementieren? Der vorliegende Beitrag zeigt, daß dem keineswegs so ist. Tatsächlich bietet jede differenzierbare Gedächtnisstruktur[1] die Möglichkeit, mittels der Kettenregel gradientenbasierte Lernalgorithmen für nichtstationäre Umgebungen herzuleiten.

Ein Beispiel dafür liefern die wohl erstmals von von der Malsburg vorgeschlagenen 'dynamischen Verbindungen' (e.g. [138]). Bei diesen handelt es sich um Kanten, deren Gewichte (= Synapsenstärken) sich innerhalb kürzester Zeit von Grund auf ändern können. Solche *'schnellen Gewichte'* stehen im Kontrast zu den in praktisch allen Netzwerkmodellen verwendeten 'langsamen Gewichten', die sich nur durch wiederholte Trainingseinflüsse signifikant ändern und keine Kurzzeitspeicherfunktion übernehmen können.

Gerade so wie die rekurrenten Verbindungen des letzten Kapitels erlauben schnelle Gewichte dank ihres Potentials zur Implementierung zeitüberbrückender Speicherfunktionen dynamische Programmabläufe. Es gibt we-

[1] Mit dem Ausdruck 'differenzierbare Gedächtnisstruktur' ist ein Speicher gemeint, dessen zeitveränderliche Inhalte bezüglich aller Parameter, die das Systemverhalten steuern, differenzierbar sind. Gleichzeitig sollte das Performanzmaß bezüglich der Gedächtnisinhalte differenzierbar sein.

nigstens drei Gründe, zu versuchen, Lernalgorithmen für dynamische Verbindungen abzuleiten.

Zum ersten gibt es in nahezu allen in der Literatur beschriebenen KNN wesentlich mehr Verbindungen als Knoten. Verwendet man also dynamische Verbindungen statt Knoten mit Rückkopplung zur Speicherung von Ereignissen, so gewinnt man an Speichereffizienz.

Zum zweiten scheinen dynamische Verbindungen ein ideales Mittel zur Implementierung temporärer Variablenbindungen in KNN darzustellen. Wie implementiert man eine temporäre Bindung? Naheliegenderweise durch einen Zeiger vom Variablennamen auf den gegenwärtigen Inhalt. Genau dafür sind schnelle Gewichte wie geschaffen[2].

Zum dritten sind gewisse experimentell an biologischen Organismen gewonnene Resultate konsistent mit der Modellvorstellung dynamischer Verbindungen. [1] zeigt beispielsweise, daß sich die dynamische effektive Konnektivität zweier Neuronen innerhalb weniger 10 msec drastisch ändern kann.

Im folgenden wird erstmalig ein gradientenbasierter Lernalgorithmus für ein auf schnellen Gewichten basierendes dynamisches System abgeleitet. Wir folgen unserem üblichen Schema: Nach der Definition der Architektur und der Netzwerkdynamik leiten wir für dasselbe Performanzmaß, das schon im letzten Kapitel verwendet wurde, durch massive Anwendung der Kettenregel den entsprechenden Lernalgorithmus her. Einige Experimente (u.a. zur temporären Variablenbindung) illustrieren schließlich die Arbeitsweise des Verfahrens.

3.1 ARCHITEKTUR UND NETZWERKDYNAMIK

Als Ereignissspeicher verwenden wir ein azyklisches Netzwerk F mit einer Menge 'schneller' Gewichtsvariablen W_F. Wie diese Variablen dynamisch zu setzen sind, wird weiter unten ausgeführt. F's Eingabe zum Zeitpunkt t ist der reelle Vektor $x(t)$, F's m-dimensionale Ausgabe ist der reelle Vektor $y(t)$. Abhängig von W_F's zeitlich veränderlichem Zustand wird F ein und dieselbe Eingabe zu verschiedenen Zeitpunkten unterschiedlich verarbeiten – diese Tatsache nützen wir aus, um ein von einem zusätzlichen adaptiven Mechanismus gesteuertes Kurzzeitgedächtnis zu konstruieren.

Da wir, unserem allgemeinen Schema folgend, später mittels der Kettenregel einen Lernalgorithmus zum Setzen der dynamischen Gewichte her-

[2] Hier sollte vielleicht angemerkt werden, daß neuronalen Netzen bis vor kurzem gelegentlich vorgeworfen wurde, keinen Mechanismus zur Variablenbindung anzubieten.

leiten wollen, fordern wir, daß dieser zusätzliche Mechanismus dergestalt durch eine parametrisierte Klasse von Funktionen dargestellt wird, daß die schnellen Gewichte in differenzierbarer Weise von den Parametern abhängen.

Es liegt nahe, zu diesem Zweck wiederum ein konventionelles mehrlagiges azyklisches Netzwerk S mit einer Menge zufällig initialisierter Gewichtsvariablen W_S zu verwenden. S' Eingabe zum Zeitpunkt t ist ebenfalls der reelle Vektor $x(t)$. Wie im folgenden zu sehen sein wird, wird S' Ausgabe in sofortige Gewichts*änderungen* für W_F umgemünzt. S dient damit als Kontrollinstanz für das 'schnelle' F-Netzwerk – S' Ausgabe entscheidet darüber, welche Ereignisse in welcher Form in F's schnellen Gewichten gespeichert werden.

Zur Initialisierung führen wir zu Beginn jeder Trainingssequenz einen 'nullten' Zeitschritt ein. Zum Zeitpunkt 0 wird jede Gewichtsvariable $w_{ba} \in W_F$ einer gerichteten Verbindung vom Knoten a zum Knoten b gleich $\Box w_{ba}(0)$ gesetzt. $\Box w_{ba}(0)$ ist eine Funktion von S' Ausgabeknoten – weiter unten werden wir zwei verschiedene Implementierungen dieser Funktion betrachten. Zum Zeitschritt $t > 0$ dient $w_{ba}(t-1)$ zur Berechnung von F's gegenwärtiger Ausgabe anhand der gewöhnlichen Aktivationsausbreitungsregeln für BP-Netzwerke (e.g. [143]): Die Aktivation $o_k(t)$ des k-ten Knoten in der r-ten Lage von F (mit $r > 1$) wird gemäß

$$net_k(t) = \sum_{l \in \ Lagen \ <r} w_{kl}(t-1) o_l(t), \quad o_k(t) = f_k(net_k(t)) \tag{3.1}$$

berechnet, wobei f_k eine differenzierbare Aktivierungsfunktion bezeichnet (gewöhnlich die logistische, siehe (2.2)). Der Ausgabevektor der obersten Lage ist $y(t)$.

Nun wird jede Gewichtsvariable $w_{ba} \in W_F$ gemäß

$$w_{ba}(t) = \sigma(w_{ba}(t-1), \Box w_{ba}(t)) \tag{3.2}$$

geändert, wobei σ bezüglich aller Parameter differenzierbar sein soll. $\Box w_{ba}(t)$ ist eine Funktion von S' Ausgabe und wird durch einen der beiden in den nächsten beiden Unterabschnitten beschriebenen Mechanismen berechnet. $\Box w_{ba}(t)$ stellt S' Beitrag zur Modifikation von w_{ba} zum Zeitpunkt t dar. Die Aktivation des k-ten Nichteingabeknoten in der r-ten Lage von S (mit $r > 1$) wird wiederum gemäß

$$net_k(t) = \sum_{l \in \ Lagen \ <r} w_{kl}(t-1) o_l(t), \quad o_k(t) = f_k(net_k(t)) \tag{3.3}$$

berechnet (f_k ist erneut eine differenzierbare Aktivierungsfunktion).

Solange F's Gewichte nicht explizit durch S modifiziert werden, bleiben sie im wesentlichen invariant. F's gegenwärtige Eingabe läßt sich als

eine Menge von *Adressen* einer Menge von Variablen interpretieren, F's gegenwärtige Ausgabe kann als der derzeitige *Inhalt* dieser Variablen angsehen werden. Bindungen lassen sich in natürlicher Weise durch temporäre Konnektivitätsmuster (statt durch temporäre Aktivationsmuster) etablieren.

Gleichung (3.2) ist im wesentlichen identisch mit Möllers und Thruns Gleichung (1) in [57]. Im Gegensatz zu [57] werden wir jedoch im folgenden einen exakten Gradientenabstiegsalgorithmus für zeitveränderliche Ein- und Ausgaben herleiten.

Wie genau lassen sich S' Ausgaben nun in F's Gewichtsänderungen übersetzen?

3.1.1 ARCHITEKTUR 1

Die vielleicht einfachste Schnittstelle zwischen S and F erhält man durch Bereitstellung eines Ausgabeknotens $s_{ab} \in S$ für jede Gewichtsvariable $w_{ba} \in W_F$, wobei

$$\Box w_{ba}(t) := s_{ab}(t). \tag{3.4}$$

$s_{ab}(t)$ bezeichnet hier die Aktivation des Ausgabeknotens zur Zeit $t \geq 0$. Siehe Abbildung 3.1.

Ein Nachteil dieser Architektur besteht darin, daß die Zahl der Ausgabeknoten in S proportional zur Anzahl der schnellen Gewichte in F wächst. Eine Alternative bietet

3.1.2 ARCHITEKTUR 2

Stelle für jeden Knoten in F, dem wenigstens eine Verbindung mit einem schnellen Gewicht entspringt, einen Ausgabeknoten in S bereit. Nenne die Menge derartiger Ausgabeknoten *VON*. Spendiere einen weiteren Ausgabeknoten in S für jeden Knoten in F, zu dem wenigstens eine 'schnellgewichtige' Verbindung führt. Nenne die Menge dieser Ausgabeknoten *ZU*. Für jede Gewichtsvariable $w_{ba} \in W_F$ haben wir jetzt einen Knoten $s_a \in VON$ und einen Knoten $s_b \in ZU$. Definiere für den Zeitpunkt t

$$\Box w_{ba}(t) := g(s_a(t), s_b(t)),$$

wobei g bezüglich all seiner Parameter differenzierbar sein muß. Wir konzentrieren uns auf ein repräsentatives Beispiel, nämlich den Fall, daß g den Multiplikationsoperator darstellt:

$$\Box w_{ba}(t) := s_a(t)s_b(t). \tag{3.5}$$

Dies bedeutet, daß F's schnelle Gewichte von S in Hebb-ähnlicher Manier [33] manipuliert werden. Mit Architektur 2 benötigt S weniger als doppelt

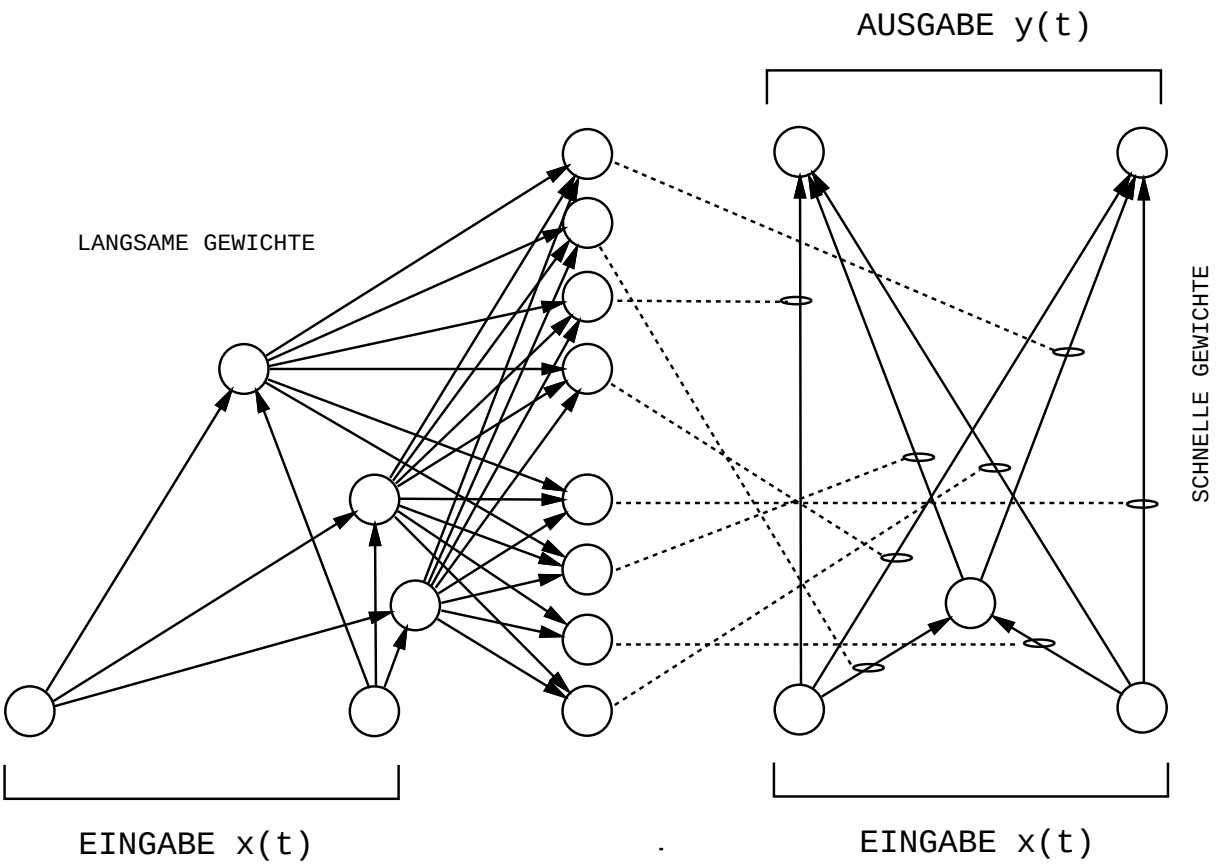

Abbildung 3.1: Gewichtsänderungen eines Netzwerkes mit dynamischen Verbindungen (rechts, mit einem versteckten Knoten) werden direkt durch die Ausgaben des 'langsamen' Netzes (links) gesteuert.

so viele Ausgabeknoten, als Knoten in F vorhanden sind. Siehe Abbildung 3.2.

Man kann sich von der *VON/ZU* Architektur folgendes Bild machen: S kreiert befristete Assoziationen, indem es zwei Parameter an das Kurzzeitgedächtnis weitergibt. Der erste Parameter ist ein durch *VON* übermitteltes Aktivationsmuster, welches den *Schlüssel* zu einem temporären Assoziationspaar darstellt, während der zweite Parameter einen durch *ZU* übermittelten *Eintrag* repräsentiert. Man beachte, daß sowohl der Schlüssel als auch der Eintrag versteckte Knoten ansprechen können.

3.2 PERFORMANZMASS

Wie bei den rekurrenten Netzen des zweiten Kapitels wollen wir auch mit der neuen Architektur Fehlertrajektorien minimieren. Der Gradient des Fehlers über alle Trainingssequenzen ist gleich der Summe der Fehlergradienten. Daher interessiert uns wieder nur der während einer Sequenz betrachtete Fehler

$$\bar{E} = \sum_t E(t),$$

wobei

$$E(t) = \frac{1}{2} \sum_i e_i^2(t),$$

$$e_i(t) = d_i(t) - y_i(t).$$

$d(t)$ ist dabei erneut der von einem externen Lehrer vorgegebene Ausgabevektor.

3.3 ABLEITUNG DER ALGORITHMEN

Für alle Gewichte $w_{ji} \in W_S$ (vom Knoten i zum Knoten j) interessiert uns das Inkrement

$$\triangle w_{ji} = -\eta \frac{\partial \bar{E}}{\partial w_{ji}} = -\eta \sum_{t>0} \frac{\partial E(t)}{\partial w_{ji}} = -\eta \sum_{t>0} \sum_{w_{ba} \in W_F} \frac{\partial E(t)}{\partial w_{ba}(t-1)} \frac{\partial w_{ba}(t-1)}{\partial w_{ji}}. \tag{3.6}$$

η bezeichnet dabei eine konstante positive Lernrate.

Zu jedem Zeitschritt $t > 0$ läßt sich der Faktor

$$\gamma_{ab}(t) = \frac{\partial E(t)}{\partial w_{ba}(t-1)}$$

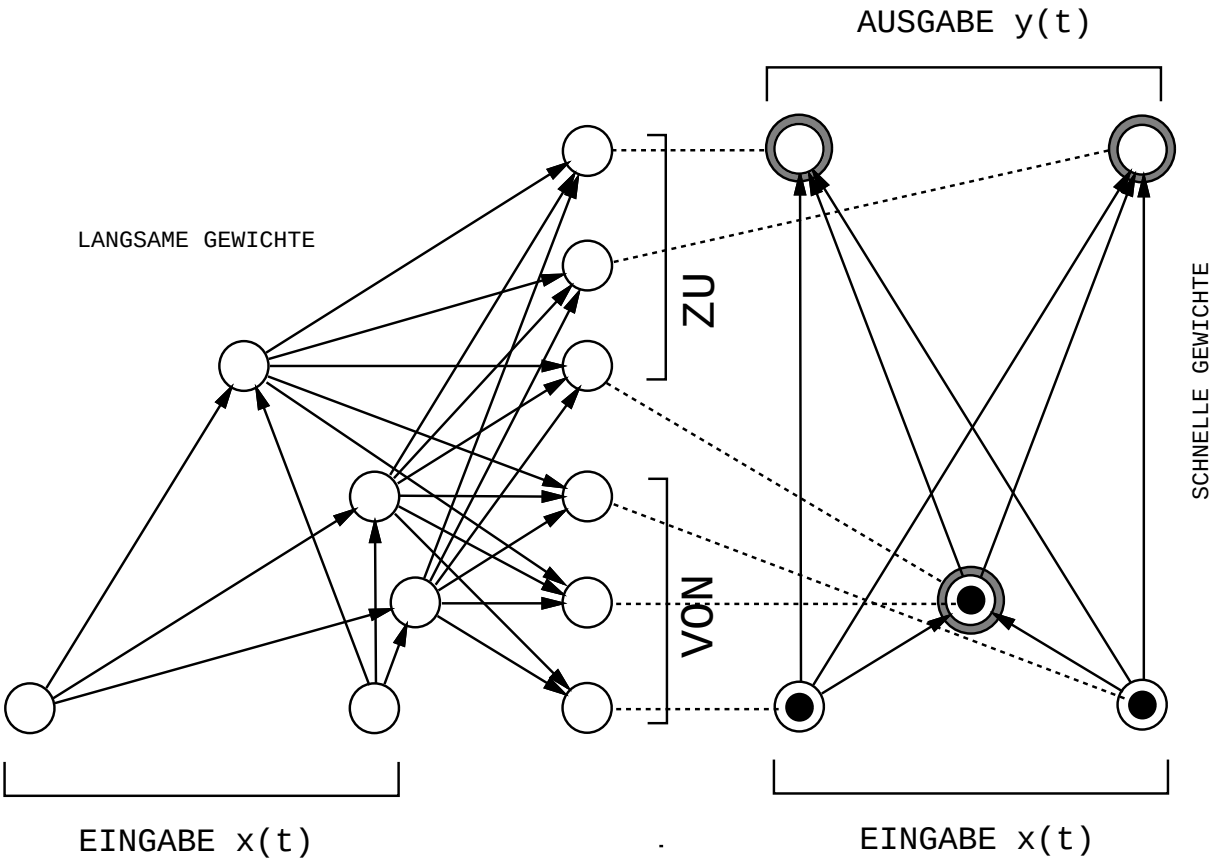

Abbildung 3.2: Gewichtsänderungen des 'schnellen' Netzwerkes ergeben sich durch die vom langsamen Netz in seinen beiden Ausgabefeldern VON (für den 'Schlüssel') und ZU (für den Inhalt) vorgeschriebene Assoziation.

durch konventionelles BP bestimmen (siehe Kapitel 1). Wir schreiben für Knoten k in F:

$$\delta_k(t) = \frac{\partial E(t)}{\partial net_k(t)}.$$

Falls b einen Ausgabeknoten bezeichnet, so ist

$$\frac{\partial E(t)}{\partial w_{ba}(t-1)} = \delta_b(t) o_a(t) \quad mit \quad \delta_b(t) = f_b'(net_b(t)) e_b(t).$$

Steht b hingegen für einen versteckten Knoten der r-ten Lage von F, so gilt

$$\frac{\partial E(t)}{\partial w_{ba}(t-1)} = \delta_b(t) o_a(t) \quad mit \quad \delta_b(t) = f_b'(net_b(t)) \sum_{l \in \ Lagen \ > r} w_{lb} \delta_l(t).$$

Für $t > 0$ erhalten wir nun mit der Kettenregel die Rekursion

$$\frac{\partial w_{ba}(t)}{\partial w_{ji}} = \frac{\partial \sigma(w_{ba}(t-1), \Box w_{ba}(t))}{\partial w_{ba}(t-1)} \frac{\partial w_{ba}(t-1)}{\partial w_{ji}} +$$

$$+ \frac{\partial \sigma(w_{ba}(t-1), \Box w_{ba}(t))}{\partial \Box w_{ba}(t)} \frac{\partial \Box w_{ba}(t)}{\partial w_{ji}}.$$

Wir verwenden eine durch den RTRL-Algorithmus (siehe Kapitel 2) inspirierte Methode: Für jedes $w_{ba} \in W_F$ und jedes $w_{ji} \in W_S$ führen wir eine Variable p_{ij}^{ab} ein und initialisieren sie zu Beginn einer Trainingssequenz mit 0. p_{ij}^{ab} läßt sich zu jedem Zeitschritt $t > 0$ aktualisieren:

$$p_{ij}^{ab}(t) = \frac{\partial \sigma(w_{ba}(t-1), \Box w_{ba}(t))}{\partial w_{ba}(t-1)} p_{ij}^{ab}(t-1) +$$

$$+ \frac{\partial \sigma(w_{ba}(t-1), \Box w_{ba}(t))}{\partial \Box w_{ba}(t)} \frac{\partial \Box w_{ba}(t)}{\partial w_{ji}}. \tag{3.7}$$

$\frac{\partial \Box w_{ba}(t)}{\partial w_{ji}}$ hängt davon ab, welche der beiden Architekturen wir benutzen wollen. Eine geeignete BP-Prozedur liefert uns jedenfalls für jedes $w_{ba} \in W_F$ den Wert $\frac{\partial \Box w_{ba}(t)}{\partial w_{ji}}$ für alle $w_{ji} \in W_S$. Nach Aktualisierung der p_{ij}^{ab} Variablen wird (3.6) mittels der Formel

$$\frac{\partial E(t)}{\partial w_{ji}} = \sum_{w_{ba} \in W_F} \gamma_{ab}(t) p_{ij}^{ab}(t-1)$$

bestimmbar.

(3.4) und (3.5) unterscheiden sich lediglich in der Art und Weise, in der Fehlersignale für S' Ausgabeknoten berechnet werden: Wird Architektur 1

verwendet, benützen wir konventionelles BP zur Berechnung von $\frac{\partial s_{ab}(t)}{\partial w_{ji}}$ in (3.7). Dann schreiben wir für Knoten k in S:

$$\kappa_k(t) = \frac{\partial s_{ab}(t)}{\partial net_k(t)}.$$

Bezeichnet j einen Ausgabeknoten (mit $o_j = s_{ab}$), so gilt

$$\frac{\partial s_{ab}(t)}{\partial w_{ji}} = \kappa_k(t) o_i(t) \quad mit \quad \kappa_k(t) = f_j'(net_j(t))$$

Stellt j hingegen einen versteckten Knoten der r-ten Lage von S dar, so ist

$$\frac{\partial s_{ab}(t)}{\partial w_{ji}} = \kappa_k(t) o_i(t) \quad mit \quad \kappa_k(t) = f_j'(net_j(t)) \sum_{l \in \ Lagen \ > r} w_{lj} \kappa_l(t).$$

Bei Verwendung von Architektur 2 ist zu berücksichtigen, daß

$$\frac{\partial \Box w_{ba}(t)}{\partial w_{ji}} = s_b(t) \frac{\partial s_a(t)}{\partial w_{ji}} + s_a(t) \frac{\partial s_b(t)}{\partial w_{ji}}. \tag{3.8}$$

Mit BP berechnet man am besten $\frac{\partial s_a(t)}{\partial w_{ji}}$ für jeden Ausgabeknoten a und für alle w_{ji}. Die Resultate lassen sich in $\mid W_S \mid * \mid VON \cup ZU \mid$ Variablen unterbringen. Nun löst man Gleichung (3.7) in einem zweiten Durchgang.

Bei beiden Architekturen beträgt die Berechnungskomplexität pro Zeitschritt $O(\mid W_F \mid\mid W_S \mid)$.

3.3.1 *On-Line* VERSUS *Off-Line*

Analog zu den Algorithmen aus Kapitel 2 spart sich die *off-line* Version des Algorithmus die Ausführung der Gesamtgewichtsänderung für S (die Summe aller durch einzelne Zeitschritte verursachten Gewichtsänderungen) bis Abschluß der Präsentation aller Trainingssequenzen auf.

Die entsprechende *on-line*-Version ändert W_S zu jedem Zeitschritt jeder Trainingssequenz. Die interessante Eigenschaft der *on-line*-Version besteht wiederum darin, daß die Notwendigkeit zur Spezifikation von Trainingssequenzbegrenzungen entfällt. Die Lernrate η muß wieder klein genug sein, um Instabilitäten zu verhindern: Nur im asymptotischen Fall (bei gegen Null gehender Lernrate) vollzieht man exakten Gradientenabstieg im Performanzmaß. Akzeptable Lernraten lassen sich gegenwärtig nur durch Experimente finden. Die später zu beschreibenden Experimente legen für η Werte zwischen $\eta = 0.02$ und $\eta = 0.5$ nahe.

3.3.2 ALTERNATIVE VERWENDUNGEN DER KETTENREGEL

Statt der RTRL-inspirierten obigen Methode könnte man sich in Anlehnung an Kapitel 2 auch eines BPTT-inspirierten Verfahrens bedienen. In diesem Fall hält man sich bequemerweise einen Aktivationskeller für jeden Knoten in S. Auch die Grundidee des in Sektion 2.4 vorgestellten $O(n^3)$-Verfahren läßt sich nach entsprechender Modifikation auf die auf schnellen Gewichten basierenden Architekturen anwenden.

3.3.3 BESCHRÄNKUNGEN UND ERWEITERUNGEN

Sind sowohl F als auch S azyklisch, erlauben obige Architekturen nur eine eingeschränkte Klasse von Algorithmen. Ist nämlich σ eine sogenannte *'sum-and-squash'*-Funktion, dann lassen sich interessante zeitlich veränderliche Ausgaben nur in Antwort auf Eingabevariationen hervorrufen. Insbesondere kann kein autonomes dynamisches Verhalten (wie z.B. Oszillationen) auftreten, solange die Eingabe festgehalten wird.

Es stellt jedoch kein Problem dar, unsere dynamischen Verbindungen in ein System einzubetten, bei dem sowohl F als auch S rekurrent sind. Für die im nächsten Abschnitt beschriebenen Experimente sind beide beteiligten Netze *nichtrekurrent*, um zu demonstrieren, daß sogar azyklische Netze gewisse Aufgaben lösen können, deren Lösung bis vor kurzem rekurrenten Netzen vorbehalten schien.

3.4 EXPERIMENTE

In Zusammenarbeit mit Klaus Bergner (einem Diplomanden an der TUM) wurden sowohl die *'on-line'*-Version als auch die *'off-line'*-Version des Algorithmus getestet (siehe auch [12]). In Fällen, bei denen beide Versionen anwendbar waren, zeigte die *'off-line'*-Version im wesentlichen dieselbe Performanz wie die *'on-line'*-Version. Die folgenden Experimente basieren auf kontinuierlichen Strömen von Eingabeereignissen (statt auf klar separierten Trainingssequenzen). Der Beginn einer neuen Trainingssequenz wurde dem System niemals explizit mitgeteilt [105]. Dies bedeutet, daß die *'on-line'*-Version des Algorithmus verwendet werden mußte (siehe Abschnitt 3.3.1).

3.4.1 EIN EXPERIMENT MIT UNBEKANNTEN ZEITLICHEN VERZÖGERUNGEN

Bei diesem Experiment wurde das System mit einem kontinuierlichen Strom von Eingabeereignissen versorgt. F's Aufgabe bestand darin, seinen einzel-

nen Ausgabeknoten genau dann anzuschalten, wenn das Ereignis 'B' zum ersten Mal nach einem zu einem beliebigen vorherigen Zeitpunkt stattgefundenen Ereignis 'A' auftrat. Zu allen anderen Zeitpunkten sollte der Ausgabeknoten ausgeschaltet bleiben. Dies entspricht im wesentlichen der in [150] durch ein rekurrentes Netz gelösten *'flip-flop'*-Aufgabe.

Eine Schwierigkeit besteht bei diesem Problem darin, daß zwischen korrelierten Ereignissen im Prinzip beliebig lange zeitliche Verzögerungen eintreten können. Eine weitere Schwierigkeit liegt in der Abwesenheit von Information über Traininingssequenzbegrenzungen. Die Aktivationen und Gewichte der beteiligten Netze wurden zu keinem Zeitpunkt rückgesetzt oder von neuem initialisiert. Von Ereignissen aus früheren Trainingssequenzen verursachte Aktivationen konnten somit im Prinzip destruktiven Einfluß auf die Verarbeitung späterer Sequenzen nehmen.

Sowohl F als auch S besaßen die Topologie eines Standard-Perzeptrons: F verfügte über 3 Eingabeknoten für 3 mögliche Eingabeereignisse 'A', 'B', and 'C', welche in lokaler Manier repräsentiert wurden: Der Eingabevektor $(1,0,0)^T$ stand für 'A', $(0,1,0)^T$ stand für 'B', $(0,0,1)^T$ stand für 'C'. Zu einem gegebenen Zeitpunkt wurden die Eingabeknoten von F durch einen zufällig ausgewählten Eingabevektor aktiviert. F's Ausgabe war eindimensional. S besaß 3 Eingabeknoten für die möglichen Ereignisse 'A', 'B', and 'C'. Bei Verwendung von Architektur 1 verfügte S weiterhin über 3 Ausgabeknoten, einen für jedes schnelle Gewicht in W_F. Bei Verwendung von Architektur 2 besaß S 4 Ausgabeknoten, drei für F's Eingabeknoten und einen für F's Ausgabeknoten. Keines der Netze benötigte für die Aufgabe versteckte Knoten. Die Aktivierungsfunktion aller Ausgabeknoten war die Identitätsfunktion. Die Gewichtsmodifikationsfunktion (3.2) für die schnellen Gewichte war durch

$$\sigma(w_{ba}(t-1), \Box w_{ba}(t)) = \frac{1}{1+e^{-T(w_{ba}(t-1)+\Box w_{ba}(t)-\frac{1}{2})}} \tag{3.9}$$

gegeben (der Parameter T bestimmt hier die maximale Steigung der logistischen Funktion, welche die schnellen Gewichte auf das Intervall zwischen 0 und 1 begrenzt).

S' 'langsame' Gewichte wurden zufällig zwischen -0.1 und 0.1 initialisiert. Die Aufgabe galt als gelöst, falls für 100 aufeinanderfolgende Zeitschritte F's Ausgabefehler den Wert 0.05 nicht überstieg. Wurden die schnellen Gewichte gemäß Architektur 1 und Gleichung (3.4) geändert, fand das System bei $T = 10$ und $\eta = 1.0$ innerhalb von 300 Zeitschritten eine Lösung. Änderten sich die schnellen Gewichte hingegen gemäß Architektur 2 und Gleichung (3.5) (dies bedeutete 4 Ausgabeknoten für S), benötigte das System bei $T = 10$ und $\eta = 0.5$ 800 Zeitschritte zur Lösung des Problems.

Typischerweise besaßen die gefundenen Algorithmen folgende Eigenschaften: Der Auftritt eines 'A'-Signals wurde von S durch die Kreierung eines starken schnellen Gewichts für die von dem zu 'B' gehörigen Eingabeknoten ausgehende Verbindung in F beantwortet. Damit wurde das F-Netzwerk zu einem 'B'-Detektor. Trat nun irgendwann das 'B'-Signal auf, 'reinitialisierte' S das F-Netz, indem es das schnelle Gewicht auf der von dem zu 'B' gehörigen Eingabeknoten ausgehenden Verbindung in F wieder dramatisch schwächte. Dadurch wurde F bis zur Beobachtung des nächsten 'A's unempfindlich gegenüber weiteren 'B'-Signalen.

3.4.2 EXPERIMENTE ZUR TEMPORÄREN VARIABLENBINDUNG

Wie schon eingangs in einer Fußnote erwähnt, haben sich Kritiker darüber beklagt, daß KNN keinen Mechanismus zur Variablenbindung anbieten würden. Mit der im vorliegenden Kapitel präsentierten Methode ist es jedoch möglich, dynamische Verbindungen *à la* von der Malsburg [138] für temporäre Bindungsprobleme auszunützen. Obiger Algorithmus erlaubt zu lernen, Variableninhalte solange an Variablennamen zu binden, wie es im Kontext einer gegebenen Aufgabe nötig ist.

Bei folgendem einfachen Experiment geht es darum, den gegenwärtigen Aufenthaltsort eines Fahrzeugs in einer Parkgarage abzuspeichern. Dies involviert das Binden eines Werts an eine den Standort des Autos repräsentierende Variable.

Weder F noch S benötigten versteckte Knoten zur Lösung des Problems. Als Aktivierungsfunktion aller beteiligten Ausgabeknoten fand erneut die Identitätsfunktion Verwendung. Alle Systemeingaben sowie F's gewünschte Ausgaben bestanden aus Binärvektoren. F besaß einen Eingabeknoten, der für den Namen der Variablen WO-IST-MEIN-AUTO? stand. Außerdem verfügte F über 3 Ausgabeknoten (die sogenannten Parkplatzindikatoren) für die Namen dreier möglicher Parkplätze P_1, P_2, und P_3 (die möglichen Antworten auf WO-IST-MEIN-AUTO?). S besaß 6 Eingabeknoten sowie 3 Ausgabeknoten, einen für jedes schnelle Gewicht (nebenbei bemerkt: hier sieht man, daß S nicht immer dieselbe Eingabe haben muß wie F). Drei der sechs Eingabeknoten fungierten als 'Parkplatzdetektoren' – I_1, I_2, I_3. Die Parkplatzdetektoren wurden immer nur für einen einzigen Zeitschritt aktiviert: Besaß der Eingabevektor für diese drei Knoten den Wert $(1,0,0)^T$, so bedeutete dies 'Automobil wurde soeben im 1. Parkplatz abgestellt'. $(0,1,0)^T$ stand für 'Automobil wurde soeben im 2. Parkplatz abgestellt'. $(0,0,1)^T$ stand für 'Automobil wurde soeben im 3. Parkplatz abgestellt'. $(0,0,0)^T$ stand für 'Automobil steht entweder auf irgendeinem Parkplatz oder fährt herum, wurde jedenfalls nicht soeben abgestellt'. Die drei zusätz-

lichen Eingabeknoten wurden zu jedem Zeitpunkt durch zufällig gewählte binäre Werte besetzt. Die Wahrscheinlichkeit, daß einer dieser Knoten die Aktivation 1.0 aufwies, betrug dabei stets 0.5. Diese Zufallsaktivationen dienten als ablenkende zeitveränderliche Eingaben aus der Umgebung eines Autobesitzers, dessen (leicht eintöniges) Leben folgenden Verlauf nimmt: Er fährt sein Auto für 0 oder mehr Zeitschritte durch die Gegend, wobei die Wahrscheinlichkeit für die Beendigung der Fahrt zu jedem Zeitpunkt bei 0.25 liegt (solange die Fahrt andauert, trägt der Eingabevektor für die Parkplatzdetektoren den Wert $(0,0,0)^T$). Daraufhin parkt er sein Fahrzeug auf einem der 3 möglichen Parkplätze (was für die Dauer eines Zeitschrittes den entsprechenden Parkplatzdetektor aktiviert). Anschließend tätigt er Geschäfte außerhalb seines Autos, was ihn für 0 oder mehrere Zeitschritte beansprucht, wobei die Wahrscheinlichkeit für die Beendigung der Geschäftstätigkeit zu jedem Zeitpunkt 0.25 beträgt (alle Parkplatzdetektoren sind während dieser Zeit ausgeschaltet). Schließlich erinnert sich der Autobesitzer seines Parkplatzes, steigt in sein Fahrzeug und beginnt von neuem, herumzufahren etc.. Dieses Verhalten erzeugt einen kontinuierlichen Strom von sechs-dimensionalen Eingabevektoren.

Die Aufgabe des Systems bestand darin, die gegenwärtige Position des Automobils abzuspeichern. Es wurde trainiert, indem der WO-IST-MEIN-AUTO?-Eingabeknoten zu zufällig gewählten Zeitschritten aktiviert wurde und F's erwünschte Ausgabe (die Aktivierung des zum gegenwärtigen Parkplatz P_i gehörigen Ausgabeknotens) gemäß (3.6) zur Berechnung der Gewichtsänderungen für S diente.

S' Gewichte wurden wieder zufällig zwischen -0.1 und 0.1 initialisiert. Falls für 100 aufeinanderfolgende Zeitschritte F's Ausgabefehler den Wert 0.05 nicht überstieg, galt die Aufgabe als gelöst. Mit Architektur 1 und Gewichtsmodifikationsfunktion (3.9) fand das System bei $T = 10$ und $\eta = 0.02$ innerhalb von 6000 Zeitschritten eine Lösung.

Wie erwartet, lernte S dabei, Parkplatzindikatoren durch kurzfristig starke Verbindungen an den WO-IST-MEIN-AUTO?-Knoten zu binden. Dank der lokalen Ausgaberepräsentation war es einfach, die Verbindungsmuster zu analysieren: Zu einem gegebenen Zeitpunkt wies die Verbindung vom WO-IST-MEIN-AUTO?-Knoten zum gegenwärtig relevanten Parkplatzindikator ein starkes schnelles Gewicht auf (vorausgesetzt, das Auto befand sich gerade in geparktem Zustand). Die übrigen schnellen Gewichte wurden währenddessen unterdrückt.

SCHRANKEN DES ANSATZES

Gerade wie die rekurrenten Netzwerke des letzten Kapitels geraten auch die Algorithmen des vorliegenden Kapitels in Schwierigkeiten, wenn *alle*

Trainingssequenzen lange zeitliche Verzögerungen zwischen korellierten Ereignissen aufweisen. Beim Variablenbindungsexperiment aus dem vorangehenden Abschnitt trat dieser Effekt dank der gelegentlich kurzen zeitlichen Abstände zwischen relevanten Ereignissen im Eingabestrom nicht in Erscheinung. Erhöhte man beim obigen Versuch jedoch die Mindestzahl der Zeitschritte, die während der Trainingsphase zwischen zwei 'Parkaktionen' verstreichen müssen, auf 10, so fand das System innerhalb von 10^6 Zeitschritten keine Lösung mehr. Auch zur Standardaufgabe (siehe Abschnitt 2.6.4) analoge Probleme ließen sich mit der im vorliegenden Kapitel dargelegten Methode nicht in akzeptabler Zeit lösen. Dies liefert erneut Motivation für die in Kapitel 7 vorzustellenden erweiterten Verfahren zur Sequenzanalyse.

3.5 SCHLUSSBEMERKUNGEN

Es sei darauf hingewiesen, daß *Kapitel 8* durch die Präsentation eines Lernalgorithmus für das erste *'selbst-referentielle'* rekurrente KNN noch einen Schritt über das vorliegende Kapitel hinausgehen wird. Dieses 'introspektive' KNN verfügt über spezielle Eigenschaften, die es ihm erlauben, *alle eigenen* Gewichtsparameter zu analysieren und explizit zu manipiulieren, und zwar einschließlich derjenigen Parameter, die für die Analysier- und Manipulierprozesse zuständig sind. Theoretisch vermag dieses Netz nicht nur seinen eigenen Gewichtsänderungsalgorithmus zu ändern, sondern auch die Art und Weise, in der es seinen eigenen Gewichtsänderungsalgorithmus ändert, sowie die Art und Weise, in der es den Algorithmus ändert, der seinen eigenen Gewichtsänderungsalgorithmus ändert, und so fort *ad infinitum*.

Eine weitere Motivation für Netze, die ihre *eigenen* Gewichte (statt lediglich die Gewichte fremder Netzwerke) manipulieren können, ergibt sich aus folgenden Überlegungen:

Ein konventionelles vollständig rückgekoppeltes Netz mit n Nichteingabeknoten verfügt über n Variablen (die zeitveränderlichen Knotenaktivationen) zur Speicherung zeitlicher Ereignisse im Kurzzeitgedächtnis. In Kapitel 2 haben wir gesehen, daß der populäre RTRL-Algorithmus $O(n^4)$ Operationen pro Zeitschritt jeder Eingabesequenz benötigt. Das Verhältnis R zwischen Operationen pro Zeitschritt und der Anzahl der Variablen zur Speicherung von Ereignissen beträgt demnach also $O(n^3)$. Der beschleunigte Algorithmus (Abschnitt 2.5) drückt dieses Verhältnis durch Reduzierung der durchschnittlichen Anzahl von Operationen pro Zeitschritt auf $O(n^2)$. Es stellte sich nun kürzlich heraus, daß es möglich ist, einen komplementären Ansatz zur Erzielung von $R = O(n^2)$ zu verfolgen [116], welcher die Zahl der

Operationen pro Zeitschritt bei $O(n^4)$ beläßt, gleichzeitig jedoch die Anzahl der Variablen zur Speicherung von Ereignissen auf $O(n^2)$ hochschraubt. Dies wird erreicht, indem dem Netzwerk gestattet wird, während der Verarbeitung einer Eingabesequenz seine *eigenen* $O(n^2)$ Gewichte aktiv zu modifizieren. Hierbei findet eine schnelle Variante der Hebb-regel Verwendung, um sofortige (möglicherweise dramatische) Änderungen von Verbindungen zwischen zu sukzessiven Zeitschritten aktiven Knoten hervorzurufen. Für eine derartige Architektur wurde kürzlich ein exakter gradientenbasierter Algorithmus für überwachtes Sequenzlernen (eine Erweiterung von RTRL mit im wesentlichen *gleicher* Berechnungskomplexität) abgeleitet [116].

Kapitel 4

DISTANZIERTE PERFORMANZMASSE

In den beiden vorangegangenen Kapiteln wurden Fehlertrajektorien und Zielfunktionen durch bekannte Zielwerte für Ausgabeknoten definiert. Im Kontext des Erlernens motorischer Bewegungsabläufe entspricht dies der Annahme, daß die angemessene Aktivation jedes gewisse Muskelkontraktionen steuernden Neurons zu jedem Zeitpunkt von vornherein bekannt ist. Solche Annahmen sind unrealistisch, wenn es um die Erklärung der Lernvorgänge in biologischen Systemen geht. Auch das ambitionierte Ziel der Konstruktion von Robotern, die *ohne Lehrer* aus Versuch und Irrtum zielgerichtete Verhaltensweisen lernen, läßt sich mit den bisher beschriebenen überwachten Lernverfahren alleine nicht erreichen.

Wir betrachten daher nun den Fall, daß die gewünschten Ausgaben *nicht* von vornherein bekannt sind, sondern lediglich eine die durch Aktionen eines 'Steuermoduls' hervorgerufenen Umgebungszustände bewertende Evaluierungsfunktion gegeben ist. Aufgrund der zwischen Ausgabe des Steuermoduls und Evaluierung zwischengeschalteten Umgebungsdynamik nennen wir die Evaluierungsfunktion ein *distanziertes* Performanzmaß. Im gleichen Sinne sprechen wir von einem distanzierten Lehrer (*'distal teacher'*, [41]).

Ein Beispiel liefert das sogenannte Lernen durch *'Reinforcement'*, im folgenden auch *R-Lernen* genannt (unglücklicherweise existiert für R-Lernen kein passender deutscher Name). Beim R-Lernen teilt die Umgebung einem lernenden Agenten mittels ihrer (meist sehr einfachen) Evaluierungsfunktion mit, ob der gegenwärtige Zustand der Umgebung 'gut' oder 'schlecht' ist, oft erlaubt das Performanzmaß auch Zwischenabstufungen wie 'mittelgut' etc... Die Ausgabe der Evaluierungsfunktion kann als Schmerz- oder Lust-

signal interpretiert werden. Die Schwierigkeit beim R-Lernen besteht darin, daß *keine* Information geliefert wird, die sich sofort in einen Gradienten für die Ausgabeknoten (und damit mittels Kettenregel in einen Gradienten für die versteckten Knoten) ummünzen ließe. Ob ein bestimmter Ausgabeknoten zu einem gegebenen Zeitpunkt mehr oder weniger aktiv sein soll, ist (im Gegensatz zu den in den Kapiteln 2 und 3 beschriebenen Verfahren) aus dem Performanzmaß alleine nicht sofort ersichtlich.

Dieses Kapitel beschäftigt sich im ersten Beitrag mit Situationen, in denen Grund zur Annahme besteht, daß die Abbildung von Netzwerkausgaben auf Performanzmaß in differenzierbarer Weise *modelliert* werden kann[1]. Das Modell läßt sich seinerseits durch ein separates adaptives Netzwerk mittels eines separaten Hilfsperformanzmaßes erstellen. Das Hilfsmodul dient zur Überbrückung der zwischen Netzausgaben und Zielfunktion klaffenden 'Differenzierbarkeitslücke'. Seine Existenz ermöglicht die Anwendung der Kettenregel zur Berechnung des Gradienten für das uns eigentlich interessierende Steuermodul.

Im zweiten (originären) Beitrag vertiefen wir die Analyse differenzierbarer Modelle von Steuerprogrammeffekten und beschreiben sowohl azyklische als auch rekurrente neuronale 'Subzielgeneratoren'. Letztere sind Netze, die mittels Kettenregel lernen, als Antwort auf eine Kombination des gegenwärtigen Umgebungszustandes und eines gewünschten Zielzustandes eine *Liste geeigneter Subziele* auszugeben. Mit Hilfe dieser Subzielgeneratoren werden experimentell Pfadfindungsprobleme in zweidimensionalen Welten mit Hindernissen gelöst.

4.1 WELTMODELLBAUER

Die grundlegenden Konzepte dieses Abschnitts gehen zurück auf Werbos [143] und Jordan (e.g. [41]).

4.1.1 ZWEI PERFORMANZMASSE

Nehmen wir an, ein (später noch genauer zu spezifizierendes) *Steuermodul* C beantwortet den p-ten Zustandsvektor x^p aus der Umgebung mit einem 'Aktionsvektor' a^p. Die Umgebung berechnet ihrerseits aus einem Zustandsvektor x^p und einem Aktionsvektor a^s mittels einer Evaluationsfunktion

[1] Es gibt Algorithmen für R-Lernen, die keine derartigen Annahmen machen, z.B. [7], [8], [147], [6], [9], [90], [140], [97], [51], [92]. Die theoretischen Grundlagen der meisten dieser Algorithmen reichen allerdings höchstens in Spezialfällen aus, um im asymptotischen Fall die Konvergenz zu (sub)optimalen Lösungen zu beweisen. Wir wollen uns hier auf Algorithmen beschränken, die aus bestimmten Voraussetzungen über Performanzmaß und Architektur mathematisch sauber *ableitbar* sind.

eval ein Resultat (z.B. einen neuen Zustand, oder einen *Reinforcement*-Wert, oder beides)

$$u^{p,s} = eval(x^p, a^s).$$

Setzen wir weiterhin voraus, daß ein 'distanziertes Performanzmaß' bestimmte *erwünschte* Zustandsvektoren d^p favorisiert. (Liefert *eval* beispielsweise 'Schmerzsignale' unterschiedlicher Intensität, so zeichnen sich erwünschte Zustände durch die Abwesenheit von Schmerz aus.) Ein sinnvolles derartiges zu minimierendes Performanzmaß ist

$$E_C = \sum_p \frac{1}{2} \sum_i (e_i^p)^2,$$

mit

$$e_i^p = d_i^p - u_i^{p,p}.$$

Das in diesem Abschnitt zu besprechende Problem besteht darin, daß *eval* in der Regel unbekannt ist – man kann aus E_C nicht sofort durch Differenzieren einen Lernalgorithmus herleiten. Wir haben es nicht mit einfachem überwachten Lernen zu tun.

Eine Lösung des Problems besteht darin, *eval* selbst durch eine differenzierbare Menge parametrisierter Funktionen (mit adaptiven Parametern) zu approximieren. Ein zusätzliches (später noch genauer zu spezifizierendes) *Weltmodell* M läßt sich dazu verwenden, die Konkatenation $x^p \circ a^s$ der Vektoren x^p und a^s mit einem den zu erwartenden geänderten Umgebungszustand prophezeihenden 'Prediktionsvektor' $y^{p,s}$ zu beantworten. Für M müssen wir dazu ein zweites Hilfs-Performanzmaß einführen:

$$E_M = \sum_{p,s} \frac{1}{2} \sum_i (y_i^{p,s} - u_i^{p,s})^2.$$

Das mit Hilfe von E_M trainierte Modul M erlaubt uns, wie bald zu sehen sein wird, die Anwendung der Kettenregel zur Minimierung der uns eigentlich interessierenden Zielfunktion E_C.

4.1.2 ARCHITEKTUR

Das *Steuernetzwerk* C sei (zur vereinfachten Darstellung des Prinzips) ein konventionelles azyklisches BP-Netzwerk. Alle Knoten in C seien durchnumeriert, die Aktivation des k-ten Knotens in Antwort auf den p-ten an der Eingabelage anliegenden Eingabevektor x^p sei mit c_k^p bezeichnet, wobei $c_i^p = x_i^p$, falls i Eingabeknoten ist. In der r-ten Lage von C ($r > 1$) berechnet sich c_k^p wie folgt:

$$net_k^p = \sum_{l \in \ Lagen \ <r} w_{kl} c_l^p, \quad c_k^p = f_k(net_k^p), \tag{4.1}$$

wobei w_{kl} das Gewicht der gerichteten Verbindung vom Knoten l zum Knoten k darstellt, und f_k wieder für eine differenzierbare Aktivierungsfunktion steht. Der Ausgabevektor der obersten Lage heiße a^p.

Das *Weltmodellnetzwerk* M sei ebenfalls als azyklisches BP-Netzwerk implementiert. M's Eingabe ist die Konkatenation $x^p \circ a^s$ der Vektoren x^p und a^s. Für M's Trainingsphase wird i.a. $p \neq s$ gelten, für C's Trainingsphase hingegen stets $p = s$. Die Aktivation des k-ten Knotens von M in Antwort auf $x^p \circ a^s$ wird mit $m_k^{p,s}$ bezeichnet, wobei $m_i^{p,s}$ gleich der i-ten Komponente von $x^p \circ a^s$ ist, falls i Eingabeknoten ist. In der r-ten Lage von M ($r > 1$) berechnet sich $m_k^{p,s}$ analog zur Aktivationsausbreitung in C wie folgt:

$$net_k^{p,s} = \sum_{l \in \ Lagen \ <r} w_{kl} m_l^{p,s}, \quad m_k^{p,s} = f_k(net_k^{p,s}). \tag{4.2}$$

Siehe hierzu Abbildung 4.1.

4.1.3 ALGORITHMUS

Durch Präsentation von Kombinationen möglicher Zustands- und Aktionsvektoren sowie durch Beobachtung der entsprechenden Umgebungseffekte wird zunächst M durch konventionelles BP (siehe Kapitel 1) adjustiert. Nach jeder Präsentation des Trainingsensembles ändert sich dabei jedes Gewicht w_{ij} in M gemäß

$$\triangle w_{ij} \sim -\frac{\partial E_M}{\partial w_{ij}}. \tag{4.3}$$

Nach Abschluß von M's Trainingsphase werden M's Gewichte 'eingefroren'. M dient ab jetzt als starre Approximation von *eval* zur Berechnung von Fehlergradienten für C. Jedes Gewicht w_{ij} in C wird nun gemäß

$$\triangle w_{ij} \sim -\frac{\partial \sum_p \frac{1}{2} \sum_i (y_i^{p,p} - d_i^p)^2}{\partial w_{ij}} = -\sum_p \delta_i^p c_j^p \tag{4.4}$$

geändert, wobei

$$\delta_i^p = -\frac{\partial \frac{1}{2} \sum_i (y_i^{p,p} - d_i^p)^2}{\partial net_i^p}$$

gilt. Zu diesem Zweck berechnen wir für alle p zunächst durch konventionelles BP in M die Werte

$$\kappa_i^p = -\frac{\partial \frac{1}{2} \sum_i (y_i^{p,p} - d_i^p)^2}{\partial a_i^p}. \tag{4.5}$$

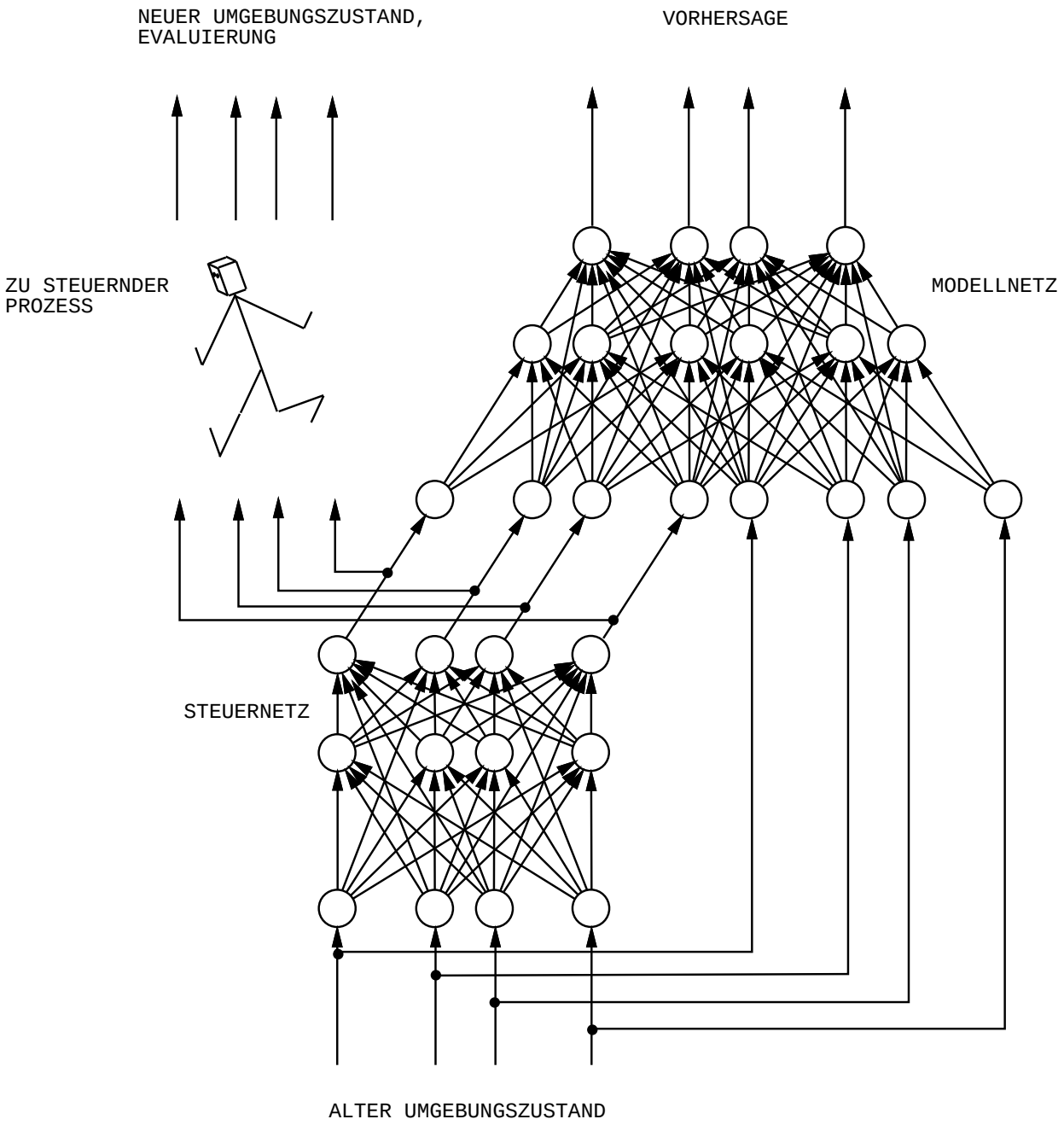

Abbildung 4.1: Ein Steuernetz reagiert auf den Umgebungszustand mit einem Steuersignal für einen externen Prozeß. Die Abbildung von Kombinationen von Zuständen und Aktionen auf Effekte in der Umgebung wird durch ein zweites adaptives 'Modellnetzwerk' modelliert. Letzteres dient auch zur Gradientenberechnung für das Steuernetz.

Falls i einen Ausgabeknoten von C bezeichnet, so ist

$$\delta_i^p = f_i'(net_i^p)\kappa_i^p$$

Steht i hingegen für einen versteckten Knoten der r-ten Lage von C, so gilt

$$\delta_i^p = f_i'(net_i^p) \sum_{l \in \ Lagen \ > r} w_{li}\delta_l^p.$$

Erweiterung auf dynamische Umgebungen. Sind sowohl M als auch C rekurrent, lassen sich alle drei in Kapitel 2 vorgestellten Algorithmen auf M bzw. auf die ebenfalls rekurrente Kombination von M und C anwenden. Derartige Erweiterungen obigen Prinzips auf zyklische Netzwerke in nichtstationären Umgebungen mit externer Rückkopplung[2] wurden in [96], [98], und [94] beschrieben.

EXPERIMENTE

Das Weltmodellbauprinzip wurde u.a. erfolgreich auf das Problem des sequentiellen Andockens eines rückwärts fahrenden Spielzeuglastwagens mit Anhänger an eine Laderampe angewendet [63]. Zunächst wurde dabei ein Weltmodellnetzwerk M anhand zufällig gewählter Trainingsbeispiele daraufhin trainiert, bei gegebener Lastwagenposition, gegebenem Winkel zwischen Lastwagen und Anhänger, gegebenem Einschlagwinkel des Lenkrades und bei fixer Lastwagengeschwindigkeit die Lastwagenposition zum nächsten diskreten Zeitschritt vorherzusagen. In der zweiten Phase lernte ein Steuernetzwerk C unter Zuhilfenahme von M, Sequenzen von Lenksignalen auszugeben, so daß der Anhänger am Ende einer größenordnungsmässig 20 Zeitschritte umfassenden Trajektorie (mit zufällig gewählter Startposition und zufälligem Anfangswinkel zwischen Anhänger und Lastwagen) stets korrekt an der Laderampe andockte.

Eine erfolgreiche Anwendung eines erweiterten Verfahrens auf das Problem des Erlernens zielgerichteter Retinatrajektorien zur selektiven Aufmerksamkeitssteuerung findet sich in [117] (siehe auch [118] und [93]). Das Ziel des Systems bestand darin, ohne Lehrer zu lernen, sequentielle Steuersignale zu erzeugen, so daß die Endposition einer durch die Steuersignale bewegten künstlichen Retina (mit hoher Auflösung im Zentrum und niedriger Auflösung in den peripheren Bereichen) einem zu findenden Objekt in einer visuellen Szene entsprach (dies läßt sich als eine Form gerichteter Aufmerksamkeitssteuerung interpretieren). Motivation war hierbei, die

[2] Bei externer Rückkopplung können *Aus*gaben zu früheren Zeitpunkten Einfluß auf spätere *Ein*gaben nehmen.

kaum erfolgreichen und ineffizienten rein statischen Ansätze zur Mustererkennung durch einen effizienteren sequentiellen Ansatz zu ersetzen. Dieser Ansatz war inspiriert durch die Beobachtung, daß biologische Systeme den Mustererkennungsprozeß auf sequentielle Augenbewegungen abstützen. In der ersten Phase wurde ein Modellnetzwerk M dabei daraufhin trainiert, bei gegebener zufällig gewählter Retinaeingabe und zufällig gewähltem Steuersignal die Retinaeingabe zum nächsten Zeitschritt möglichst gut vorherzusagen. In der zweiten Phase (nach Einfrieren von M's Gewichten) lernte C im Laufe von größenordnungsmäßig 20000 Trainingsversuchen unter Zuhilfenahme von M, Sequenzen von Kombinationen von Retinatranslationen und -rotationen zu erzeugen, und zwar dergestalt, daß die Retinaeingabe am Ende der größenordnungsmässig 20 Zeitschritte umfassenden Trajektorie gleich einer gewünschten finalen Eingabe korrespondierend zu dem zu findenden Objekt war. Kein Lehrer teilte C dabei mit, welche Steuerausgabe es zu welchem Zeitpunkt auszugeben hatte. Die einzige Information über das Ziel bestand aus der gewünschten Eingabe am Ende der Steuersequenz.

Weitere Applikationen (u.a. auf Balancierprobleme in Nicht-Markov-Umgebungen) werden in [108] beschrieben.

4.2 ADAPTIVE SUBZIELGENERIERUNG

Alle bisher behandelten Algorithmen betreiben stets sturen Gradientenabstieg im Performanzmaß, wobei jede Aktion zu jedem Zeitpunkt in gleichberechtigter Weise zur Berechnung des Gradienten herangezogen wird. Dieser Ansatz ist zwar allgemein, verzichtet jedoch auf möglichen Effizienzgewinn im folgenden Sinne: In vielen typischen Umgebungen lassen sich Lösungen für neu auftretende Probleme in hierarchischer Weise aus Lösungen für weniger komplexe, schon bekannte Probleme zusammensetzen. In solchen Fällen wäre es in der Tat Zeit- und Ressourcenverschwendung, jedes Detail einer bisher unbekannten umfangreichen Aktionssequenz aufs Neue zu lernen, wenn es statt dessen irgendwie möglich wäre, auf höherem Niveau denselben Effekt durch das geeignete Zusammensetzen bereits existierender 'Unterprogramme' zu erlernen ($\rightarrow$ *teile und herrsche!*).

Im folgenden untersuchen wir eine hierarchische Architektur, die auf derartige Situationen zugeschnitten ist. Statt Aktionen zu beliebigen Zeitpunkten zu betrachten, konzentriert sich der problemlösende Teil der Architektur auf das Finden geeigneter Schnittstellen (den *Subzielen*) zwischen u.U. langen Aktionsuntersequenzen (den *Unterprogrammen*).

Wie im letzten Abschnitt beruht die Architektur auf einem differenzierbaren Modell der Effekte bestimmter Netzausgaben auf ein 'distanziertes Performanzmaß'. Ein- und Ausgaben dieses Modells werden jedoch mit ei-

ner im Vergleich zum letzten Abschnitt sehr unterschiedlichen Interpretation belegt. Sie lassen sich in von Abschnitt 4.1 abweichender Weise zur Gradientenbestimmung heranziehen.

Wie immer leiten wir nach der Definition von Architektur und Zielfunktion mittels Kettenregel den Lernalgorithmus (diesmal einen für adaptive Subzielgenerierung) ab. Experimente zur adaptiven Hindernisvermeidung illustrieren schließlich die Arbeitsweise des Verfahrens.

4.2.1 ARCHITEKTUR

DIE GRUNDLEGENDEN MODULE

Ein Unterprogramm p sei eine Sequenz von Aktionen, die von einem Startzustand s^p zu einem Zielzustand g^p führen. Sowohl s^p als auch g^p seien hier reelle Vektoren, die einen möglichen Zustand der Umgebung repräsentieren.

Eines der drei an unserer Gesamtarchitektur beteiligten Module ist der Programmausführer C. C kann ein neuronales Netzwerk sein, muß aber nicht. (Tatsächlich stellt C im Kontext der Subzielgenerierung die unbedeutendste Komponente dar.) Zum Zeitpunkt t der Ausführung des p-ten Programms produziert C einen Ausgabevektor $o^p(t)$ und nimmt einen Eingabevektor $x^p(t)$ aus der Umgebung wahr. Es sei

$$dim(x^p(t)) = dim(g^p) = dim(s^p).$$

Daß C das Programm p ausführen soll, erfährt er durch einen zusätzlichen stationären Eingabevektor $s^p \circ g^p$, der Konkatenation des entsprechenden Start- und Zielzustandes.

g_p kann als Aktivationsmuster angesehen werden, welches den finalen gewünschten Zustand beschreibt. $s^p \circ g^p$ läßt sich als 'Programmname' betrachten. Wir nehmen an, daß C bereits eine Anzahl von Programmen korrekt ausführen kann. Diese Programme mögen durch einen konventionellen Lernalgorithmus oder durch eine rekursive Anwendung des unten zu beschreibenden Verfahrens erworben worden sein.

Abbildung 4.2 zeigt ein zweites Modul, das Evaluatormodul E, dessen Eingabe erneut die Konkatenation $s \circ g$ eines einen Startzustand repräsentierenden Vektors s und eines einen Zielzustand repräsentierenden Vektors g ist. E's eindimensionale Ausgabe $eval(s, g) \in R_0^+$ wird als Vorhersage der *Kosten* (= negatives *'Reinforcement'*) interpretiert, die mit dem entsprechenden von s nach g führenden Programm assoziiert sind. Eine Evaluation von 0 bedeutet minimale zu erwartende Kosten.

E stellt ein Modell der gegewärtigen Fähigkeiten des Programmausführers dar. Wir wollen auch E hier nicht detailliert spezifizieren – E mag sowohl

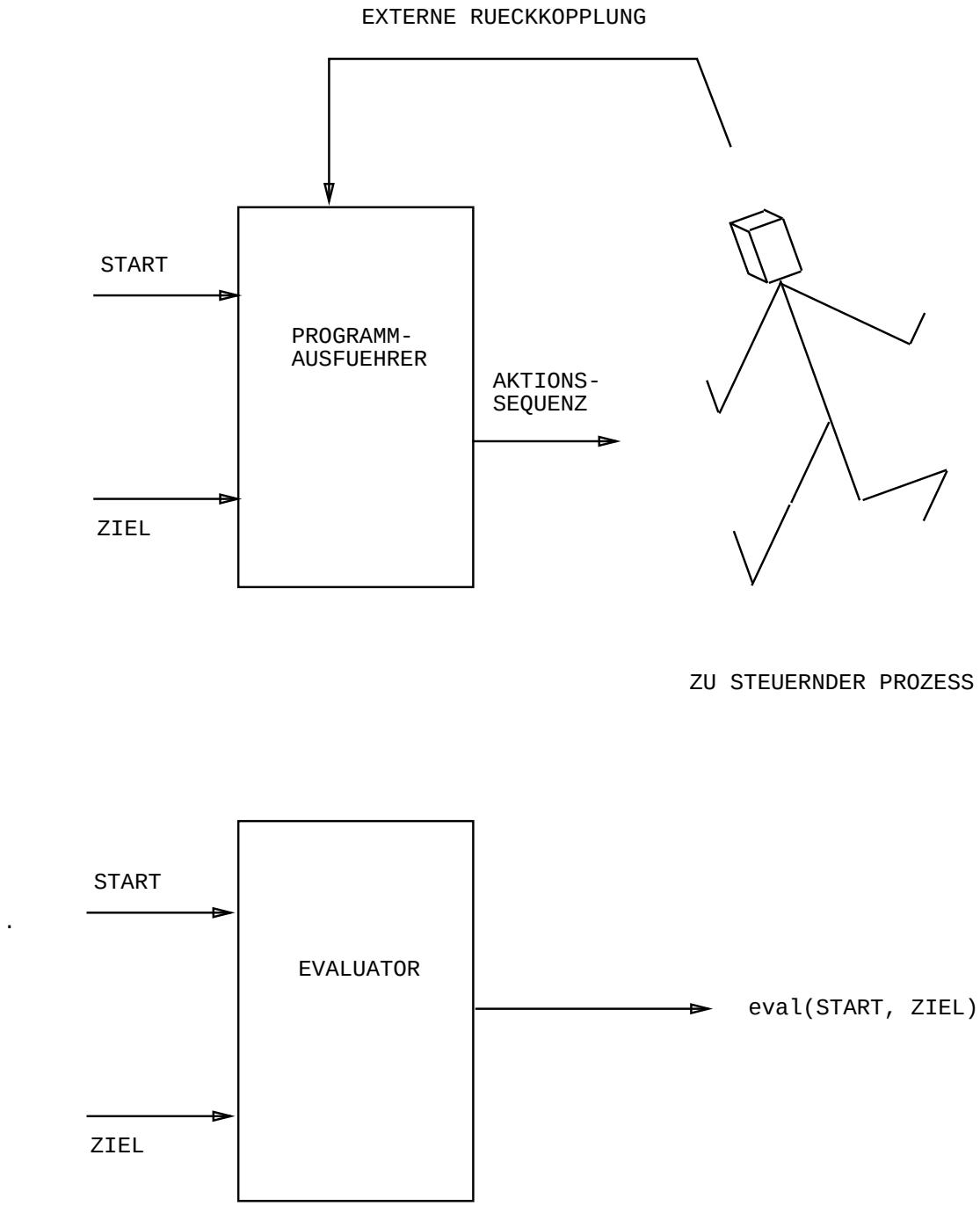

Abbildung 4.2: Der *Programmausführer* produziert Aktionssequenzen (Programme), deren Namen er durch Eingabe einer START/ZIEL-Kombination erfährt. Das *Evaluatormodul* liefert Vorhersagen über die für die Ausführung eines gegebenen Programms zu erwartenden Kosten.

ein neuronales Netzwerk mit bezüglich der Eingabe differenzierbaren Ausgaben [106] als auch irgendeine andere differenzierbare Abbildung sein.

Das für uns interessanteste der drei Module ist der adaptive Subzielgenerator S. S soll lernen, neue Start-Ziel Kombinationen durch eine Liste geeigneter aufeinanderfolgender Subziele zu beantworten.

Das i-te vektorwertige Subziel der Subzielliste ($i = 1 \ldots n$) heiße $s^p(i)$, die j-te Komponente des Vektors $s^p(i)$ sei wie stets mit $s^p_j(i)$ bezeichnet. Alle $s^p(i)$ haben gleiche Dimension $dim(s^p(i)) = dim(x^p(t)) = dim(g^p) = dim(s^p)$. Es sei $s^p = s^p(0), g^p = s^p(n+1)$. Im Idealfall sollte die Subzielliste $s^p = s^p(1), s^p(2), ..., s^p(n) = g^p$ nach der Lernphase folgende Bedingung erfüllen:

$$eval(s^p(0), s^p(1)) = eval(s^p(1), s^p(2)) = \ldots = eval(s^p(n), s^p(n+1)) = 0. \tag{4.6}$$

(4.6) bedeutet, daß es ein kostenfreies Unterprogramm gibt, das vom Startzustand zum ersten Subziel führt, ein weiteres, das vom ersten Subziel zum zweiten Subziel führt, und so fort, bis das Endziel erreicht ist. Wir werfen nun einen näheren Blick auf zwei unterschiedliche Architekturen für S.

SUBZIELARCHITEKTUR 1

Abbildung 4.3 zeigt einen statischen Subzielgenerator S (ein azyklisches BP-Netz), der als Antwort auf eine Start/Ziel-Kombination eine von vornherein festgelegte Zahl n von Subzielen liefert. S ist in eine Architektur eingebettet, die $n+1$ Kopien des Evaluatornetzes mit umfaßt.

S' Eingabevektor ist die Konkatenation $s^p \circ g^p$ der vektorwertigen Repräsentation des Startzustandes $s^p = s^p(0)$ und der Repräsentation des Zielzustandes $g^p = s^p(n+1)$.

S' Ausgabevektor ist die Konkatenation

$$s^p(1) \circ s^p(2) \circ \ldots \circ s^p(n)$$

der n die Subziele repräsentierenden Vektoren $s^p(i), i = 1 \ldots n$.

$n+1$ Kopien des Evaluatormodules E werden dergestalt mit dem Subzielgenerator verschaltet (siehe Abbildung 4.3), daß die Eingabe der k-ten Kopie von E gleich $s^p(k-1) \circ s^p(k)$ ist. Die Ausgabe der k-ten Kopie von E wird damit zu $eval(s^p(k-1), s^p(k))$.

SUBZIELARCHITEKTUR 2

Ein Nachteil der Subzielarchitektur 1 besteht darin, daß die Anzahl der Subziele pro Problem als konstant angenommen wird. In typischen Umgebungen ist diese Annahme unrealistisch. Im allgemeinen bräuchte ein

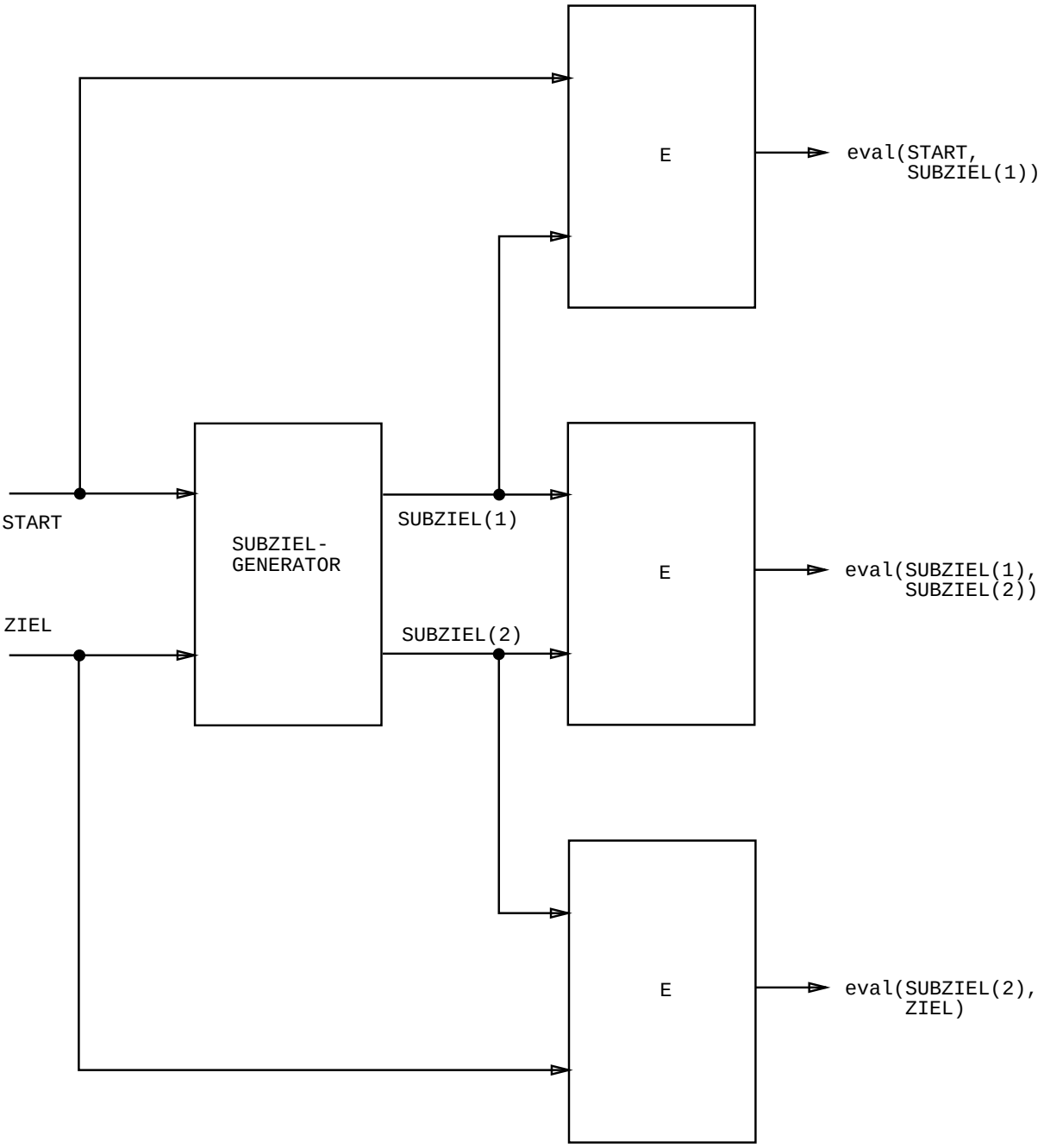

Abbildung 4.3: Die Ausgaben des azyklischen Subzielgenerators (die Subziele) werden von Kopien des differenzierbaren Evaluatormoduls bewertet – letztere erlauben die Berechnung eines Gradienten für die Ausgaben des Subzielgenerators (und damit auch für seine Gewichte).

System basierend auf Subzielarchitektur 1 eine ganze Reihe von Subzielgeneratoren: Einen für Probleme, die nur ein Subziel erfordern, einen weiteren für Probleme, die zwei Subziele erfordern, und so fort.

Subzielarchitektur 2 hingegen gestattet ein und demselben Subzielgenerator S, verschieden lange Sequenzen von Subzielen zu produzieren. Dies wird erreicht, indem man S' Ausgabe wie folgt auf die eigene Eingabe rückkoppelt:

Bei einer gegebenen durch ein Start/Ziel-Paar ($s^p = s^p(0), g^p = s^p(n+1)$) spezifizierten Aufgabe p ist S' Eingabevektor zum 1. Iterationsschritt gleich $s^p \circ s^p(n+1)$. S' Ausgabevektor ist $s^p(1)$.

Beim Iterationsschritt $t, 1 < t < n+1$ wird S' Eingabevektor zu $s^p(t-1) \circ s^p(n+1)$, die Ausgabe wird zu $s^p(t)$. Zur Berechnung der Aktivation $o_k^p(t)$ des k-ten Nichteingabeknotens in S zum Iterationsschritt t schreiben wir

$$net_k^p(t) = \sum_{l \in \ Lagen \ <r} w_{kl} o_l^p(t), \quad o_k^p(t) = f_k(net_k^p(t)), \tag{4.7}$$

wobei f_k wie üblich eine differenzierbare Aktivierungsfunktion darstellt.

Wieder verwenden wir das Evaluatormodul E zur Berechnung von $eval(s^p(k-1), s^p(k)), k = 1, \ldots n+1$, aus $s^p(k-1) \circ s^p(k)$. Die Architektur hat sich aber nun gemäß Abbildung 4.4a geändert. Abbildung 4.4b zeigt dieselbe Architektur im zeitlich entfalteten Zustand (nach der Generierung dreier sukzessiver Subziele).

4.2.2 PERFORMANZMASS

Für beide Subzielarchitekturen wollen wir dieselbe Zielfunktion minimieren, nämlich die Summe aller vorhergesagter Kosten

$$E^p = \sum_p \sum_{k=1}^{n+1} \frac{1}{2} (eval(s^p(k-1), s^p(k)))^2. \tag{4.8}$$

4.2.3 DIE ALGORITHMEN

Die Summe der Gradienten für verschiedene Probleme p ist gleich dem Gradienten der Summe. Daher genügen uns für die verschiedenen Architekturen Methoden zur Berechnung von

$$\triangle W_S^T = \eta_S \frac{\partial \sum_{k=1}^{n+1} \frac{1}{2} eval^2(s^p(k-1), s^p(k))}{\partial W_S} =$$

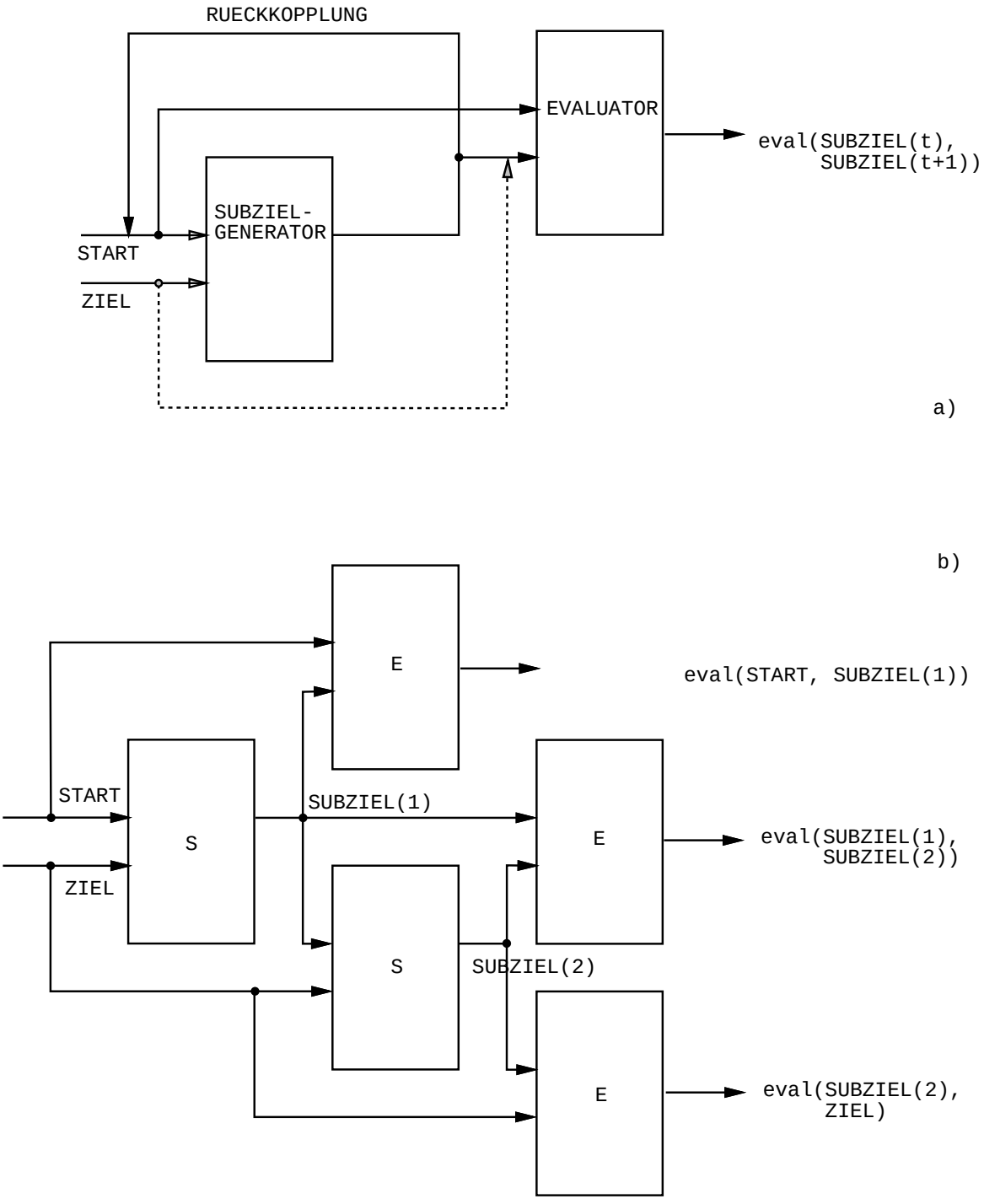

Abbildung 4.4: (a) Die Ausgabe des rekurrenten Subzielgenerators S zur Zeit t wird auf seinen eigenen START-Eingang rückgekoppelt. S produziert demnach eine zeitlich gedehnte Sequenz von Subzielen. Der Evaluator E prophezeiht die für zwei aufeinanderfolgende Subziele zu erwartenden Kosten. Die gestrichelte Linie zeigt an, daß E am Ende der Subzielsequenz das eigentliche ZIEL als Eingabe erhalten muß. (b) zeigt die rekurrente Architektur (a) im 'zeitlich entfalteten' Zustand (für 2 Subziele).

$$= \eta_S \sum_{k=1}^{n+1} eval(s^p(k-1), s^p(k)) \left(\frac{\partial s^p(k-1)}{\partial W_S}\right)^T \left(\frac{\partial e(s^p(k-1), s^p(k))}{\partial s_p(k-1)}\right)^T \tag{4.9}$$

wobei η_S die Lernrate von S, W_S seinen Gewichtsvektor, und $\triangle W_S$ des Gewichtsvektors Inkrement bezeichnet. (Beachte: $\left(\frac{\partial s^p(k-1)}{\partial W_S}\right)^T$ stellt eine Jacobimatrix dar.)

Ist E selbst ein BP-Netzwerk (wie z.B. in [106]), so verwenden wir nach Abschluß von E's Trainingsphase konventionelles BP zur Berechnung von

$$\frac{\partial eval^2(s^p(k-1), s^p(k))}{\partial s_j^p(k)} \tag{4.10}$$

für alle j. E's Gewichte bleiben dabei 'eingefroren'. Andernfalls muß (4.10) eben analytisch berechnet werden (wir haben ja zu Beginn gefordert, daß *eval* bezüglich aller Parameter differenzierbar ist).

Für Architektur 1 ist nun die Berechnung von (4.9) nicht mehr schwierig: Das Fehlersignal für S' $(k\ dim(s^p) + l)$-ten Ausgabeknoten ($k = 0 \ldots n-1$, $l = 1 \ldots dim(s^p)$) korrespondierend zur $(k\ dim(s^p) + l)$-ten Komponente seines Ausgabevektors

$$s^p(1) \circ s^p(2) \circ \ldots \circ s^p(n)$$

ist gerade gleich

$$\frac{1}{2}\left[\frac{\partial eval^2(s^p(k), s^p(k+1))}{\partial s_l^p(k+1)} + \frac{\partial eval^2(s^p(k+1), s^p(k+2))}{\partial s_l^p(k+1)}\right].$$

Alle Fehlersignale interner Knoten lassen sich nun gemäß den inzwischen geläufigen BP-Regeln (siehe Kapitel 1) bestimmen.

Architektur 2 fordert ein etwas komplexeres, von BPTT (siehe Kapitel 2) inspiriertes Verfahren. Falls k ein Ausgabeknoten von S und $t = n$ ist, so ist das Fehlersignal für $o_k^p(t)$ gleich

$$\delta_k^p(t) = \frac{1}{2}\left[\frac{\partial eval^2(s^p(t-1), s^p(t))}{\partial s_k^p(t)} + \frac{\partial eval^2(s^p(t), s^p(t+1))}{\partial s_k^p(t)}\right]. \tag{4.11}$$

Falls k zwar ein Ausgabeknoten von S, aber $1 \leq t < n$ ist, so ergibt sich das Fehlersignal für $o_k^p(t)$ als

$$\delta_k^p(t) = \frac{1}{2}\left[\frac{\partial eval^2(s^p(t-1), s^p(t))}{\partial s_k^p(t)} + \frac{\partial eval^2(s^p(t), s^p(t+1))}{\partial s_k^p(t)}\right] + \kappa_k^p(t), \tag{4.12}$$

wobei sich

$$\kappa_k^p(t) = \frac{\partial \sum_{i \text{ in } Ausgabelage} \delta_i^p(t+1)}{\partial s_k^p(t)} \tag{4.13}$$

mit BP für alle Eingabeknoten k von S berechnen läßt. Gleichzeitig erhält man durch diesen BP-Pass alle durch den Iterationsschritt t verursachten Beiträge für S' Gewichtsänderungen, nämlich

$$\triangle^p w_{ij}(t) = -\frac{\partial E^p}{\partial w_{ij}(t)}, \tag{4.14}$$

wobei $w_{ij}(t)$ analog zu Kapitel 2 das 'virtuelle' Gewicht vom Knoten j zum Knoten i in der t-ten Kopie von S ist. Die vom Problem p verursachte Gesamtänderung eines Gewichts w_{ij} ergibt sich nun zu

$$\triangle^p w_{ij} = -\eta_S \sum_t \triangle^p w_{ij}(t). \tag{4.15}$$

Analog zu BPTT braucht man auch bei obigem Verfahren nicht n vollständige Kopien der beteiligten Netze anzulegen – es genügt, alle zu den verschiedenen Iterationsschritten auftretenden Aktivationen zu speichern.

Wie viele Subziele braucht man für welche Aufgaben? Die einfachste Antwort ist wohl die folgende: Versuche, eine gegebenen Aufgabe zunächst ohne Subziel zu lösen. Falls das nicht klappt, versuche es mit einem Subziel, dann mit zwei, etc. Eine mögliche Erweiterung dieses Versuch/Irrtum-Ansatzes bestünde darin, ein viertes Modul daraufhin zu trainieren, Start/Ziel-Kombinationen auf die minimale Anzahl der benötigten Subziele abzubilden.

4.3 EXPERIMENTE ZUR HINDERNISVERMEIDUNG

Im folgenden werden zwei Arten von Experimenten beschrieben. Bei den Experimenten der ersten Art konzentrieren wir uns zunächst ganz auf den Lernprozeß des Subzielgenerators: Die *eval*-Funktion und ihre partiellen Ableitungen werden auf analytischem Wege berechnet (siehe auch [122]). Bei den Experimenten der zweiten Art wird der Evaluator E selbst durch ein in einer separaten Lernphase trainiertes BP-Netzwerk implementiert (siehe auch [101] [22]).

4.3.1 EXPERIMENTE MIT NICHT ADAPTIVEM E

Zur Illustration des Subzielgenerierungsprozesses wurde in Zusammenarbeit mit Reiner Wahnsiedler (Diplomand an der TUM) eine in kartesischen Koordinaten definierte zweidimensionale 'Miniwelt' konstruiert [139].

Ein 'Agent' kann sich in einer durch x-Achse und y-Achse gegebenen Ebene reeller Zahlenpaare frei bewegen, so daß die ausgeführte Trajektorie (seine 'Spur') eine ein-dimensionale Mannigfaltigkeit in R^2 darstellt. In der Miniwelt existieren allerdings Hindernisse in Form kreisförmiger 'Sümpfe'. Solange sich der Agent außerhalb der Sümpfe bewegt, entstehen ihm keine Kosten (in Form negativen reellwertigen Reinforcements). Der Aufwand, den der Agent zur Querung eines Sumpfes treiben muß, berechnet sich wie folgt:

Der i-te kreisförmige Sumpf Φ_i mit Zentrum (x_i, y_i) und Radius r_i bildet die Basis eines in die durch die z-Achse definierte dritte Dimension wachsenden Kegels mit Spitze $(x_i, y_i, -h_i)$. Bei gegebener Spur ω sind die durch Φ_i verursachten Kosten gleich

$$\int_{\omega} g(x, y, \Phi_i)\ dx\ dy. \tag{4.16}$$

Hierbei gilt $g(x, y, \Phi_i) = 0$, falls (x, y) außerhalb von Φ_i liegt, andernfalls ist $g(x, y, \Phi_i)$ gleich dem Abstand zwischen (x, y) und dem Schnittpunkt der zur Miniwelt Senkrechten durch (x, y) mit dem zu Φ_i korrespondierenden Kegelmantel.

Des Agenten Aufgabe besteht in der 'Planung' einer von einem gegebenen Startpunkt zu einem gegebenen Endpunkt führenden Aktionssequenz mit *minimalen Kosten*. Für die Experimente wurde angenommen, daß die Kosten aller Unterprogramme, die zu *geradlinigen* Bewegungen des Agenten führen, bereits bekannt sind. Für alle derartigen Unterprogramme ist (4.16) einfach zu berechnen: Wieder sei der Startpunkt des i-ten Unterprogramms $s^p(k) = (s_1^p(k), s_2^p(k))$ und sein Zielpunkt $s^p(k+1) = (s_1^p(k+1), s_2^p(k+1))$. (4.16) wird damit zur durch den parabelförmigen Kegelschnitt und der durch den Agenten hinterlassenen Spur definierten Fläche

$$F(s_1^p(k), s_2^p(k), s_1^p(k+1), s_2^p(k+1), \Phi_i). \tag{4.17}$$

Siehe hierzu Abbildung 4.5.

Als Evaluationsmodul wurde natürlich die bezüglich Start und Zielpunkt eines Unterprogramms differenzierbare Summe aller Kosten

$$eval((s^p(k), s^p(k+1)) = \sum_i F(s_1^p(k), s_2^p(k), s_1^p(k+1), s_2^p(k+1), \Phi_i) \tag{4.18}$$

verwendet.

Die im folgenden gezeigten Abbildungen basieren auf Rechnerausdrucken in [139]. Man betrachte Abbildung 4.6. Ein einzelner Sumpf versperrt dem Agenten den bereits bekannten geradlinigen Weg vom Start zum Ziel. Für

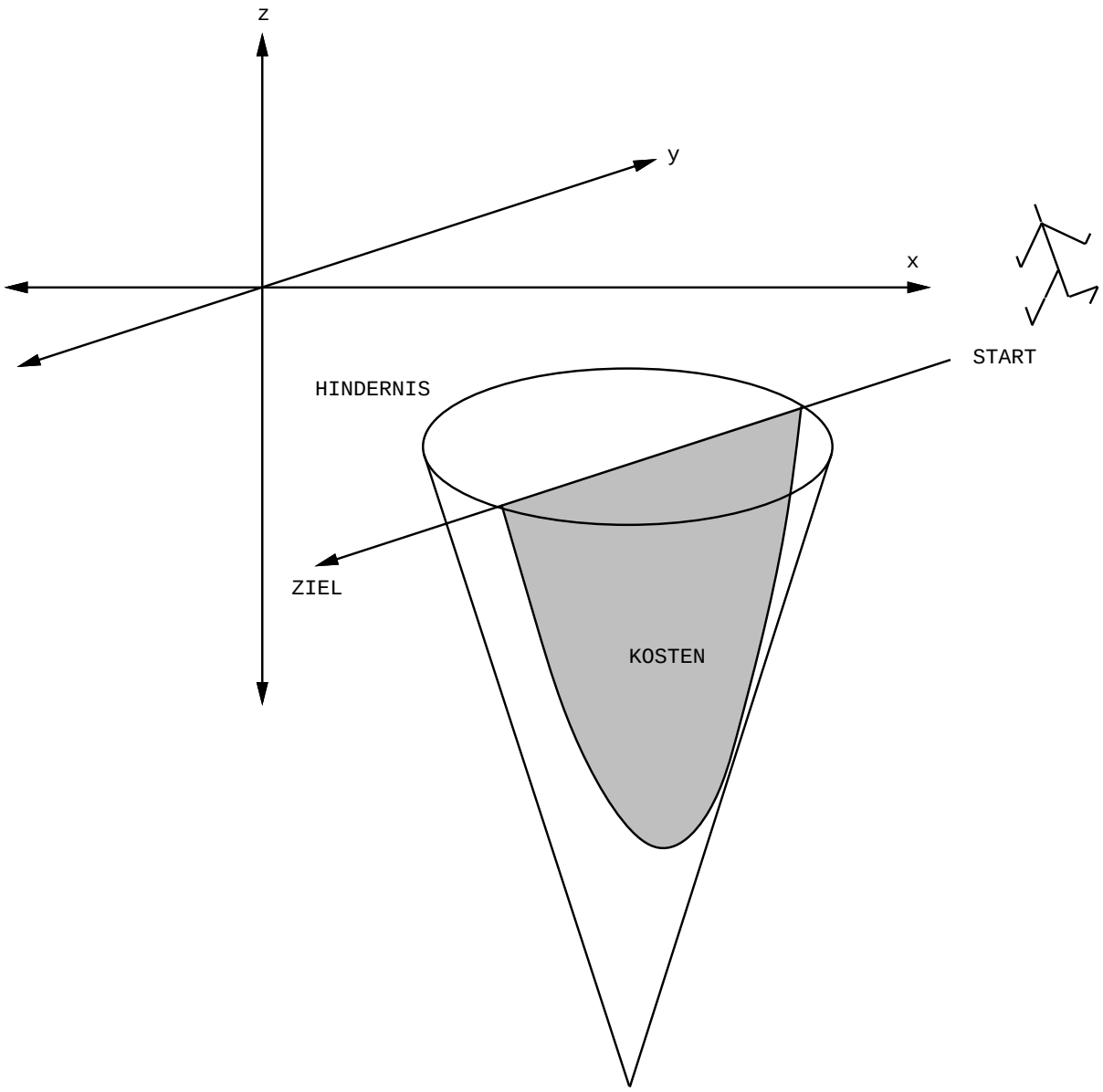

Abbildung 4.5: Ein kreisförmiger Sumpf steht dem Agenten auf seinem geradlinigen Marsch vom START zum ZIEL im Wege. Die *Kosten* der Querung des Sumpfes ergeben sich zu der grau markierten Fläche.

eine saubere kostenfreie Komposition einer Lösung aus schon bekannten Unterprogrammen stellen sich zwei Subziele als zweckmäßig heraus. Für einen statischen Subzielgenerator (Architektur 1) mit 4 Eingabeknoten und 4 Ausgabeknoten (für zwei Subziele) erwiesen sich bei 4 versteckten Knoten und einer Lernrate von $\eta_S = 0.2$ drei Iterationsschritte als ausreichend, um eine befriedigende Subzielkombination zu finden.

Der rekurrente Subzielgenerator (Architektur 2) mit 4 Eingabeknoten, aber nur zwei Ausgabeknoten (dieselben Ausgabeknoten können ja bei dieser Architektur für verschiedene aufeinanderfolgende Subziele verwendet werden) stieß erwartungsgemäß auf etwas größere Schwierigkeiten, dieselbe Aufgabe zu lösen. Da ein und derselbe Ausgang von S zu verschiedenen Zeitpunkten verschiedene kontextabhängige Subzielrepräsentationen emittieren soll, leidet Architektur 2 stärker unter dem altbekannten *'cross-talk'*-Phänomen als Architektur 1. Diesem Problem kann man durch Erniedrigung der Lernrate beikommen, wofür man allerdings in Form von mehr Trainingsiterationen zahlen muß. Bei 40 versteckten Knoten und einer Lernrate von $\eta_S = 0.03$ wurden 22 Iterationsschritte zur Auffindung einer befriedigenden Lösung benötigt. Siehe hierzu Abbildung 4.7.

Bei mehr als einem Hindernis erwies sich folgende Initialhilfestellung für den Subzielgenerator günstiger als eine vorurteilsfreie zufällige Gewichtsinitialisierung: Bei gegebener Start/Ziel-Kombination wurde S zunächst ohne Rücksicht auf etwaige im Wege stehende Sümpfe daraufhin trainiert, äquidistante Subziele auf der Start und Ziel verbindenden Linie auszugeben. Erst danach begann die eigentliche kostenminimierende Lernphase. Abbildung 4.8 zeigt die Evolution der aus 4 Subzielen bestehenden Ausgabe eines wie oben initialisierten statischen Subzielgenerators mit 10 Ausgabeknoten, 40 versteckten Knoten bei einer Lernrate von $\eta_S = 0.002$.

Größere Lernraten führten bei diesem Problem zu schlechterer Performanz. Der Grund liegt in der aufgrund der nahe beieinanderliegenden zahlreichen Hindernisse vergleichsweise komplexen Zielfunktion. Je komplexer die Zielfunktion (je kleiner der Einzugsbereich globaler oder lokaler Minima), desto kleiner muß i.a. die Lernarte gewählt werden, und desto mehr Trainingsiterationen sind i.a. erforderlich. Unglücklicherweise gibt es aufgrund der Problemabhängigkeit geigneter Lernraten keine allgemeine Methode zur optimalen Lernrateneinstellung. Die einfachste (und in dieser Arbeit verwendete) Methode zur Auffindung brauchbarer Lernraten besteht von Fall zu Fall im systematischen Probieren.

4.3.2 EXPERIMENTE MIT ADAPTIVEM E

Die folgenden Untersuchungen wurden von Martin Eldracher und Boris Baginski mit Hilfe von Bernhard Glavinas Robotersimulationsumgebung

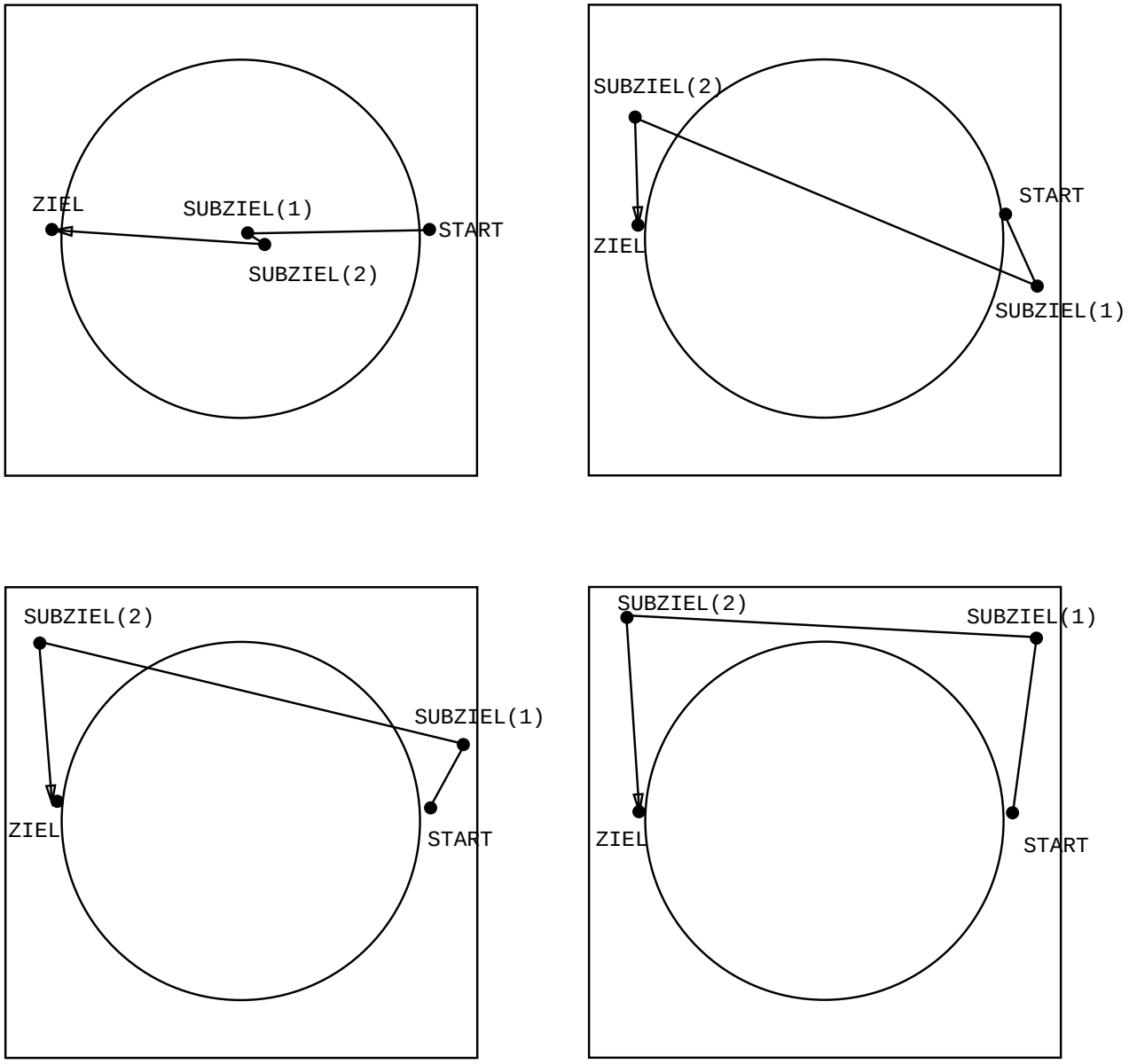

Abbildung 4.6: Ein einzelner Sumpf (Kreis) versperrt den geradlinigen Weg vom START zum ZIEL. Gezeigt ist die Evolution zweier von einem azyklischen Subzielgenerator ausgegebener Subziele.

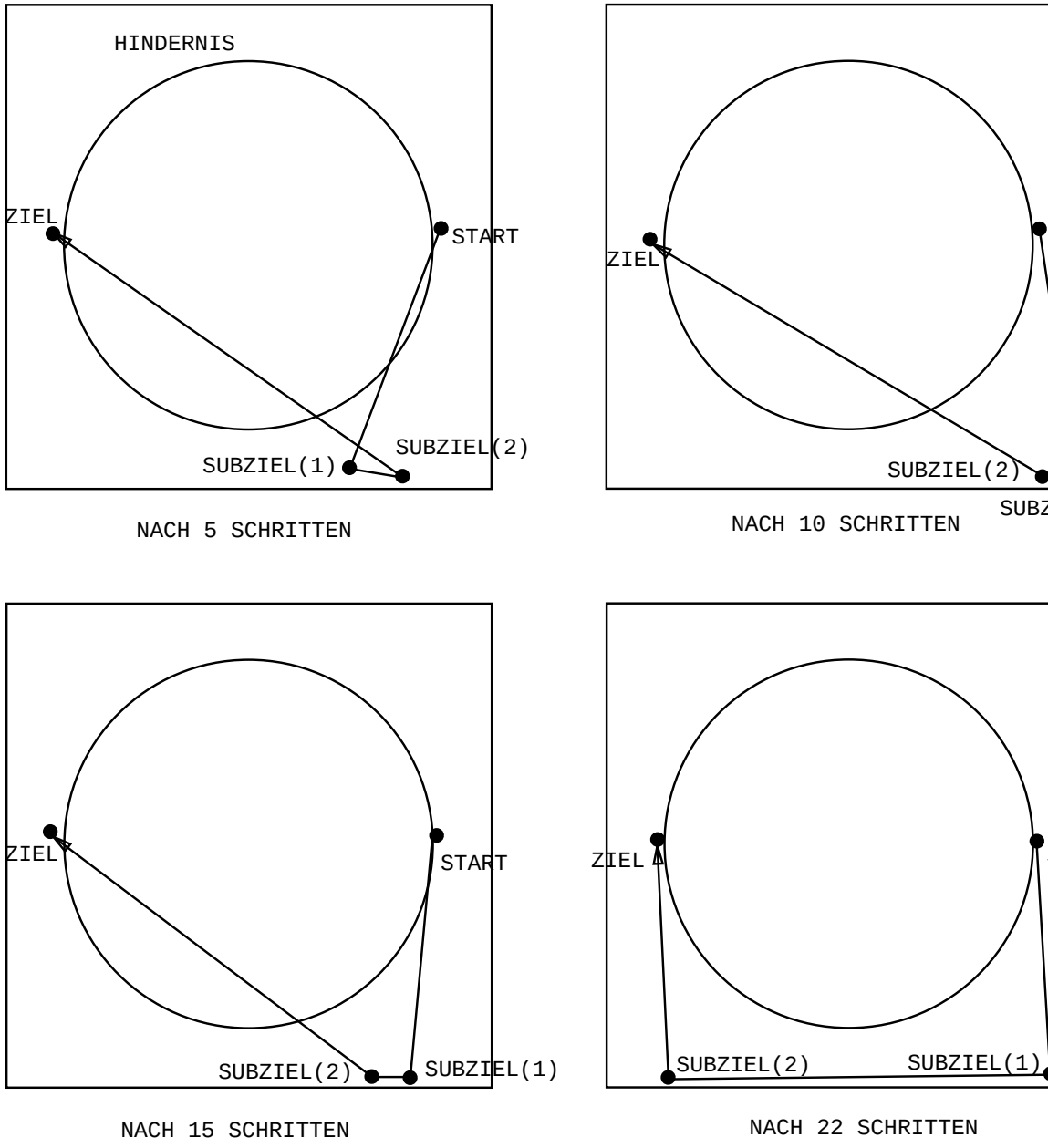

Abbildung 4.7: Wie Abbildung 4.6 – allerdings nun mit rekurrentem Subzielgenerator (Architektur 2).

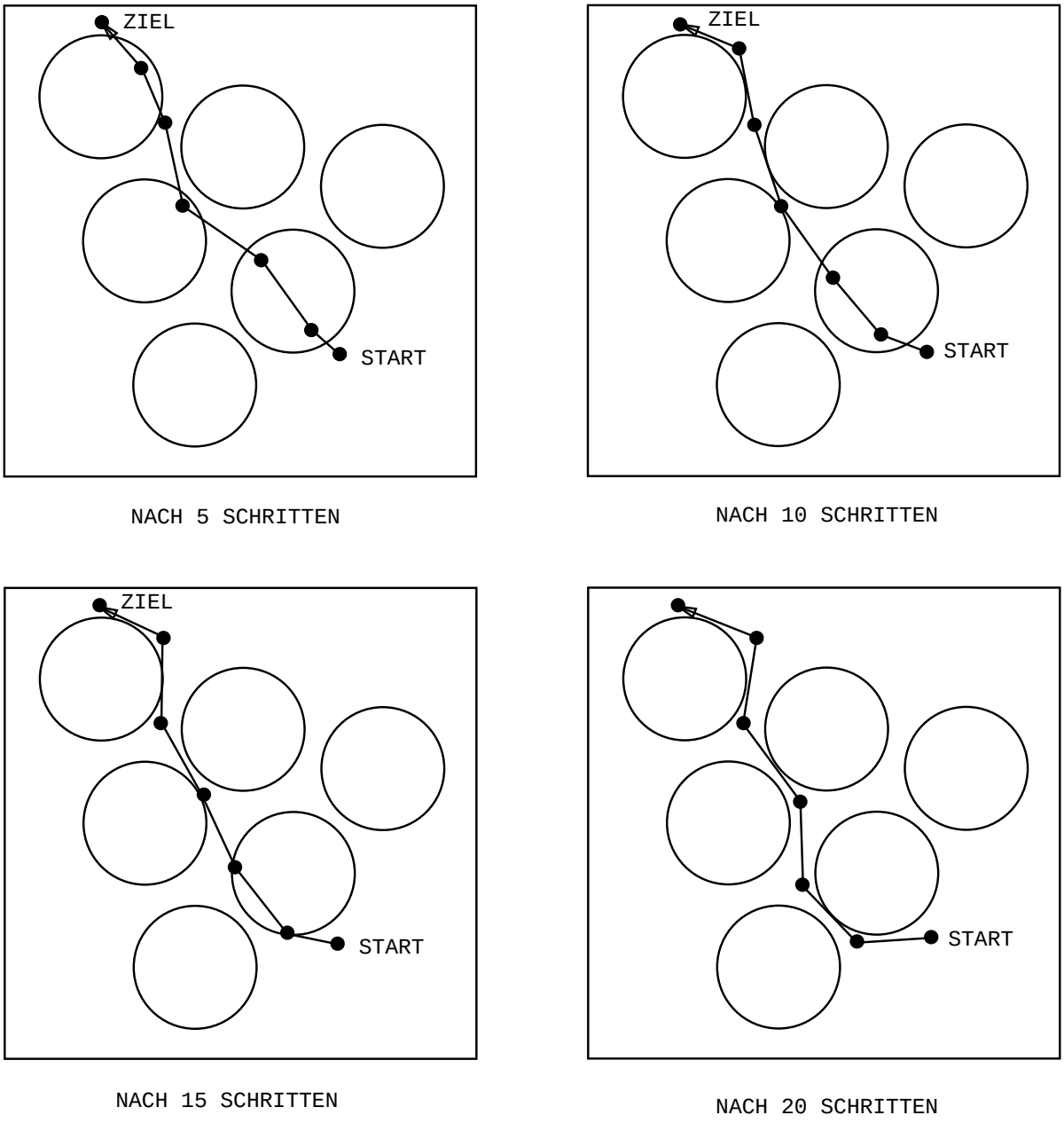

Abbildung 4.8: Eine ganze Reihe von Sümpfen versperrt den geradlinigen Weg vom START zum ZIEL. Gezeigt ist die Evolution von fünf von einem azyklischen Subzielgenerator emittierten Subzielen.

[30] durchgeführt (siehe auch [22]).

Abbildung 4.9 zeigt einen zweigelenkigen Manipulatorarm in einer Umgebung mit zwei seine Bewegungsfreiheit einschränkenden Wänden als Hindernissen.

Im Gegensatz zu den Experimenten aus dem vorangegangenem Abschnitt war E selbst als vollständig vorwärts vernetztes BP-Netzwerk implementiert (Knoten einer bestimmten Lage entsprangen Verbindungen zu allen Knoten höherer Lagen). E wurde mit BP daraufhin trainiert, vorherzusagen, ob ein zufällig gewählter Startpunkt im Winkelraum (siehe Abbildung 4.10) mit einem zufällig gewählten Zielpunkt durch eine kollisions- und krümmungsfreie Winkelraumtrajektorie verbunden werden kann.

E's Trainingsphase lief wie folgt ab: Die 4-dimensionale Eingabe bestand stets aus einer Start/Zielkombination aus dem Bereich $[0,\ldots,2\pi] \times [0,\ldots,2\pi] \times [0,\ldots,2\pi] \times [0,\ldots,2\pi]$. Die erste Komponente des Eingabevektors beschrieb dabei den Anstellwinkel des 'Oberarms' des Manipulators für die Startposition, die zweite Komponente beschrieb den Anstellwinkel des 'Unterarms' (die beiden verbleibenden Komponenten standen in analoger Weise für die Zielposition). E's gewünschte Ausgabe war 1, falls sich durch Ausprobieren in der Robotersimulationsumgebung herausstellte, daß eine geradlinige Verbindung zwischen Start und Ziel im Winkelraum möglich war (Klasse 1), und 0 sonst (Klasse 2).

Für jede der beiden Klassen wurden 50000 Trainingsbeispiele verwendet. Tabelle 4.1 zeigt die Resultate für verschiedene Netzwerktopologien (angegeben ist die Zahl der Knoten in aufeinanderfolgenden Lagen) und 'Epochenzahlen', wobei eine Epoche einem Durchlauf durch alle Trainingsbeispiele entspricht. Die angegebenen Prozentsätze beziehen sich auf die Zahl der korrekt klassifizierten Start/Zielkombinationen, wobei eine Evaluatorausgabe $\geq 1-\varepsilon$ ($\leq \varepsilon$) als korrekte Klassifikation von Klasse 1 (Klasse 2) akzeptiert wurde.

Netztopologie	Epochen	Klassifikationssatz			
		$\varepsilon = 0.5$		$\varepsilon = 0.1$	
		Training	Test	Training	Test
4-25-25-25-1	4,000	99.3%	98.9%	96.8%	96.6%
4-40-40-40-1	2,000	99.3%	98.9%	96.5%	96.2%

Tabelle 4.1: *Prozentsatz korrekter Klassifikationen des Evaluators nach dem Training (für Trainingsdaten bzw. Testdaten).*

Nach Beendigung der Trainingsphase des Evaluators wurde ein statischer, vollständig vorwärts vernetzter Subzielgenerator S mit 4 Eingabeknoten und 2 Ausgabeknoten (zur Ausgabe eines Subziels im Winkelraum)

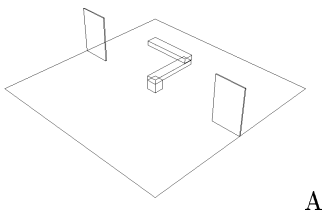

Abbildung 4.9: *Simulationsumgebung mit zweigelenkigem Manipulator und zwei Wänden als Hindernissen.*

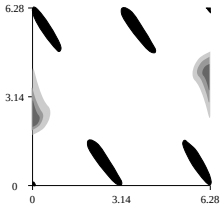

Abbildung 4.10: *Winkelraum: Die durch die Hindernisse verbotenen Bereiche sind schwarz gekennzeichnet. Subziele konzentrieren sich nach dem Training in den grauen Bereichen.*

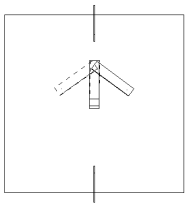

Abbildung 4.11: *Die beiden Zentren der Subzielhäufungen in kartesischen Koordinaten.*

gemäß Abbildung 4.3 mit zwei Evaluatorkopien zusammengeschaltet.

Einzelne Subziele für spezifische Probleme zu lernen, fiel S nun ähnlich leicht wie bei den Experimenten aus dem vorangegangenen Abschnitt. Um jedoch S dazu zu veranlassen, über einen weiten Bereich von Problemstellungen (Start/Zielkombinationen) auf Anhieb brauchbare Ergebnisse zu produzieren, mußte S vergleichsweise langsam trainiert werden:

Tabelle 4.2 zeigt die mit verschiedene Netzwerktopologien und 'Epochenzahlen' erhaltenen Resultate. Die angegebenen Prozentsätze beziehen sich auf die Zahl der als 'gültig' bewerteten Subziele, wobei ein Subziel als 'gültig' angesehen wurde, wenn die entsprechenden Ausgaben beider Evaluatorkopien 0.95 überstiegen. Ein sehr hoher Prozentsatz der 'gültigen' Subziele erwies sich in der Simulationsumgebung tatsächlich als brauchbar, wie sich aus der Spalte mit der Überschrift *'Güte der generierten Subziele'* in Tabelle 4.2 ergibt. Die Mehrheit der ausgegebenen Subziele entsprach dabei einer angewinkelten Manipulatorposition, die auch nach menschlichem Ermessen sinnvoll ist (siehe Abbildungen 4.10 und 4.11 für Veranschaulichungen der Subzielhäufungen). Die Spalte mit dem Titel *'Gesamtlösungssatz'* in Tabelle 4.2 schließlich zeigt, daß der Subzielgenerator nach dem Training für über 90 Prozent der Testfälle auf Anhieb ein geeignetes Subziel ausgeben konnte (die verbleibenden Fälle erforderten zusätzliche Gradientenabstiegsiterationen).

Diese von Eldracher und Baginski durchgeführten Untersuchungen zeigen, daß die Trainingsvorarbeiten für das Subziele generierende System (insbesondere das Trainieren des Evaluators) aufwendig sein können. Nach der Lernphase kann S jedoch in vielen Fällen schnell (auf Anhieb) geeig-

Netztopologie	Subziele oberhalb der Schwelle				Güte der generierten Subziele		Gesamt-Lösungssatz	
	Epochen							
	1000	2000	3000	test	training	test	training	test
4-5-5-5-2	92.9%	93.0%	93.8%	93.2%	99.9%	99.9%	93.8%	93.2%
4-10-10-10-2	93.2%	93.6%	93.7%	93.3%	98.7%	97.9%	91.9%	91.3%
4-20-20-20-2	92.7%	93.5%		93.2%	97.3%	97.1%	91.0%	90.8%

Tabelle 4.2: *Resultate des Subzielgenerators nach dem Training.*

nete Subziele produzieren. Die Beschreibungen weiterer Experimente mit adaptivem E finden sich in [22] und [101].

4.3.3 SCHRANKEN DER SUBZIELGENERATOREN

Der oben beschriebene Ansatz vertraut auf differenzierbare (möglicherweise adaptive) Modelle der mit bereits bekannten Aktionssequenzen assoziierten Kosten. In diesen Modellen (den Evaluationsmodulen) residiert das Domänenwissen – letzteres wird durch den Subzielgenerierungsprozeß extrahiert. Es lassen sich jedoch Domänen finden, für die differenzierbare Evaluationsmodule entweder ungeeignet oder aber nur mit Schwierigkeiten oder mit vergleichsweise großem Aufwand (wie im letzten Abschnitt) zu erstellen sind. Für solche Fälle werden sich möglicherweise gewisse neuartige Ideen zum hierarchischen Reinforcement-Lernen *ohne* differenzierbare Kostenmodelle als hilfreich erweisen [128][140][21].

Kapitel 5

UNÜBERWACHTES LERNEN: ZIELFUNKTIONEN

Nachdem die Kapitel 2 und 3 ein sehr allgemeines Performanzmaß für überwachtes Lernen mit zwei unterschiedlichen Architekturklassen minimierten, und nachdem das dritte Kapitel zeigte, wie mittels zusätzlicher differenzierbarer Modelle der Effekte von Netzausgaben auch zielgerichtete Lernalgorithmen für Situationen *ohne* Lehrer ableitbar sind, erhebt sich die Frage, ob nun die Möglichkeiten des in dieser Arbeit verfolgten Ansatzes ausgeschöpft sind. Was wollte man denn noch mehr erreichen?

Die Antwort lautet: Auch wenn sich die bisher behandelten Performanzmaße und die entsprechenden Algorithmen durch Allgemeinheit auszeichnen, so lassen sich dennoch in gewissen typischen Umgebungen durch Einführung zusätzlicher Zielfunktionen und Architekturen für *unüberwachtes Lernen* Effizienzgewinne erzielen.

Einige der ersten (von biologisch orientierten Forschern vorgeschlagenen) Algorithmen für unüberwachtes Lernen kamen durch *ad-hoc* Überlegungen zustande. Prominentestes Beispiel hierfür ist die Hebb'sche Regel aus dem Jahre 1949 [33]: *Sind zwei Neuronen gleichzeitig aktiv, so stärke die zwischen ihnen existierende Verbindung.* Wir werden derartige *ad-hoc* Lernregeln hier *nicht* berücksichtigen. Statt dessen konzentrieren wir uns wieder ausschließlich auf Algorithmen, die aus sinnvollen vorgegebenen Performanzmaßen ableitbar sind. Interessanterweise wird sich dabei herausstellen, daß dieser formale Ansatz Varianten gewisser ehemals nur intuitiv motivierter Lernregeln (wie der Hebbschen Regel) im nachhinein rechtfer-

tigt.

Das Kapitel erhebt keinen Anspruch auf Vollständigkeit – das Material in der Literatur ist bereits zu umfangreich, um auf wenigen Seiten dargestellt zu werden. Wir werden jedoch wesentliche Motivationen und Aspekte bisher in der Literatur aufgetauchter *'moderner'* unüberwachter Lernalgorithmen für stationäre Umgebungen in repräsentativer Manier zusammenfassen. Das Wort 'modern' soll hier den Gegensatz zu älteren, nicht durch mathematisch saubere Herleitung aus sinnvollen Zielfunktionen gewonnenen Algorithmen betonen. An geeigneter Stelle werden originäre Beiträge eingeflochten, unter anderem ein neues, durch Gradientenabstieg gewonnenes Verfahren zur unüberwachten Bildung lokaler Musterrepräsentationen sowie eine neuartige Methode zur Extraktion nicht-trivialer wechselseitiger Information aus Eingabemustern. Letztere erweist sich in Experimenten zur Extraktion von Stereoinformation aus einfachen Stereobildern als geeigneter als eine ältere, von Becker und Hinton vorgeschlagene Methode [11].

5.1 UNÜBERWACHTES LERNEN: WOZU?

Wir betrachten ein System, das Eingaben aus einer unbekannten Umgebung intern repräsentieren soll. Wir konzentrieren uns auf den Fall, daß die interne Repräsentation des p-ten externen Eingabevektors x^p ein n-dimensionaler Vektor y^p ist.

Im Gegensatz zu den bisher behandelten zielgerichteten Verfahren (für überwachtes Lernen und R-Lernen) fragt man im Kontext des unüberwachten Lernens: Lassen sich bei einer gegebenen Umgebung nützliche zielunabhängige interne Repräsentationen der Eingaben konstruieren? Die beiden kritischen Wörtchen sind hier die scheinbar widersprüchlichen Adjektive *'nützlich'* und *'zielunabhängig'*.

Die Eigenschaften 'nützlich' und 'zielunabhängig' sind in dem Maße kompatibel, in dem Umgebungseingaben dergestalt transformierbar sind, daß sich die durch die Transformationen gewonnenen internen Repräsentationen als Eingaben für *zielgerichtete* Lernsysteme (mit nicht von vornherein bekannten Zielen) gegenüber den untransformierten 'rohen' Eingaben durch irgendwelche Vorteile auszeichnen. Solche Vorteile können dabei z.B. durch Zeit- und Ressourcenersparnis beim zielgerichteten Lernen definiert sein. Unüberwachtes Lernen für sich allein macht keinen Sinn. Es ist dann berechtigt, wenn es den Lernvorgang erleichternde 'Präprozessoren' für die Eingaben zielgerichteter Systeme ermöglicht.

Genau genommen gibt es kein allgemeines unüberwachtes Lernverfahren, welches das Erlernen beliebiger Ziele vereinfacht – falls nämlich beispielsweise die Probleme eines zielgerichteten Lerners nichts mit den Umge-

bungseingaben zu tun haben, ist es sinnlos, Aufwand zur Kreierung interner Repräsentationen zu treiben. In allen realistischen Fällen sind vernünftige Ziele jedoch in irgendeiner Weise mit den Umgebungszuständen verknüpft: Häufig sucht man beispielsweise Abbildungen von die Umgebungszustände repräsentierende Eingaben auf geeignete zielgerichtete Aktionen. Mit unüberwachten Lernalgorithmen kann man in derartigen Fällen versuchen, *'Regularitäten'* aus den Eingaben zu extrahieren und geeignet zu repräsentieren, so daß ohne detailliertes Vorwissen über später zu lösende Aufgaben Effizienzgewinne erwartet werden können.

Wie definiert man 'Regularitäten'? Normalerweise geht man diese Frage vom Standpunkt des Statistikers aus an[1]. Es wurden in der Literatur mehrere Performanzkriterien zur Fassung statistischer Regularitäten vorgeschlagen, die verschiedenartige Vorteile eröffnen. Wir unterscheiden 5 Kategorien unüberwachter Ziele, die allerdings bis auf Kategorie 4 alle etwas mit der Entdeckung statistischer Abhängigkeiten zwischen den Eingabekomponenten zu tun haben:

(1) Redundanzminimierung. Typischerweise enthalten die einzelnen Komponenten der 'rohen' Umgebungseingaben redundante Information. Zielfunktionen zur Redundanzelimination ermöglichen kompakte Repräsentationen und damit Speicherplatzreduktion sowie ein Potential für erhöhte Generalisierungsfähigkeit [10].

(2) Informationsmaximierung. Abbildungen mit maximaler Informationstransmission vom Eingaberaum zum Raum interner Repräsentationen sind insbesondere bei beschränkter Repräsentationskapazität unter der Annahme verrauschter Eingaben *nicht* äquivalent zu Identitätsabbildungen. Dies liefert Motivation für nicht-triviale Zielfunktionen zur Informationsmaximierung.

(3) Dekorrelation zur Beschleunigung überwachter Lernvorgänge. Liefert man einem überwacht lernenden linearen Netzwerk L als Eingaben Musterrepräsentationen mit unkorrelierten Komponenten, so ist die Hessematrix

$$\nabla_{w_L} \nabla_{w_L} E_L$$

von L's (als zweimal differenzierbar vorausgesetzter) Fehlerfunktion E_L

[1] Das Konzept einer 'Regularität' kleidet sich jedoch in viele Gewänder. Um die Tatsache zu würdigen, daß es nicht nur einen einzigen allgemein akzeptierten Ansatz zur Definition 'regulärer Struktur' gibt, beachte man, daß *algorithmische Informationstheorie* [43][17] reguläre Muster in einer von *statistischer Informationstheorie* [125] unterschiedlichen Weise definiert. Die algorithmische Information eines Musters ist im wesentlichen die Läge des kürzesten Programms, das das Muster reproduzieren kann. Da es kein allgemeines Verfahren zum Finden solch eines Programms gibt (im wesentlichen deswegen nicht, weil keine obere Schranke für seine Laufzeit gefordert wird $\rightarrow$ Halteproblem), wird algorithmische Informationstheorie in der Regel nicht als ein praktikables Werkzeug zur Entdeckung von Regularitäten angesehen.

diagonal (w_L bezeichnet hier L's Gewichtsvektor). Damit läßt sich die Krümmung der Fehlermannigfaltigkeit durch Verfahren berechnen, welche dieselbe Komplexität aufweisen wie gewöhnliche Gradientenberechnung. Dies zieht effiziente Methoden zweiter Ordnung zur Beschleunigung des Lernvorgangs in L nach sich. Nichtlineare Netze sollten in ähnlicher Weise profitieren.

(3b) Kausaldetektion in Eingabeströmen. Diese neuartige Methode für dynamische Umgebungen kann enorme Ressourceneinsparungen für überwacht lernende sequenzverarbeitende Netze bieten, ihre detaillierte Ausführung bleibt aber dem 7. Kapitel überlassen (das vorliegende Kapitel beschränkt sich auf stationäre Umgebungen).

(4) Klassifizierung. Oft reicht es, Eingaben unter Informationsverlust geeignet zu *klassifizieren* und mit den Klassenrepräsentanten (statt mit den rohen Eingabedaten) weiterzuarbeiten. Die meisten Klassifikationsalgorithmen ordnen (im euklidischen Sinne) benachbarte Eingaben vorzugsweise in die gleiche Klasse ein. Implizite Voraussetzung ist dabei die Gleichwertigkeit 'semantischer' und 'syntaktischer' Ähnlichkeit.

(5) Extraktion vorhersagbarer nicht-trivialer Mustereigenschaften. Zielfunktionen zur Kreierung *vorhersagbarer* und dennoch (soweit als möglich) informationstragender Mustertransformationen können ebenfalls zur Bildung sinnvoller Eingabe*klassen* führen. In Abschnitt 5.5 werden wir beispielsweise sehen, wie durch Extraktion vorhersagbarer nicht-trivialer Mustereigenschaften aus Zufallsstereogrammen in unüberwachter Weise Information über 'stereoskopische Tiefe' gewonnen werden kann.

5.2 DAS INFOMAX-PRINZIP

Linskers *Infomaxpinzip* [49][50] zielt auf die Maximierung der wechselseitigen (Shannon-) Information [125] zwischen Ein- und Ausgabeensemble. Wir setzen eine endliche Zahl von Eingabevektoren x^p und Repräsentationsvektoren y^q voraus. Nach Shannon ist die wechselseitige Information zwischen Ein- und Ausgabe gleich

$$R = < ln \frac{P(y^q \mid x^p)}{P(y^q)} >, \tag{5.1}$$

wobei $< ... >$ den über das ganze Musterensemble genommenen Durchschnitt bezeichnet.

Es ist keine effiziente Methode zur Maximierung von (5.1) im Falle beliebiger azyklischer Netzarchitekturen, beliebiger Ausgabedimensionalität $dim(y)$ und beliebiger Eingabeensembles bekannt.

Betrachtet man allerdings ein Netzwerk, das nur über einen einzigen Ausgabeknoten verfügt, dessen Ausgabe y_1^p die Summe einer linearen Funktion des Eingabevektors x^p und eines nicht verschwindenden Rauschterms n ist, wobei sowohl die Eingabesignale als auch das Ausgaberauschen Gauss-Verteilungen unterliegen, so läßt sich eine kontinuierliche Version von (5.1) als

$$R = \frac{1}{2} ln(\frac{VAR(y_1)}{VAR(n)}) \tag{5.2}$$

schreiben [49]. Unter der Annahme, daß $VAR(n)$ fix bleibt, läßt sich R nun ohne weiteres mit der Kettenregel differenzieren. Der resultierende Lernalgorithmus ist äquivalent zur Maximierung der Ausgabevarianz. Interessanterweise stellt dies eine Variante der Hebbregel dar [33].

Wird nun zu jedem Eingabesignal Rauschen mit Varianz $V(n)$ addiert, so ergibt sich nach Linsker

$$R = \frac{1}{2} ln(\frac{VAR(y_1)}{VAR(n) \sum_i w_i^2}), \tag{5.3}$$

wobei w_i das Gewicht der Verbindung vom i-ten Eingabeknoten zum Ausgabeknoten ist. Der resultierende Lernalgorithmus ist äquivalent zur Maximierung der Ausgabevarianz unter gleichzeitiger Minimierung der Länge des Gewichsvektors, was der Analyse der prinzipiellen Komponenten des Eingabeensembles enstpricht (siehe hierzu auch [65][87][84][126][73]).

Experimente. [49] berichtet u.a., daß das Infomaxprinzip bei unstrukturierten Gauss-verteilten Eingaben im linearen Fall zu rezeptiven Feldern mit einer *'on-center/off-surround'*-Struktur führt, wie sie auch in biologischen Nervensystemen häufig beobachtet wird. Unter gewissen Voraussetzungen wurden auch Kohonen-artige topologische Karten [42] gefunden [50].

5.2.1 MEHRDIMENSIONALE AUSGABEN

Bei mehr als einem Ausgabeknoten wird die kontinuierliche Version von Gleichung (5.1) unter ansonsten gleichen Voraussetzungen komplizierter. Nun gilt nämlich

$$R = \frac{1}{2} ln(\frac{Det(Q(y))}{VAR(n)}), \tag{5.4}$$

wobei $Q(y)$ die Kovarianzmatrix der einzelnen Komponenten von y bezeichnet [124]. Der aus der Maximierung von (5.4) durch Anwendung der Kettenregel ableitbare Lernalgorithmus ist insofern aufwendig (und auch biologisch recht unplausibel), als die partiellen Ableitungen der Determinante von $Q(y)$ bezüglich aller y_i berechnet werden müssen. Der Effekt des Lernalgorithmus besteht in der *Dekorrelation* der Ausgaben, falls das

Rauschsignal geringe Varianz besitzt. Bei starkem Rauschen erhöht sich die Redundanz der beteiligten Repräsentationskomponenten.

Kapitel 6 wird eine neue Methode zur Redundanzbeseitigung vorstellen, die im Gegensatz zu obiger Methode weder Gauss-Verteilungen noch *lineare* Abbildungen vom Eingaberaum in den Repräsentationsraum voraussetzen muß. Diese Methode wird weiterhin nicht nur die Dekorrelation der Ausgabeknoten, sondern sogar das wesentlich stärkere Kriterium statistischer Unabhängigkeit erzwingen.

5.3 UNÜBERWACHTE BILDUNG LOKALER REPRÄSENTATIONEN

Dieser Unterabschnitt (ein kleiner originärer Beitrag dieser Arbeit) beschreibt eine Zielfunktion zur unüberwachten Bildung lokal in y repräsentierter Klassen von Eingabevektoren. Jeder Eingabevektor x^p soll dabei nach der Trainingsphase auf einen Binärvektor y^p mit einer einzigen Komponente $\neq 0$ abgebildet werden. Aus der Zielfunktion kann mit Hilfe der Kettenregel ein Lernverfahren abgeleitet werden, das mit herkömmlichen *kompetitiven unüberwachten Lernverfahren* (e.g. [153][42][31][86]) verwandt ist: Nur ein einziger Repräsentationsknoten (der 'Sieger') wird in Antwort auf die Präsentation eines Eingabemusters aktiviert. Es ist gerade die mathematische Herleitbarkeit der gradientenbasierten Methode, die sie im Rahmen der vorliegenden Arbeit interessant macht.

Vorteile lokaler Repräsentationen sind: *(1) Orthogonalität:* Jeder Repräsentationsvektor steht auf jedem anderen senkrecht (siehe auch [65][87][126]). *(2) 'Sparsame Codierung':* Wie auch häufig in biologischen Systemen beobachtet, ist i.a. nur ein geringer Bruchteil der Nichteingabeknoten aktiv. (1) und (2) ermöglicht einem zusätzlichen zielgerichteten Lerner schnelles Lernen. *(3) Verständlichkeit:* Im Gegensatz zu distribuierten Repräsentationen bereitet es einem menschlichen Beobachter nach der Lernphase i.a. keine Schwierigkeiten, die internen Repräsentationen zu analysieren.

Der schwerwiegende *Nachteil* lokaler Repräsentationen ist die mangelhafte Ausnützung der vorhandenen Ressourcen: Distribuierte Repräsentationen gestatten wesentlich höhere Repräsentationskapazität.

Zielfunktion. Wir maximieren die halbe Varianz

$$V = \frac{1}{2} \sum_p \sum_i (y_i^p - \bar{y_i})^2 \tag{5.5}$$

der Ausgabeknoten *unter der Nebenbedingung*[2]

$$\forall p : \sum_i y_i^p = 1 \ . \tag{5.6}$$

Es läßt sich zeigen, daß (5.5) unter der Nebenbedingung (5.6) genau dann maximal ist, wenn jede Klasse lokal durch genau eine Ecke des n-dimensionalen, von allen möglichen reellen Ausgabevektoren aufgespannten Hyperwürfels repräsentiert wird (falls genügend Ausgabeknoten vorhanden sind). Um dies intuitiv einzusehen, beachte man, daß alle möglichen Ausgabevektoren sowie ihr Mittelwert innerhalb dieses Hyperwürfels auf einer $n-1$-dimensionalen Hyperebene liegen, welche durch jene Ecken definiert wird, deren Distanz zum Ursprung gleich 1 ist [120] (siehe [75] für Details).

Bedingung (5.6) läßt sich dadurch erzwingen, daß man

$$y_i^p = \frac{u_i^p}{\sum_i u_i^p}$$

setzt, wobei u^p der Aktivationsvektor einer n-dimensionalen Lage versteckter Knoten ist, die als *un*normalisierte Ausgabelage angesehen werden kann.

Um eine gleichmäßige Verteilung der Eingabemuster zwischen den verschiedenen Klassenrepräsentanten zu erzielen, addieren wir einen zusätzlichen Term zu V und maximieren

$$V - \frac{\lambda}{2} \sum_i [\frac{1}{q} - \bar{y}_i]^2 \tag{5.7}$$

unter der Nebenbedingung (5.6), wobei λ eine positive Konstante bezeichnet. Dies ermutigt jeden Klassenrepräsentanten, nur bei einem Bruchteil $\frac{1}{dim(y)}$ aller möglichen Eingabemuster aktiv zu werden.

Man beachte die bedeutsamen Unterschiede zu Linskers Algorithmus der Varianzmaximierung eines *einzelnen* Knotens (Abschnitt 5.2). Das Verfahren weist jedoch eine gewisse Verwandtschaft zu Bridles und MacKays Verfahren auf [16]. Während sie die normalisierten Ausgabeaktivationen eines Knotens als Wahrscheinlichkeit interpretieren, daß die Eingabe in der entsprechenden Klasse liegt, maximieren Bridle und MacKay die Entropie des Ausgabemittelwerts minus dem Mittelwert der Ausgabeentropien.

[2]Der Nebenbedingung (5.6) kommt eine entscheidende Rolle zu. Beispiel: Man betrachte vier 4-dimensionale Eingabevektoren $(1,0,0,0)^T$, $(0,1,0,0)^T$, $(0,0,1,0)^T$, $(0,0,0,1)^T$. Wir nehmen an, daß die ersten beiden Eingabemuster auf das 4-dimensionale Ausgabemuster $(1,1,0,0)^T$ und die verbleibenden zwei Eingabemuster auf $(0,0,1,1)^T$ abgebildet werden. Dies liefert zwar maximale Varianz 4, maximiert jedoch nicht die Zahl der verschiedenen Musterklassifizierungen, und führt auch nicht zu lokaler Repräsentation. Die 'bessere' informationserhaltende (und Bedingung (5.6) erfüllende) Abbildung, welche jedes Muster auf sich selbst abbildet, besitzt lediglich Varianz 3.

Ihre Zielfunktion favorisiert ebenfalls lokale Klassenrepräsentationen mit gleichmäßiger Verteilung verschiedener Eingaben zwischen den Klassen.

Experimente. [120] und [75] enthalten die Details verschiedener Experimente zur bedingten Varianzmaximierung. Wir werden die Methode in den Experimenten des Abschnitts 5.5.3 verwenden.

5.4 UNÜBERWACHTES LERNEN UND STATISTISCHE UNABHÄNGIGKEIT

Ein hochgestecktes Ziel unüberwachten Lernens besteht darin, aus der Umgebung nicht-triviale, statistisch voneinander unabhängige Eigenschaften zu extrahieren. Dekomposition der Umgebungseingaben in statistisch unabhängige 'Objekte' führt zu sinnvoller Eingabesegmentierung, zu automatischer Redundanzelimination, zu beschleunigtem zielgerichteten Lernen, zur Vereinfachung der Aufgabe traditioneller Klassifizierverfahren aus der Statistik, und zu mehr (siehe 5.4.2).

5.4.1 G-MAX

Der G-Max Algorithmus [69] zielt darauf ab, in den Eingaben enthaltene Redundanz zu entdecken. G-Max maximiert im wesentlichen die Differenz zwischen der Wahrscheinlichkeitsverteilung P der Ausgaben eines azyklischen Netzes mit einem einzelnen binären probabilistischen Ausgabeknoten und der Verteilung Q, die man unter der Annahme, daß die Komponenten der Eingabevektoren statistisch unabhängig sind, erwarten würde. Der Ausgabeknoten kann nur die Werte 0 oder 1 annehmen. Die Wahrscheinlichkeit, daß er in Antwort auf x^p den Wert 1 annimmt, sei mit P^p bezeichnet. Die Wahrscheinlichkeit, daß er in Antwort auf x^p unter der Annahme, daß die Eingabekomponenten statistisch voneinander unabhängig sind, den Wert 1 annimmt, sei mit Q^p bezeichnet. Die zu maximierende Zielfunktion ist die sogenannte asymptotische Divergenz (oder auch Kullback-Distanz, siehe [44])

$$G = \sum_p P^p log \frac{P^p}{Q^p} + (1 - P^p) log \frac{1 - P^p}{1 - Q^p}. \tag{5.8}$$

Das Problem des G-Max-Algorithmus besteht darin, daß er nicht (zumindest nicht in offensichtlicher Weise) auf mehrdimensionale Ausgaben erweiterbar ist.

Experimente. Pearlmutter und Hinton berichten, daß G-max nach Training mit Bildern orientierter Balken wie schon Linskers Methode (Abschnitt

5.2) ebenfalls zu biologisch plausiblen *'on-center/off-surround'*-Strukturen führt [69].

5.4.2 FAKTORIELLE CODES

Das vielleicht ambitionierteste Ziel unüberwachten Lernens besteht in der Entdeckung sogenannter faktorieller Codes der Umgebungseingaben (auch faktorielle Repräsentationen genannt). Die an faktorielle Codes gestellten Anforderungen gehen über die vom Infomaxprinzip geforderten Bedingungen hinaus, indem sie außer Informationsmaximierung auch noch statistische Unabhängigkeit der Codesymbole fordern. Was genau ist ein faktorieller Code?

Wir setzen eine deterministische Abbildung von Eingabemustern x^p auf Repräsentationsvektoren y^p voraus. Jede Komponente y_i der Repräsentation y nimmt bei einem gegebenen Eingabeensemble mit Wahrscheinlichkeit $P(y_i = a)$ den Wert a an.

Bei einem faktoriellen Code ist die Wahrscheinlichkeit eines bestimmten Eingabemusters gleich dem Produkt der Wahrscheinlichkeiten der Komponenten des entsprechenden Repräsentationsvektors [4](daher die Bezeichnung 'faktoriell'):

$$\forall p: \quad P(x^p) = \prod_i P(y_i = y_i^p). \tag{5.9}$$

Ist ein auf deterministischem Wege gewonnener Code reversibel (also informationserhaltend), so ist die Abbildung von x^p auf y^p bijektiv und es gilt $\forall p: \quad P(x^p) = P(y^p)$. (5.9) läßt sich dann umformulieren:

$$\forall p: \quad P(y^p) = \prod_k P(y_k = y_k^p). \tag{5.10}$$

Die einzelnen Komponenten von y müssen also statistisch voneinander unabhängig sein.

Man beachte, daß die Bedingung der *statistischen Unabhängigkeit* weit stärker ist als die von bisher besprochenen Performanzmaßen erzwungene *Dekorrelationsbedingung*. Statistische Unabhängigkeit der Codesymbole zieht automatisch ihre Dekorrelation nach sich – die umgekehrte Richtung gilt nicht.

Vorteile faktorieller Codes. Welche Vorteile bietet die statistische Unabhängigkeit der Repräsentationskomponenten?

(1) *Verbesserung traditioneller statistischer Klassifikationsmethoden.* Viele in der Praxis verwendeten Klassifikationsalgorithmen machen aus Effizienzgründen die Annahme, daß die Komponenten der Eingabesignale statistisch voneinander unabhängig sind (e.g. [19], [25], [132]). Je weniger diese

Annahme zutrifft, desto schlechter die Performanz dieser Verfahren (e.g. [25]). Eine effiziente Methode zur Erstellung (nahezu) faktorieller Codes aus nicht-faktoriellen Eingaben wäre demzufolge interessant als Präprozessor für derartige traditionelle Verfahren.

(2) *Eingabesegmentierung.* Man stelle sich ein System vor, das visuelle 'retinale' Eingaben x^p aus der Umgebung erhält. Die x^p seien dabei Pixelvektoren, welche zweidimensionale Projektionen von Straßenszenen darstellen. Die x^p sind typischerweise aus 'Primitiven' wie Ecken, Kanten (auf höherer Ebene Autos) zusammengesetzt, deren Unterkomponenten in hohem Maße korreliert sind. So reicht z.B. ein kleiner Teil der von einem bestimmten Automobil verursachten Eingabekomponenten bereits aus, um mit überdurchschnittlicher Trefferwahrscheinlichkeit vom selben Fahrzeug verursachte physikalisch benachbarte Eingabekomponenten vorhersagen zu können. Ist das Heck blau, so sind vermutlich auch die Farbwerte der den Bug repräsentierenden Eingabepixel blau. Auf der anderen Seite mögen z.B. die Positionen zweier verschiedener Fahrzeuge wenig miteinander zu tun haben; Fahrzeugpositionen stellen damit in stärkerem Maße unabhängige Eigenschaften der Eingaben dar.

Statt nun bei einer Bildinterpretation eine alle Unterkomponenten umfassende Suche anzuwerfen, will man durch sinnvolle Segmentierung die Eingabekomponenten so gruppieren, daß der Berechnungsaufwand möglichst drastisch reduziert wird. Dies ist das treibende Motiv aller Segmentierungsalgorithmen. So macht es wenig Sinn, einen bestimmten Aspekt einer Eingabe zu repräsentieren, wenn dieser Aspekt nur redundante Information enthält, da er aus bereits repräsentierten Eingabeeigenschaften ableitbar ist.

Korrelationen zwischen Eingabekomponenten repräsentieren also entdeckenswerte 'Regularitäten'. Faktorielle Codes segmentieren nun die Umgebung in einer in gewissem Sinne *optimalen* Weise. Sie repräsentieren die Umgebung so, daß die möglichen Eingaben in statistisch voneinander unabhängige Eigenschaften (Abstraktionen, Konzepte) zerlegt werden. Will man einem Netzwerk beibringen, zu 'teilen und zu herrschen', so stellen Algorithmen zur Erlernung faktorieller Codes etwas Erstrebenswertes dar.

(3) *Occam's Rasiermesser.* In gewissem Sinne verkörpern faktorielle Codes automatisch 'Occam's Rasiermesser', welches 'einfache' Umgebungsmodelle komlexeren vorzieht. Das Maß der 'Einfachheit' ist hier durch die Anzahl der Codesymbole definiert, die zur Repräsentation der Umgebung durch statistisch unabhängige Konzepte vonnöten sind. Ist die Umgebung faktoriell codierbar, und übersteigt y's Repräsentationskapazität die Umgebungsentropie (die in den Umgebungseingaben enthaltene Shannon-Information), so erzwingt das Unabhängigkeitskriterium 'unbenützte' Codekomponenten, welche auf jedes beliebige Eingabemuster stets mit einem

mit Wahrscheinlichkeit 1 vorhersagbaren konstanten Wert reagieren.

(4)*Schnelles Lernen.* Liefert man einem überwacht lernenden linearen Netzwerk L Repräsentationen als Eingaben, deren Komponenten statistisch unabhängig sind, so ist die Hessematrix von L's Fehlerfunktion diagonal. Dies zieht effiziente Methoden zweiter Ordnung zur Beschleunigung des Lernvorgangs in L nach sich.

(5)*Neuigkeitsentdeckung.* Wie Barlow et. al [5] ausführen, bedeutet das plötzliche Auftreten statistischer Abhängigkeiten zweier bislang unabhängiger Codekomponenten die Entdeckung einer bisher unbekannten Assoziation (*'novelty detection'*).

Der bis vor kurzem einzige mir bekannte nicht-triviale Ansatz zur Entdeckung faktorieller Codes besteht in Heuristiken zur

MINIMIERUNG VON BITENTROPIESUMMEN

Barlow et. al [5] untersuchen das schwierige Problem, faktorielle Binärcodes zu finden. Jede Komponente von y darf dabei nur die Werte 0 oder 1 annehmen. Barlow et. al zeigen, daß die Minimierung der Bitentropiesumme

$$-\sum_i P(y_i = 1) log P(y_i = 1) - \sum_i P(y_i = 0) log P(y_i = 0)) \qquad (5.11)$$

äquivalent zum Finden eines der faktoriellen Codes ist, falls mindestens ein derartiger Code existiert.

Barlow et. al. geben einige 'nicht-neuronale' heuristische sequentielle Methoden an, um Codes mit niedriger Bitentropiesumme zu finden. Die einzige sichere bekannte sequentielle Methode besteht jedoch in der erschöpfenden Suche, einem Verfahren, das sich aufgrund des mit der Größe des Codes exponentiell wachsenden Aufwands von selbst verbietet (es ist durchaus möglich, daß das allgemeine Problem des Finden faktorieller Codes NP-vollständig ist).

Bis vor kurzem war auch kein 'neuronaler' unüberwachter Lernalgorithmus zur Entdeckung faktorieller Repräsentationen bekannt. Wegen der potentiellen Bedeutung derartiger Repräsentationen wird das nächste Kapitel jedoch zeigen, wie durch Umformulierung des Problems und Definition geeigneter Performanzmaße gradientenbasierte Algorithmen zur Auffindung faktorieller Codes hergeleitet werden können.

5.5 EXTRAKTION VORHERSAGBARER KONZEPTE

Viele Lernverfahren (u. a. auch die Weltmodellbauer aus Kapitel 4) basieren auf Untermodulen, die gewisse Muster aus anderen Mustern vorhersagen sollen. Häufig sind perfekte Vorhersagen jedoch prinzipiell unmöglich.

Informationstragende *Musterklassifikationen* hingegen mögen sehr wohl prophezeihbar sein. Unüberwachte Extraktion vorhersagbarer Konzepte (im folgenden auch *Vorhersagbarkeitsmaximierung* genannt – ein weiterer größerer originärer Beitrag dieser Arbeit, siehe auch [121]) beschäftigt sich mit dem Problem, *möglichst spezifische vorhersagbare* Musterklassifikationen zu finden.

Die folgenden vier Beispiele sollen dies näher erläutern.

Beispiel 1: Man gehe in den Tierpark, um sich den Elefanten anzuschauen. Es ist unmöglich, von vornherein alle Details wie z.B. seinen genauen Aufenthaltsort, sein exaktes Gewicht und die Schattierung seiner Hautfarbe vorherzusagen. Man kann sich jedoch darauf verlassen, daß sich der Elefant auf dem Boden befinden wird (statt in der Luft zu schweben) und daß er eher grau als pinkfarben sein wird. Eine *nützliche* Klassifikation wird die von beliebigen Elefanten verursachten Eingaben auf eine einzige vorhersagbare interne Repräsentation abbilden, die sich von der internen Repräsentation beliebiger Nilpferde unterscheidet (und damit informationstragend ist[3]).

Beispiel 2: Man werfe einen Spielwürfel. Die Fläche, auf der er zu liegen kommen wird, läßt sich gewöhnlich nicht vorhersagen – wohl aber die Tatsache, daß der Würfel, nachdem er zur Ruhe gekommen ist, nicht in der Luft hängen wird oder auf einer Ecke balancieren wird. Das Resultat einer Abbildung, die alle zur Ruhe gekommene Würfel auf dieselbe Klassenrepräsentation abbildet, ist vorhersagbar. Es sollte sich unterscheiden von vorhersagbaren Repräsentanten von, sagen wir, zur Ruhe gekommenen Kugelschreibern.

Beispiel 3: Sind die ersten beiden Wörter des Satzes 'Henrietta ißt Gemüse' bekannt, so läßt sich vorhersagen, daß das dritte Wort wohl für etwas Eßbares stehen wird, nicht aber, für welches Nahrungsmittel. Die Klasse des dritten Wortes ist aus den beiden vorangegangenen Wörtern prophezeihbar – die spezielle Instanz der Klasse ist es nicht. Die Klasse 'Eßbares' ist nicht nur vorhersagbar, sondern auch spezifisch in dem Sinne, daß sie nicht alles mögliche umfaßt – 'Ferdinand' ist beispielsweise keine Instanz von 'Eßbares'.

Das Problem besteht darin, Muster so zu klassifizieren, daß sie sowohl

[3]Eine triviale Klassifikation, die alle Eingaben auf denselben Repräsentationsvektor abbildet, wäre zwar stets vorhersagbar, aber nicht informationstragend.

vorhersagbar als auch nicht zu allgemein sind, um solcherart in der Umgebung verborgene Regelmäßigkeiten zu extrahieren. Eine allgemeine Lösung für dieses Problem wäre beispielsweise nützlich für die Entdeckung von Struktur in von unbekannten Grammatiken generierten Sätzen (wie im letzten Beispiel). Eine weitere wichtige Anwendung wäre die unüberwachte Klassifizierung von zur gleichen Klasse gehörigen Mustern, wie im folgenden erläutert werden soll.

Gegeben sei eine Menge von Paaren von Eingabemustern. Wir wissen, daß beide Muster eines Paares zur selben Klasse gehören. Wir wissen jedoch nichts über die Klassen selbst, wieviele Klassen es gibt, oder welche Muster zu welcher Klasse gehören. Das Ziel sei die Erstellung einer Abbildung von Eingabemustern auf Klassenrepräsentanten dergestalt, daß Muster, die zur selben Klasse gehören, auf gleiche Weise repräsentiert werden, während Muster, die zu verschiedenen Klassen gehören, auf unterschiedliche Weise repräsentiert werden. Wir wollen den Sinn dieses Ziels an einem weiteren praktischen Beispiel verdeutlichen:

Beispiel 4 (Stereoproblem, nach Becker und Hinton, 1989): Becker und Hinton beschreiben eine 'Stereoaufgabe', bei der es darum geht, aus zwei getrennten Pixelfeldern Stereoinformation zu extrahieren [11]. Eines der Pixelfelder wird mit einem zufällig generierten Bild belegt, während das andere Pixelfeld dasselbe Bild in um ein Pixel nach links oder rechts verschobener Position darstellt (siehe Abbildung 5.2). Pixel, die über den rechten Rand des Eingabefeldes hinausgeschoben werden, erscheinen auf der linken Seite des Feldes wieder (und umgekehrt). Zweideutige *'shifts'* werden ausgeschlossen. 8-dimensionale Eingabemuster werden durch Konkatenation eines 4-dimensionalen Streifens des linken Bildes mit dem entsprechenden 4-dimensionalen Streifens des rechten Bildes generiert. Betrachten wir nun *Paare* solcherart konstruierter, nicht überlappender Eingabemuster. Kennt man das erste Muster eines Paars, so kann man eine nicht-triviale Aussage über das zweite Muster machen (und umgekehrt), denn beiden Mustern ist etwas gemeinsam, nämlich die Information über die 'stereoskopische Tiefe'. Diese ist gleichzeitig die einzige nicht-triviale Eigenschaft, die beiden Eingaben zu eigen ist. Das Ziel besteht darin, ohne irgendetwas über das Konzept 'stereoskopische Tiefe' erzählt zu bekommen, diese einzige nicht-triviale wechselseitig vorhersagbare Eigenschaft zu finden und Eingabemuster mit gleicher stereoskopischer Tiefe auf dieselbe Klasse abzubilden, während Eingabemuster mit unterschiedlicher stereoskopischer Tiefe auf unterschiedliche Klassen abzubilden sind.

Die Beispiele $1-3$ sind Vertreter des sogenannten *asymmetrischen Falles*, während Beispiel 4 eine Instanz des sogenannten *symmetrischen Falles* darstellt. Welche Architekturen und Zielfunktionen eignen sich nun zum Finden möglichst informativer und doch gleichzeitig vorhersagbarer Einga-

betransformationen?

Im einfachsten Fall basiert der hier vorgeschlagene Ansatz zur unüberwachten Extraktion vorhersagbarer Klassifikationen auf zwei neuronalen Netzwerken T_1 und T_2. Beide lassen sich als BP-Netze implementieren [143][46][66][85] (siehe auch Kapitel 1). T_1 sieht bei einem gegebenen Paar von Eingabemustern das erste Muster, T_2 sieht das zweite Muster. Konzentrieren wir uns zunächst auf den asymmetrischen Fall: Im Falle des Beispiels 3 sieht T_1 beispielsweise eine Repräsentation der Wörter 'Henrietta ißt', während T_2 eine Repräsentation des Wortes 'Gemüse' als Eingabe bekommt. T_2's Aufgabe besteht darin, seine Eingabe zu klassifizieren. T_2's Aufgabe besteht nicht etwa darin, T_2's rohe Eingabe zu prophezeihen, sondern statt dessen T_2's *Ausgabe*.

Beide Netze besitzen q Ausgabeknoten. $p \in \{1, \ldots, m\}$ indiziere die Trainingsmuster. T_2 beantwortet einen Eingabevektor $x^{p,2}$ durch die Klassifikation $y^{p,2} \in [0, \ldots, 1]^q$. T_1 beantwortet einen Eingabevektor $x^{p,1}$ durch die *Vorhersage* $y^{p,1} \in [0, \ldots, 1]^q$ der Klassifikation $y^{p,2}$.

Wir sehen uns zwei miteinander im Konflikt stehenden Zielen gegenüber: (A) Alle Vorhersagen $y^{p,1}$ sollen den entsprechenden Klassifikationen $y^{p,2}$ gleichkommen. (B) Die $y^{p,2}$ sollen diskriminierend sein – verschiedene Eingaben $x^{p,2}$ sollen verschiedene Klassifikationen $y^{p,2}$ nach sich ziehen.

Wir drücken den Konflikt zwischen (A) und (B) durch zwei gegensätzliche Zielfunktionen aus, welche gleichzeitig minimiert werden sollen.

(A) wird durch einen Fehlerterm M (für '*M*atch') ausgedrückt:

$$M = \sum_{p=1}^{m} \|y^{p,1} - y^{p,2}\|^2. \tag{5.12}$$

(B) wird durch einen zusätzlichen Fehlerterm D_2 (für '*D*iskriminierung') repräsentiert, welchen (im asymmetrischen Fall) nur T_2 zu minimieren braucht. Im nächsten Unterabschnitt werden wir D_2 so wählen, daß signifikante Euklidische Abstände zwischen Klassifikationen unterschiedlicher Eingabemuster ermutigt werden. Es gibt mehr als einen vernünftigen Weg, D_2 zu definieren – der nächste Unterabschnitt wird vier *alternative* Möglichkeiten mit verschiedenen Vor- und Nachteilen erwähnen.

T_2's zu minimierende Zielfunktion ist

$$\epsilon M + (1 - \epsilon) D_2, \tag{5.13}$$

wobei wobei $0 < \epsilon < 1$ eine positive Konstante bezeichnet.

Im asymmetrischen Fall ist T_1's zu minimierende Zielfunktion

$$\epsilon M. \tag{5.14}$$

.

.

Abbildung 5.1: Ein Zufallsstereogramm, bestehend aus zwei Bildern zu je 4×9 Pixeln. Das linke Bild wird zufällig generiert, das rechte Bild ergibt sich aus dem linken durch Verschiebung um ein Pixel nach rechts. Pixel, die über den rechten Rand des rechten Bildes hinausgeschoben werden, erscheinen auf der linken Seite des Feldes wieder.

Die Zielfunktionen werden wie üblich durch Gradientenabstieg minimiert. Dies zwingt einerseits die Klassifikationen, sich den Vorhersagen anzugleichen, und andererseits in symmetrischer Weise die Vorhersagen, sich den Klassifikationen anzugleichen, während die Klassifikationen gleichzeitig ermutigt werden, möglichst aussagekräftig zu sein. Abbildung 5.1 zeigt ein entsprechendes aus einem Prediktor und einem Klassifikator bestehendes System, welches sich zur Implementierung von D_2 eines Autoassoziators bedient (siehe auch den folgenden Unterabschnitt 5.5.1).

Gelegentlich (siehe das Stereoexperiment aus Beiapiel 4) ist es angebracht, T_1 und T_2 in symmetrischer Manier zu behandeln. In solchen Fällen dienen *beide* Netzwerke als Klassifikatoren. Jeder der beiden Klassifikatoren 'sieht' ein Eingabemuster eines Paars. Die Klassifikatoren sollen ohne Lehrer lernen, die nicht-trivialen wechselseitig vorhersagbaren Eigenschaften in ihren Ausgaben zu repräsentieren. Um solchen symmetrischen Problemen in natürlicher Weise zu begegnen, brauchen wir T_1's Zielfunktion nur leicht durch Einführung eines zusätzlichen 'Diskriminationsterm' D_1 für T_1 zu modifizieren: Beide $T_l, l = 1, 2$ minimieren[4] nun

$$\epsilon M + (1 - \epsilon) D_l, \tag{5.15}$$

wobei vier alternative Möglichkeiten zur Definition von D_l im nächsten Unterabschnitt aufgelistet werden.

Gewichtsteilung. Sollen beide Klassifikatoren gleiche Eingaben mit gleichen Ausgaben beantworten (dies gilt z. B. für obige Stereoaufgabe), so läßt sich das Verfahren weiter vereinfachen. Dann genügt es nämlich, für beide Klassifikatoren dieselben Gewichtsparameter zu verwenden – dies reduziert die Anzahl der freien Parameter und erhöht demzufolge die Generalisierungsperformanz [10].

5.5.1 METHODEN ZUR DEFINITION VON D_l

D_l läßt sich durch verschiedene in diesem Kapitel bereits erwähnte Methoden (mit verschiedenen wechselseitigen Vor- und Nachteilen) festlegen.

INFOMAX

D_l kann die wechselseitige Information zwischen Ein- und Ausgaben von T_l sein (siehe Abschnitt 5.2). Aus den folgenden Gründen wurde Infomax

[4] Man könnte zusätzliche 'Vorhersagenetzwerke' einführen – eines, um $y^{p,2}$ aus $y^{p,1}$ vorherzusagen, und ein weiteres, um $y^{p,1}$ aus $y^{p,2}$ vorherzusagen. Dann ließen sich Zielfunktionen zur Erzwingung wechselseitiger Vorhersagbarkeit konstruieren (statt die im wesentlichen gleichwertige Funktion M zu verwenden). Dies würde die Allgemeinheit des Ansatzes allerdings nicht erhöhen, sondern lediglich unnötige zusätzliche Komplexität nach sich ziehen.

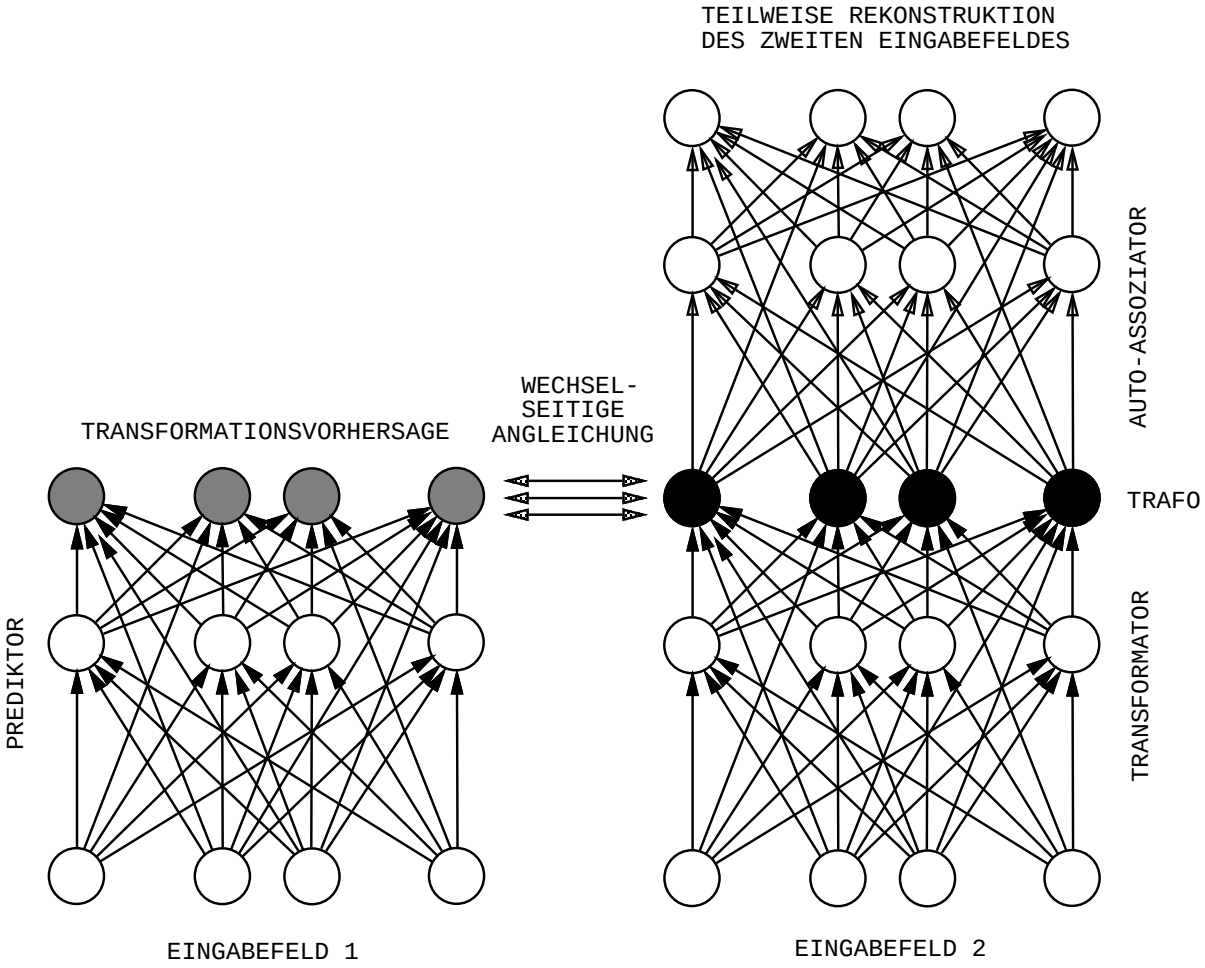

Abbildung 5.2: Ein Autoassoziator findet Verwendiung, um eine möglichst informationstragende Eingabeklassifikation in den schwarzen Knoten zu erzielen. Ein Prediktornetz versucht, aus einer unterschiedlichen (z.B. früheren) Eingabe in seiner Ausgabe (graue Knoten) die interne Klassifikation möglichst gut vorherzusagen, während sich die Klassifikation gleichzeitig möglichst weitgehend der Vorhersage anpaßt.

in den später zu beschreibenden Experimenten *nicht* verwendet: (a) Es ist keine effiziente und allgemeine Methode zur Maximierung wechselseitiger Information bekannt. (b) Infomax würde bei unserem grundlegenden Ansatz nur dann Sinn machen, wenn es automatisch hohe Varianz der Ausgaben von T_l zur Folge hätte. Dies ist zwar der Fall bei den vereinfachenden, von Linsker studierten Rauschmodellen. Für den allgemeinen Fall trifft dies allerdings nicht zu. (c) Selbst unter Annahme geeigneter Gaussscher Rauschmodelle erfordert Infomax bei nichtskalaren, vektorwertigen Klassifikationen die Maximierung von Funktionen der Determinante der Kovarianzmatrix der Klassifikationskomponenten (DETMAX, siehe Abschnitt 5.2.1).

AUTOASSOZIATOREN

Ein Rekonstruktionsmodul A^l erhält $y^{p,l}$ als Eingabe. Die Kombination von T_l und A^l wird mittels BP trainiert, die Rekonstruktion $h^{p,l}$ der externen Eingabe $x^{p,l}$ von T_l auszugeben. Die grundlegende Architektur ist die eines Autoassoziators. D_l läßt sich dabei als der Rekonstruktionsfehler

$$D_l = \frac{1}{2} \sum_p \|h^{p,l} - x^{p,l}\|^2. \tag{5.16}$$

definieren. Man werfe erneut einen Blick auf Abbildung 5.2.

Autoassoziatoren sind einfach zu implementieren. Ein Nachteil der Methode liegt jedoch darin, daß die vorhersagbare Information eines Eingabemusters unter Umständen trotz Nichttrivialität nichts dazu beiträgt, den Rekonstruktionsfehler zu minimieren (als Beispiel diene das Stereoexperiment aus Abschnitt 5.5).

BEDINGTE VARIANZMAXIMIERUNG

Siehe Abschnitt 5.3. Vorteile dieser Methode sind, daß sie einfach zu implementieren ist, und daß die resultierenden Klassenrepräsentationen dank ihrer Lokalität für den Beobachter i.a. leicht verständlich sind. Ein Nachteil ist die mangelhafte Ausnutzung der Repräsentationskapazität der Ausgabeknoten der Klassifikatoren.

VORHERSAGBARKEITSMINIMIERUNG

D_l kann eine differenzierbare Zielfunktion zur Gewinnung reversibler faktorieller Codes (siehe Abschnitt 5.4.2) aus den Eingaben von T_l sein. Das nächste Kapitel zeigt, daß sich solch ambitionierte Performanzmaße in der Tat konstruieren lassen. Die entsprechende Technik trägt den Namen 'Vorhersagbarkeitsminimierung'.

5.5.2 BEZUG ZU FRÜHEREN ARBEITEN: IMAX

Becker und Hinton (1989) lösen symmetrische Probleme durch Maximierung der wechselseitigen Information zwischen den Ausgaben von T_1 und T_2 (IMAX). Dies dient ebenfalls zur Auffindung informativer und dennoch wechselseitig vorhersagbarer Eingabeklassifikationen.

Die zu maximierende Zielfunktion ist

$$< ln\ P(y^{p,1}, y^{p,2}) > - < ln\ P(y^{p,1}) > - < ln\ P(y^{p,2}) >, \qquad (5.17)$$

wobei $P(y^{p,k})$ die Wahrscheinlichkeit dafür bezeichnet, daß die Ausgabe von T_k den Wert $y^{p,k}$ annimmt ($< ... >$ steht wieder für den Ensemblemittelwert).

Ist die Ausgabe jedes Klassifikators eindimensional, so wird (5.17) unter gewissen vereinfachenden Gaussschen Annahmen zu

$$R = \frac{1}{2} ln(\frac{VAR(y^1)}{VAR(y^1 - y^2)}).$$

Becker und Hinton berichten allerdings, daß ihr mit probabilistischen Ausgabeknoten ausgestattetes System die oben beschriebenen Stereoaufgabe nur dann zu lösen imstande war, wenn der Algorithmus in sukzessiven *'bootstrap'*-Stufen angewendet wurde. Auch mußte die Lernrate während der Lernphase geeignet adjustiert werden. Schließlich erwies sich eine obere Schranke für die maximale Gewichtsänderung pro Gradientenanstiegsiteration als erforderlich.

Keine derartigen Tricks waren notwendig, um dieselbe Aufgabe mit unserem alternativen System (siehe Abschnitte 5.5.3 und 6.6.5) zu lösen.

In [35] wird nur der Fall eindimensionaler Ausgaben beider Klassifikatoren betrachtet. In [155] beschäftigen sich Zemel und Hinton hingegen mit der Möglichkeit, daß jeder Klassifikator über mehr als einen Ausgabeknoten verfügt. Es wird zunächst von neuem die einschränkende Annahme Gaussverteilter Signale gemacht. Dann läßt sich (5.17) wieder umformulieren – zu maximieren bleibt

$$R = \frac{1}{2} ln(\frac{Det(Q(y^1 + y^2))}{Det(Q(y^1 - y^2))}, \qquad (5.18)$$

wobei $Q(z)$ wieder die Kovarianzmatrix der einzelnen Komponenten von z bezeichnet [124]. Zemel und Hinton berechnen explizit die partiellen Ableitungen von $Det(Q(.))$ bezüglich aller y_i, was wie Linskers Methode (Abschnitt 5.2.1) aufwendig (und auch vom Standpunkt des Biologen her unplausibel) ist.

5.5.3 EXPERIMENTE ZUR VORHERSAGBARKEITS-MAXIMIERUNG

[120] und [75] enthalten beschreiben verschiedene Experimente zur Vorhersagbarkeitsmaximierung. So war es beispielsweise möglich, die in Abschnitt 5.5 beschriebene Stereo-Aufgabe [35] durch Vorhersagbarkeitsmaximierung (mit durch Vorhersagbarkeits*minimierung* definiertem D_l) bei 100-prozentiger Erfolgsquote innerhalb kürzerer Zeit zu lösen als mittels Beckers und Hintons Methode, welche noch dazu bestimmte 'Tricks' erforderte (siehe auch Abschnitt 6.6.5).

In untenstehenden Experimenten wurden Mittelwerte $\bar{y}_i^l$ annäherungsweise durch die Formel

$$\hat{y}_i^l \leftarrow 0.95\hat{y}_i^l + 0.05y_i^l,$$

berechnet, wobei $\hat{y}_i^l$ die Approximation von $\bar{y}_i^l$ nach Beobachtung von y^l ist. Im Falle lokaler Ausgaberepräsentation (Abschnitt 5.3) wurde $\hat{y}_i^l$ mit $\frac{1}{q}$ initialisiert, ansonsten mit 0.5.

VORHERSAGBARE LOKALE KLASSENREPRÄSENTANTEN

Aufgabe 1. Motiviert durch Beispiel 3, Abschnitt 5.5 wurde ein einfacher 'Satzgenerator' konstruiert, welcher aus 2 Symbolen bestehende 'Sätze' ausgeben konnte. Jede Alternative in der folgenden Grammatik kam dabei mit gleicher Wahrscheinlichkeit vor. S diente als Startsymbol, Kleinbuchstaben als Terminalzeichen und Großbuchstaben als Nichtterminalzeichen.

$$S \rightarrow A|B|C|D,\ \ A \rightarrow aa^1|aa^2,\ \ B \rightarrow bb^1|bb^2,\ \ C \rightarrow cc^1|cc^2,\ \ D \rightarrow dd^1|dd^2.$$

Während des Trainings sah T_1 das erste Symbol eines zufällig gewählten legalen Satzes, während T_1 das zweite Symbol wahrnahm. T_1 benötigte hierzu 2 Eingabeknoten für seine 4 möglichen durch 2-dimensionale binäre Vektoren repräsentierten Eingaben, T_2 benötigte 3 Eingabeknoten für seine 8 möglichen durch 3-dimensionale binäre Vektoren repräsentierten Eingaben.

Sowohl T_1 als auch T_2 besaßen 4 versteckte Knoten und 6 Ausgabeknoten – zwei mehr als notwendig, um die 4 vorhersagbaren Klassen

$$\{a^1, a^2\},\ \ \{b^1, b^2\},\ \ \{c^1, c^2\},\ \ \{d^1, d^2\}$$

lokal zu repräsentieren.

10 Testläufe mit $\epsilon = 0.25, \lambda = 1$, Vorhersagbarkeitsmaximierung gemäß (5.13) und (5.14), D_2 definiert durch beschränkte Varianzmaximierung gemäß (5.5) und (5.6), einer Lernrate von 1.0, und 15000 Musterpräsentationen

wurden durchgeführt. Alle Experimente waren erfolgreich – stets emittierte T_2 nach dem Training 4 lokale Klassenrepräsentanten in Antwort auf Elemente der 4 vorhersagbaren Klassen, wobei die beiden überflüssigen Ausgabeknoten immer ausgeschaltet blieben.

Es wurden keine Versuche unternommen, den Lernvorgang zu beschleunigen.

Obiges Experiment zeigt eine Anwendung der Methode auf den asymmetrischen Fall. Das folgende Experiment zeigt eine Anwendung der Methode auf den symmetrischen Fall (wie beim Stereoexperiment).

Aufgabe 2. Zwei Eigenschaften eines 4-dimensionalen binären Eingabevektors sind die Wahrheitswerte folgender Ausdrücke:

1. Die 'rechte' Hälfte des Eingabevektors enthält mehr Einsen als die 'linke' Hälfte.

2. Der Eingabevektor verfügt über mehr Einsen als Nullen.

Eingabevektoren mit gleich viel Einsen und Nullen sowie Eingabevektoren mit gleicher Anzahl von Einsen auf beiden Seiten seien ausgeschlossen. Es verbleiben 2 mögliche Eingabevektoren für jede mögliche Eigenschaftskombination. Das Ziel besteht in der Generierung unterschiedlicher Repräsentationen der 4 möglichen wechselseitig vorhersagbaren Eigenschaftskombinationen. Außer diesen Eigenschaftskombinationen soll nichts repräsentiert werden.

Während einer Trainingsiteration nahm T_1 einen zufällig gewählten legalen Eingabevektor wahr. Ein weiterer legaler Eingabevektor, zufällig gewählt aus der Menge derjenigen Vektoren mit der Eigenschaftskombination des ersten, wurde T_2 dargeboten. T_1 und T_2 besaßen einen gemeinsamen Gewichtssatz (siehe auch 5.5.2) sowie 4 Eingabeknoten und 4 Ausgabeknoten.

10 Testläufe mit $\epsilon = 0.25, \lambda = 1$, Vorhersagbarkeitsmaximierung gemäß (5.15), D_2 definiert durch beschränkte Varianzmaximierung gemäß (5.5) und (5.6), einer Lernrate von 1.0, und 5000 Musterpräsentationen wurden durchgeführt. Alle Experimente waren erfolgreich – stets emittierten sowohl T_1 als auch T_2 nach dem Training 4 lokale Klassenrepräsentanten in Antwort auf die 4 vorhersagbaren Eigenschaftskombinationen.

Wie schon erwähnt, besteht der Nachteil lokaler Repräsentationen in der mangelhaften Ausnutzung vorhandener Speicherkapazität (Vorteil ist jdeoch u.a. die einfache Interpretierbarkeit der Ergebnisse). Folgender Test soll demonstrieren, daß sich Vorhersagbarkeitsmaximierung bei entsprechender Umdefinition von D_l (mit Hilfe eines Autoassoziators) auch zur Extraktion distribuierter Repräsentationen mehr als einer Eingabeeigenschaft eignet.

VORHERSAGBARE DISTRIBUIERTE REPRÄSENTATIONEN

Wir verwenden wieder *Aufgabe 2* aus dem letzten Unterabschnitt. Da bei distribuierter binärer Repräsentation der 4 möglichen wechselseitig vorhersagbaren Eigenschaftskombinationen im Gegensatz zu lokaler Repräsentation nur 2 Ausgabevariablen vonnöten sind, besaßen T_1 und T_2 bei diesem Experiment nur noch 2 (statt 4) Ausgabeknoten sowie 4 versteckte Knoten.

Trainingsiterationen wurden abgewickelt wie im vorherigen Abschnitt beschrieben. 10 Testläufe mit 15,000 Musterpräsentationen und Vorhersagbarkeitsmaximierung gemäß (5.15) wurden durchgeführt. Dabei war D_l durch einen Autoassoziator definiert (5.16), A_l besaß 4 versteckte Knoten, ϵ war gleich 0.1, und alle Lernraten waren gleich 0.5. Das System fand stets eine distribuierte, nahezu binäre Repräsentation der 4 möglichen vorhersagbaren Eigenschaftskombinationen.

Das gleiche Experiment wurde mit vergleichbarem Erfolg durchgeführt, wobei D_l jedoch durch Vorhersagbarkeitsminimierung (Abschnitt 5.5.1, siehe auch das nächste Kapitel) statt durch einen Autoassoziator definiert war.

STEREO-EXPERIMENT

Das in Abschnitt 5.5.1 sowie in [11] beschriebene Stereoexperiment diente zum Vergleich von IMAX und Vorhersagbarkeitsmaximierung.

Zunächst muß hervorgehoben werden, daß die vorhersagbare Information eines Eingabemusters unter Umständen trotz Nichttrivialität nichts dazu beiträgt, den Rekonstruktionsfehler eines Autoassoziators zu minimieren – beim Stereoexperiment beispielsweise scheidet der Autoassoziator als Möglichkeit zur Implementierung von D_l aus.

Daher wurde statt eines Autoassoziators die nicht auf den Rekonstruktionsfehler angewiesene Methode der Vorhersagbarkeits*minimierung* verwendet, um D_l zu definieren. Da dieses Verfahren erst im nächsten Kapitel detailliert beschrieben wird, sparen wir auch die Details des Experiments für das nächste Kapitel (Abschnitt 6.6.5) auf.

Es sei jedoch bereits vorweggenommen, daß das Experiment zeigen wird, daß es Beispiele gibt, bei denen sich Performanzmaße zur Informationsmaximierung besser als Bestandteil der Vorhersagbarkeitsmaximierungsprozedur nach (5.15) eignen als das Rekonstruktionsfehlermaß.

5.5.4 SCHLUSSBEMERKUNGEN ZU 5.5

Verglichen mit den bisherigen Implementierungen von Beckers und Hintons IMAX-Verfahren weisen die oben vorgestellten Methoden zur Extraktion vorhersagbarer Konzepte folgende Vorteile auf: (1) Sie tendieren dazu, einfacher anwendbar zu sein (e.g., inkrementelles 'bootstrapping' sukzessiver

Lagen ist nicht erforderlich). (2) Sie fordern keine Gaussschen Annahmen über die Verrauschtheit der Ein- und Ausgabesignale. (3) Sie fordern nicht die Maximierung von Funktionen der Determinante der Kovarianzmatrix der Klassifikationskomponenten (DETMAX). (4) Wie in Kapitel 6 noch zu sehen sein wird, entsteht durch geeignete Definition von D_l (im Gegensatz zu DETMAX) ein Potential zur Auffindung von Klassifikationen mit statistisch voneinander unabhängigen Komponenten (siehe auch Abschnitt 5.4.2). (5) Die Beantwortung der Frage, ob die Aktivationsmuster für Klassifikation und Vorhersage ein und dasselbe repräsentieren, wird durch die neue Methodik trivial. (6) Im direkten experimentellen Vergleich sind gewisse unüberwachte Lernaufgaben innerhalb kürzerer Zeit zu lösen als mittels IMAX, welches noch dazu bestimmte 'Tricks' erfordert (siehe Abschnitt 6.6.5).

Es bleibt jedoch experimentell zu untersuchen, ob das Verfahren auch bei komplexeren 'Echtweltanwendungen' zu guten Ergebnissen führen kann.

Kapitel 6

VORHERSAGBARKEITS-MINIMIERUNG

Faktorielle Codes sind reversible Codes, bei denen das Auftreten jedes Codesymbols statistisch unabhängig vom Auftreten der übrigen Codesymbole ist. Im letzten Kapitel (Abschnitt 5.4.2) haben wir gesehen, welche Vorteile faktorielle Repräsentationen der Umgebungseingaben bieten können. Hier seien einige davon nochmals in knapper Form aufgelistet:

(1) *Verbesserung der Eingaberepräsentation für konventionelle Klassifikationsmethoden der Statistik.* Viele Klassifikationsalgorithmen für praktische Anwendungen beruhen auf der Annahme statistischer Unabhängigkeit der Eingabesignalkomponenten (e.g. [19], [25], [132]). Je weniger diese Annahme zutrifft, desto schlechter die Performanz dieser Verfahren (siehe beispielsweise [25] für eine experimentelle und theoretische Analyse).

(2) *Eingabesegmentierung.* Faktorielle Codes stellen in gewisser Weise das *Non-plus-ultra* der Segmentierung dar. Sie repräsentieren die Umgebung so, daß die möglichen Eingaben in statistisch voneinander unabhängige Eigenschaften (Abstraktionen, Konzepte) zerlegt werden. Dies entspricht dem höchst sinnvollem Aspekt des 'Teilens' beim nicht nur von Informatikern erstrebten Ziel des 'Teilens und Herrschens'.

(3) *Redundanzelimination.* Faktorielle Codes verkörpern automatisch 'Occams Rasiermesser', welches 'einfache' Modelle der Umgebung komplexeren Modellen vorzieht. Das Maß der 'Einfachheit' ist hier durch die Anzahl der Codesymbole definiert, die zur Repräsentation der Umgebung vonnöten sind.

(4)*Schnelles Lernen.* Liefert man einem überwacht lernenden linearen Netzwerk faktorielle Repräsentationen als Eingaben, so ist die Hessema-

trix seiner Fehlerfunktion diagonal. Damit lassen sich effiziente Methoden zweiter Ordnung zur Beschleunigung des Lernvorgangs im überwacht lernenden Netzwerk anwenden. Nichtlineare Netze sollten in ähnlicher Weise profitieren. Auch verkleinert die mit redundanzfreien Codierungen einhergehende Kompaktheit im allgemeinen den Suchraum für nachgeschaltete zielgerichtete Lerner.

Bisher gab es keine 'neuronale' Methoden zum Finden faktorieller Codes, und auch die 'nicht-neuronalen' sequentiellen Ansätze sind entweder unrealistisch (erschöpfende Suche) oder basieren auf Heuristiken [5]. Es existieren zwar Methoden zur Dekorrelation von Netzwerkausgaben (e.g. [49], [155], [65], [87], [27], [84], [49], [126], siehe auch Kapitel 5), diese sind jedoch auf den linearen Fall beschränkt und fassen nicht das höhergesteckte Ziel der statistischen Unabhängigkeit ins Auge. Außerdem beruhen einige dieser Methoden auf der (oft unrealistischen) Annahme Gauss-verteilter Eingaben und erfordern dabei auch noch die explizite Berechnung der Ableitungen der Determinante der Kovarianzmatrix der Ausgabeknoten [124].

Im vorliegenden Kapitel soll daher demonstriert werden, wie durch die Wahl einer geeigneten Architektur und passender Zielfunktionen mit der Kettenregel gradientenbasierte Algorithmen zum Finden faktorieller Codes hergeleitet werden können.

Es sollte jedoch angemerkt werden, daß es keine große Überraschung wäre, wenn sich herausstellen sollte, daß das allgemeine Problem der Entdeckung von Codes mit statistisch unabhängigen Komponenten NP-vollständig ist. In diesem Fall dürfte man von gradientenbasierten Methoden nicht erwarten, für beliebige Umgebungen stets beim ersten Versuch völlig redundanzfreie Repräsentationen zu finden. Wie auch in den bisherigen Kapiteln werden wir nicht versuchen, das allen nicht-linearen Gradientenverfahren anhaftende Problem der lokalen Minima und Maxima anders als durch wiederholte Suchen (ausgehend von zufällig gewählten Startpunkten) anzugreifen. Wir fokussieren uns statt dessen wieder auf der Methodik des Algorithmenentwurfs sowie dem daraus erwachsenden zentralen *Prinzip der Vorhersagbarkeitsminimierung* [111].

Dieses Prinzip führt zu einem System, das aus zwei Arten 'sich bekämpfender' Module besteht. Die Module der ersten Art versuchen, die Aktivität jedes an der Coderepräsentation beteiligten Netzwerkknotens möglichst exakt aus den Aktivitäten der übrigen Knoten vorherzusagen. Die Module der zweiten Art hingegen versuchen, durch Ausnützung der statistischen Eigenschaften der Umgebung ihre Eingaben so zu transformieren, daß sie von den Modulen der ersten Art möglichst *schlecht* vorhersagbar sind.

Die später zu beschreibenden Experimente (unter anderem zur Redundanzverminderung der Repräsentation von Buchstaben mit durch die Statistik der englischen Sprache gegebenen Auftretenswahrscheinlichkeiten) zei-

gen, daß das Prinzip der Vorhersagbarkeitsminimierung in der Tat praktisch anwendbar ist.

6.1 PROBLEMFORMULIERUNG

Wir setzen n verschiedene adaptive eingabeverarbeitende sogenannte *Repräsentationsmodule* voraus. Jedes Modul sieht zu einem gegebenen Zeitpunkt denselben Eingabevektor. Die Ausgabe jedes Moduls wird durch eine Menge neuronenartiger Knoten geliefert. Wir konzentrieren uns auf den einfachsten Fall: Ein Ausgabeknoten (auch *Repräsentationsknoten* oder *Codeknoten* genannt) pro Modul. Das i-te Modul produziert als Antwort auf den p-ten anliegenden externen Eingabevektor x^p einen skalaren Ausgabewert $y_i^p \in [0, 1]$.

Die im vorliegenden Kapitel beschriebenen Methoden sind zumeist daraufhin angelegt, binäre oder zumindest *quasi-binäre* Codes finden (die einzige Ausnahme findet sich in Abschnitt 6.3.5). Jede an einem quasi-binären Code teilhaftige Codevariable liefert entweder einen konstanten Wert für alle Eingabemuster, oder sie nimmt für jedes gegebene Eingabemuster entweder den Wert 0 oder den Wert 1 an. Binäre Codes sind also ein Spezialfall quasi-binärer Codes. Da wir wie immer mit differenzierbaren Aktivationen hantieren wollen, werden wir unsere quasi-binären Codes ausgehend von reellwertigen Codes schaffen.

Ein binärer faktorieller Code muß drei Kriterien erfüllen:

1. *Das Binärkriterium*: Keine Codevariable darf Werte außer 1 oder 0 annehmen.

2. *Das Reversibilitätskriterium*: Es muß möglich sein, die Eingabe aus ihrer Codierung zu rekonstruieren. Ist die Umgebung zu komplex (oder zu 'verrauscht'), um ohne Informationsverlust in der von der Kapazität her beschränkten internen Repräsentation codiert zu werden (i.e., im Falle binärer Codes bei mehr als $2^{dim(y)}$ Eingabemustern), wollen wir das Reversibilitätskriterium abschwächen. In diesem Falle sollten die internen Repräsentationen immer noch soviel Information wie möglich über die Eingaben übermitteln. Der Schwerpunkt dieses Kapitels liegt jedoch auf Situationen wie die von Barlow studierten [5]: Rauschfreie Umgebungen bei hinreichender Kapazität in y, um alle Eingaben eindeutig und informationsverlustfrei zu repräsentieren. Unter diesen Voraussetzungen impliziert das Reversibilitätskriterium Linskers Infomax-Prinzip (Abschnitt 5.2).

3. *Das Unabhängigkeitskriterium*: Die Wahrscheinlichkeit, daß ein beliebiges Codesymbol einen bestimmten Wert annimmt, soll unabhängig von den Werten aller anderen Codesymbole sein. Ist das Binärkriterium erfüllt, so können wir das Unabhängigkeitskriterium wie folgt formulieren: Wir for-

dern, daß

$$E(y_i \mid \{y_k, k \neq i\}) = P(y_i = 1 \mid \{y_k, k \neq i\}) = P(y_i = 1) = E(y_i). \quad (6.1)$$

Der Ausdruck (6.1) impliziert, daß y_i nicht von $\{y_k, k \neq i\}$ abhängt. Mit anderen Worten: $E(y_i \mid \{y_k, k \neq i\})$ läßt sich aus einer Konstante mit Informationsgehalt Null berechnen. Man beachte, daß bei *reellwertigen* Codes die Bedingung $E(y_i \mid \{y_k, k \neq i\}) = E(y_i)$ nicht notwendigerweise die statistische Unabhängigkeit der y_i gewährleistet.

6.2 GRUNDPRINZIP UND GRUNDLEGENDE ARCHITEKTUR

Für jeden Repräsentationsknoten $i = 1, \ldots, n$ führen wir ein zusätzliches azyklisches adaptives (im allgemeinen nicht-lineares) Prediktornetzwerk P_i ein. Beim p-ten Eingabevektor x^p ist P_i's Eingabevektor die $(n-1)$-dimensionale Konkatenation aller Ausgaben y_k^p aller Repräsentationsknoten $k \neq i$. P_i sieht also alle Repräsentationsknoten außer dem von P_i vorherzusagenden Knoten mit Nummer i. Siehe Abbildung 6.1. P_i's ein-dimensionale Ausgabe P_i^p wird mittels BP daraufhin trainiert, sich dem bedingten Erwartungswert $E(y_i \mid \{y_k^p, k \neq i\})$ anzugleichen. Dieses Ziel kann dadurch erreicht werden, daß man P_i die altbekannte Zielfunktion

$$E_{P_i} = \frac{1}{2} \sum_p (P_i^p - y_i^p)^2 \quad (6.2)$$

minimieren läßt[1].

Das ist schon die grundlegende Topologie, die für alle im Rest dieses Kapitels auftretenden Performanzmaße und Algorithmen gleichbleiben wird. Die n Prediktoren dienen dazu, die Redundanz unter den Repräsentationsknoten zu messen. Mit Hilfe der n (bezüglich ihrer Eingaben differenzierbaren) Prediktoren lassen sich nun *zusätzliche* differenzierbare Zielfunktionen für die Repräsentationsknoten definieren (siehe Abschnitte 6.3 und 6.4), so daß diese gedrängt werden, den drei Kriterien aus Abschnitt 6.1 Genüge zu tun und faktorielle Eingabecodierungen zu entdecken. Allen diesen zusätzlichen Zielfunktionen für die Repräsentationsmodule ist eines gemeinsam: Sie zwingen die Codeknoten, die wechselseitige Vorhersagbarkeit durch die Prediktoren zu minimieren.

[1] Bisher haben wir von dieser MSE-Eigenschaft keinen Gebrauch gemacht – alle vorangegangenen Kapitel verwendeten MSE im wesentlichen zur Funktionsapproximation, nicht zur Modellierung von Erwartungswerten. Das *Kreuzentropiemaß* stellt eine zum Erreichen desselben Ziels geeignete Alternative dar.

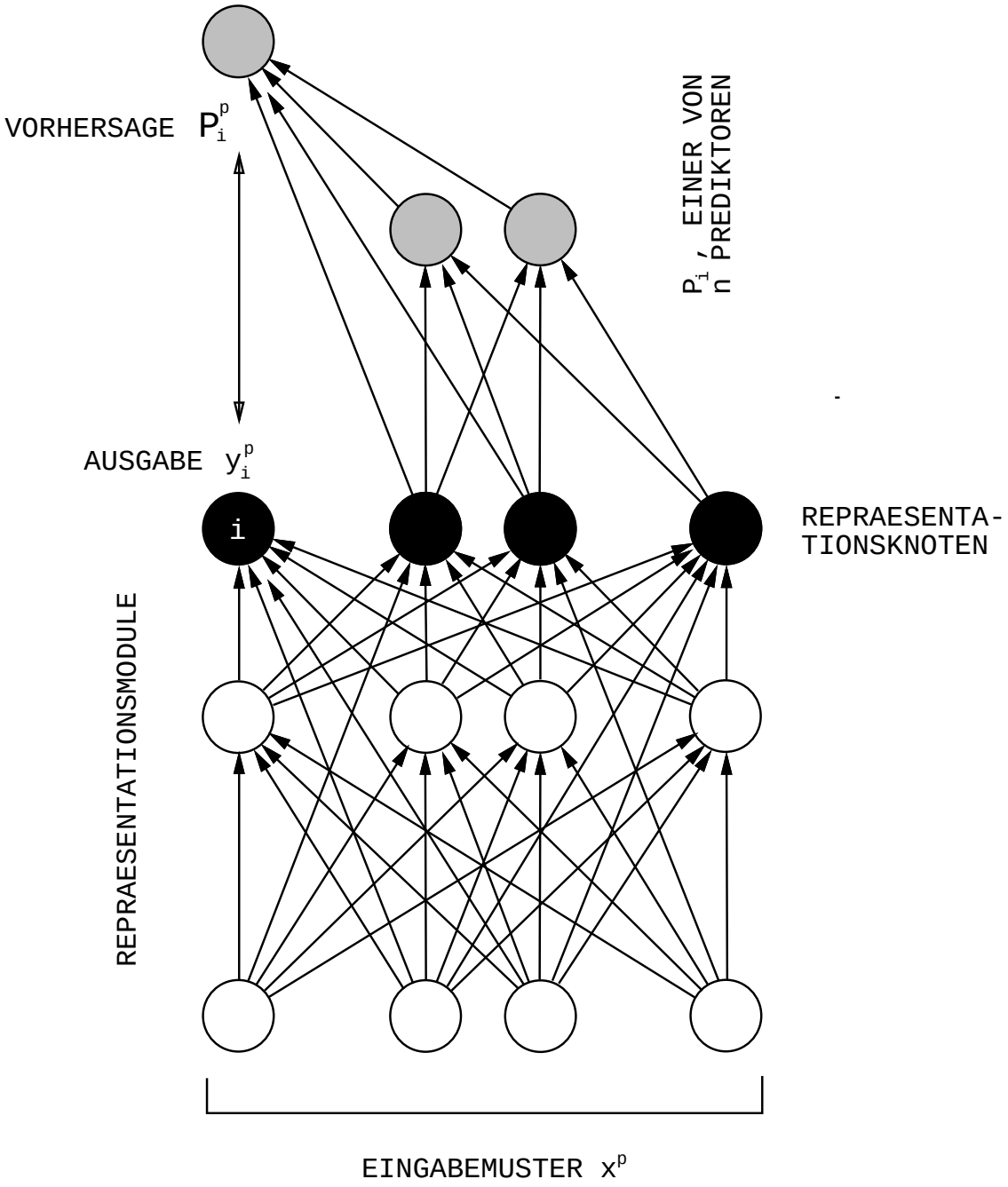

Abbildung 6.1: Die n Repräsentationsknoten (schwarz) liefern eine Codierung der gegewärtigen Eingabe. Jeder der n Prediktoren (in der Abbildung ist nur einer zu sehen (graue Knoten)) bemüht sich, aus je $n-1$ Repräsentationsknoten den verbleibenden vorherzusagen. Die Repräsentationsknoten ihrerseits wehren sich unter Ausnutzung der Eingabestatistik durch geeignete Gewichtsänderungen gegen Vorhersagbarkeit.

Jeder Repräsentationsknoten versucht dabei, aus der Umgebung irgendwelche Eigenschaften zu extrahieren, so daß keine Kombination von $n-1$ Knoten Information (im Shannonschen Sinne) über den verbleibenden Knoten trägt. Mit anderen Worten: Keine auf der Kombination von $n-1$ Repräsentationsknoten basierende Vorhersage des verbleibenden Knotens sollte höhere Qualität aufweisen als eine Vorhersage, die *ohne* Wissen über die $n-1$ Knoten auskommt. Im folgenden werde ich dieses Prinzip das *Prinzip der intra-repräsentationellen Vorhersagbarkeitsminimierung* oder kürzer das *Prinzip der Vorhersagbarkeitsminimierung* nennen.

Dem Prinzip der Vorhersagbarkeitsminimierung folgend versucht jeder Repräsentationsknoten, die statistischen Eigenschaften der Umgebung dergestalt auszunützen, daß er sich selbst vor Vorhersagbarkeit schützt. Jedes Repräsentationsmodul 'will' sich auf Aspekte der Umgebung konzentrieren, die unabhängig von denjenigen abstrakten Umgebungseigenschaften sind, auf die sich die Aufmerksamkeit der restlichen Module lenkt.

6.3 PERFORMANZMASSE FÜR DIE DREI KRITERIEN

Die Unterabschnitte 6.3.1, 6.3.2 und 6.3.3 liefern Zielfunktionen für die drei Kriterien aus Abschnitt 6.1. Die Unterabschnitte 6.3.4, 6.3.5, und 6.3.6 zeigen, wie sich die Zielfunktionen in unterschiedlicher Weise kombinieren lassen. Unterabschnitt 6.3.7 weist auf ein mit den vorangegangenen Zielfunktionen verknüpftes Problem sorgfältiger Parameterwahl hin. Schließlich beschreibt Abschnitt 6.4 eine dieses Problem umgehende Methode zur Vorhersagbarkeitsminimierung (meine bevorzugte Methode).

6.3.1 EINE ZIELFUNKTION FÜR DAS UNABHÄNGIGKEITSKRITERIUM

Wir wollen für den Augenblick annehmen, daß die Prediktoren P_i zu allen Zeiten bis zur Perfektion trainiert werden, was bedeutet, daß P_i stets den bedingten Erwartungswert $E(y_i \mid \{y_k^p, k \neq i\})$ von y_i liefert (bei gegebenen Ausgaben der übrigen Repräsentationsmodule). Im Falle quasi-binärer Codes ist das folgende Performanzmaß H genau dann gleich Null, wenn das Unabhängigkeitskriterium erfüllt ist:

$$H = \frac{1}{2} \sum_i \sum_p [P_i^p - \bar{y_i}]^2 . \tag{6.3}$$

Dieser Term für wechselseitige Redundanzminimierung zielt darauf ab, die Ausgaben statistisch voneinander unabhängig zu machen. Hier existiert

eine Verwandtschaft zu Linskers Dekorrelationsmethode durch Maximierung der Determinante der Kovarianzmatrix der Repräsentationsknoten unter Annahme Gauss-verteilter Signale [49]. Letztere Methode zielt jedoch lediglich auf die Vermeidung *linearer* Abhängigkeiten, während die Minimierung von (6.3) aufgrund der von den allgemeinen Prediktoren modellierbaren Nichtlinearitäten auch nicht-lineare Redundanz beseitigen kann (sogar *ohne* Voraussetzung Gauss-verteilter Eingaben).

6.3.2 EINE ZIELFUNKTION FÜR DAS BINÄRKRITERIUM

Eine wohlbekannte Zielfunktion V zur Erzwingung binärer Ausgaben ist durch

$$V = \frac{1}{2} \sum_i \sum_p (\bar{y}_i - y_i^p)^2 \tag{6.4}$$

gegeben. V ist maximal, wenn jeder Repräsentationsknoten ausschließlich binäre Werte annimmt. Der Beitrag, den ein Knoten i für V liefert, ist dann maximal, wenn $E(y_i)$ so nahe an $\frac{1}{2}$ liegt wie möglich. Dies zieht unter der Binärbedingung maximale Entropie des Knotens i nach sich; das Repräsentationsmodul i übermittelt dabei maximale (Shannon-)Information über seine Eingaben.

6.3.3 EINE ZIELFUNKTION FÜR DAS REVERSIBILITÄTSKRITERIUM

Um den in den Repräsentationsknoten dargestellten Code der Umgebungseingaben reversibel zu machen, läßt sich folgende altbekannte Methode verwenden: Für ein anliegendes Muster x^p erhält ein *Rekonstruktionsmodul* (ein azyklisches BP-Netzwerk) die Konkatenation aller $y_i^p, i = 1, \ldots, n$ als Eingabe und wird daraufhin trainiert, seinen Ausgabevektor z^p dem anliegenden Eingabevektor x^p anzugleichen. Die zugrundeliegende Struktur ist die eines Autoassoziators (siehe Abbildung 6.2). Die zu minimierende Zielfunktion ist

$$I = \frac{1}{2} \sum_p (z^p - x^p)^T (z^p - x^p). \tag{6.5}$$

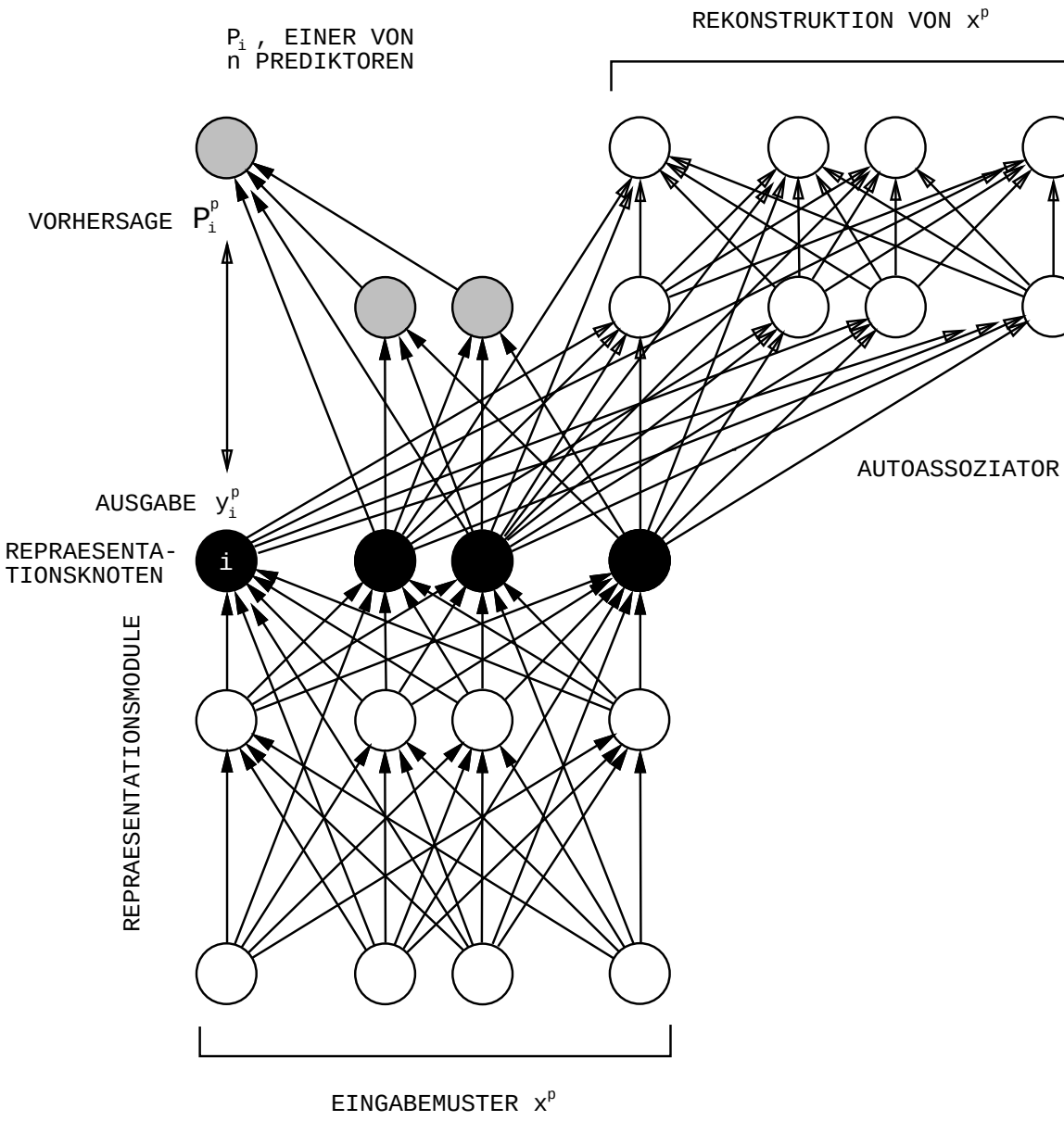

Abbildung 6.2: Wie Abbildung 6.1, mit zusätzlichem Autoassoziator zur Erzielung reversibler Codes in den Repräsentationsknoten. Im Text wird gezeigt, daß der Autoassoziator nicht unbedingt nötig ist.

6.3.4 ZIELFUNKTIONSKOMBINATIONEN

Der geradlinige Ansatz, die gleichzeitige Erfüllung aller drei Kriterien zu ermutigen, besteht darin, das Gesamtperformanzmaß

$$T = \alpha V - \beta I - \gamma H \tag{6.6}$$

zu maximieren. α, β, γ sind dabei nicht-negative Konstanten, die die relative Gewichtung der teilweise miteinander im Konflikt stehenden Terme bestimmen. Sind α, β, γ positiv, so ermutigt T's Maximierung die Repräsentationsknoten, stets binäre Werte anzunehmen, so daß wechselseitige Unabhängigkeit maximiert wird, während gleichzeitig der Rekonstruktionsfehler minimiert wird.

6.3.5 REELLWERTIGE CODES

Möchte man bei einer spezifischen Anwendung die überlegene repräsentationelle Kapazität reellwertiger (statt binärer) Knoten ausschöpfen, und ist man mit dekorrelierten (statt tatsächlich statistisch unabhängigen) Knoten zufrieden, dann kann man den V-Term aus (6.6) entfernen, indem man $\alpha = 0$ setzt. Es bleibt die Minimierung von

$$\beta I + \gamma H. \tag{6.7}$$

Man beachte, daß bei reellwertigen Aktivationen das Reversibilitätskriterium dank der unbeschränkten Informationskapazität des Intervalls zwischen 0 und 1 theoretisch stets mit einem einzigen Repräsentationsknoten erfüllt werden kann. Das Unabhängigkeitskriterium ermutigt in diesem Fall alle übrigen Repräsentationsknoten dazu, konstante (stets mit Wahrscheinlichkeit 1 vorhersagbare) Werte in Antwort auf alle Eingabemuster auszugeben. In verrauschten Umgebungen mag es sich jedoch als vorteilhaft erweisen, die Eingabe in mehr als einem Repräsentationsknoten darzustellen, wie bereits von Linsker im Kontext seines Infomax-Prinzips bemerkt wurde [49].

6.3.6 ENTFERNUNG DES GLOBALEN INVERTIBILITÄTSTERMS

Man kann zur Gewinnung binärer Codes auf den Autoassoziator verzichten und in (6.6) $\beta = 0$ setzen. In diesem Fall bliebe die Maximierung von

$$T = \alpha V - \gamma H.$$

Warum sollte dies zur Erzielung von Reversibilität ausreichen? Die Minimierung des H-Terms verhindert, daß verschiedene Repräsentationsknoten

dieselbe (und damit redundante) Information über den gegenwärtigen Eingabevektor liefern. Jeder Repräsentationsknoten wird lokal ermutigt, seine Entropie zu maximieren, während er gleichzeitig versucht, Information aus der Umgebung zu extrahieren, die *nicht* bereits in anderen Repräsentationsknoten enthalten ist. Reversible Codes werden also favorisiert, obwohl es *keinen globalen* Reversibilitätsterm mehr gibt.

6.3.7 EIN NACHTEIL OBIGER VERFAHREN

Die im Unterabschnitt 6.3.4 vorgestellte Linearkombination der für die drei Kriterien zuständigen Zielfunktionen scheint mit einem Nachteil behaftet: Ein faktorieller Code verursacht bei $\alpha > 0$ *nicht-maximales* V und somit auch *nicht-maximales* T (abgesehen von seltenen Fällen, wie z.B. bei 2^n gleich wahrscheinlichen unterschiedlichen Eingabemustern). Dies bedeutet, daß die relativen Gewichtungsparameter bei einem gegebenen Problem unter Umständen sorgfältig *'getunt'* werden müssen. Die elegante Methode des nächsten Abschnitts vermeidet diese Notwendigkeit sorgfältiger Parameterwahl, indem sie den Term für Varianzmaximierung durch einen prediktorbasierten Term für *bedingte* Varianzmaximierung ersetzt.

6.4 LOKALE BEDINGTE VARIANZMAXIMIERUNG

Im folgenden stelle ich das von mir bevorzugte Verfahren zur Implementierung des Prinzips der Vorhersagbarkeitsminimierung vor. Es leidet nicht unter dem soeben in Abschnitt 6.3.7 erwähnten Parameterwahlproblem. Außerdem weist die Methode eine bemerkenswerte Symmetrie zwischen 'sich bekämpfenden' Modulen (vorhersagenden Prediktoren sowie den Vorhersagen ausweichenden Repräsentationsmodulen) auf.

Wir definieren

$$V_C = \frac{1}{2} \sum_i \sum_p (P_i^p - y_i^p)^2 = E_P. \tag{6.8}$$

Zur Erinnerung: Es wird angenommen, daß $P_i^p = E(y_i \mid \{y_k^p, k \neq i\})$. Man beachte die formale Äquivalenz von V_C und der Summe der Zielfunktionen der Prediktoren $\sum_i E_{P_i}$ (vergleiche (6.2)).

Wie in Abschnitt 6.3.6 wird auf den Autoassoziator *verzichtet*. Nun definieren wir die Gesamtzielfunktion T der Repräsentationsknoten wie folgt um:

$$T = V_C - \gamma H. \tag{6.9}$$

Ich will nun die (nicht restlos bewiesene) Vermutung aufstellen, daß man in (6.9) sogar auf den für das Unabhängigkeitskriterium zugeschnittenen H-Term verzichten (also $\gamma = 0$ setzen) darf:

Vermutung 6.4.1. *Existiert für ein gegebenes Ensemble von Eingabemustern unter den möglichen reellwertigen Codierungen ein quasi-binärer faktorieller Code, so ist das Gesamtperformanzmaß*

$$T = V_C \tag{6.10}$$

genau dann maximal, wenn die Repräsentationsmodule einen derartigen Code gefunden haben. Es genügt also, wenn alle Repräsentationsmodule versuchen, dieselbe Zielfunktion zu maximieren, *welche von den Prediktoren* minimiert *wird.*

Für den allgemeinen Fall bleibt diese Vermutung unbewiesen. Im Appendix zu diesem Kapitel wird sie allerdings für gewisse Spezialfälle mathematisch gerechtfertigt. Der Appendix bietet auch etwas intuitive Rechtfertigung für den allgemeinen Fall. Schließlich liefert der Appendix eine auf Peter Dayan und Richard Zemel zurückgehende Argumentation dafür, daß das Verfahren aus Abschnitt 6.3.6 und Vorhersagbarkeitsminimerung gemäß Abschnitt 6.4 für $\alpha = 1, \gamma = 1$ Kraft im wesentlichen äquivalent sind.

Außerdem haben sich Algorithmen, die ausschließlich auf der Maximierung von (6.10) beruhen, auch in den später zu beschreibenden Experimenten bewährt.

6.5 'NEURONALE' IMPLEMENTIERUNG

In praktischen Anwendungen ist die Annahme, daß der Erwartungswert der Fehler aller P_i stets minmal ist, unplausibel. Nach jeder Modifikation der Repräsentationsmodule müssen die P_i für geraume Zeit trainiert werden, um sich der neuen Situation anzupassen.

Das möglicherweise vorhandene Autoassoziationsmodul sowie jedes der n Prediktonsmodule und der n Repräsentationsmodule läßt sich als azyklisches BP-Netzwerk implementieren. Zum Training sind zwei alternierende Phasen vonnöten:

PHASE 1 (Minimierung der Prediktionsfehler):
Wiederhole für eine 'hinreichende' Anzahl von 'Trainingsepochen':
1. *Für alle p:*
   1.1. *Berechne alle* y_i^p.

1.2. Berechne alle P_i^p.

1.3. Berechne für alle Gewichte w *in* P_i *mittels BP den Wert*

$$\frac{\partial (P_i^p - y_i^p)^2}{\partial w}.$$

2. Ändere alle Gewichte w *jedes Prediktors* P_i *gemäß*

$$\triangle w = -\eta_P \frac{1}{2} \sum_p \frac{\partial (P_i^p - y_i^p)^2}{\partial w},$$

wobei η_P *die positive Lernrate der Prediktormodule bezeichnet.*

PHASE 2:

1. Für alle p:

1.1. Berechne alle y_i^p.

1.2. Berechne alle P_i^p.

1.3. Falls ein Autoassoziator verwendet wird, berechne z^p.

2. Ändere alle Gewichte v *jedes Repräsentationsmoduls* i *mittels BP gemäß*

$$\triangle v = -\eta_R \sum_p \frac{\partial}{\partial v} T(x^p),$$

wobei $T(x^p)$ *der Beitrag der Präsentation des Eingabevektors* x^p *zur Gesamtzielfunktion* T *ist, und* η_R *die positive Lernrate der Repräsentationsmodule bezeichnet. Dazu muß zunächst mit Hilfe der kurzfristig 'eingeforenen' Prediktoren und BP für alle Repräsentationsknoten* i *der Wert* $\sum_p \frac{\partial}{\partial y_i^p} T(x^p)$ *berechnet werden. Auch* $\sum_p \frac{\partial y_i^p}{\partial v}$ *läßt sich durch BP bestimmen. Mit der Kettenregel ergibt sich*

$$\triangle v = -\eta_R \sum_p \sum_i \frac{\partial}{\partial y_i^p} T(x^p) \frac{\partial y_i^p}{\partial v}.$$

Die Gewichte der P_i ändern sich also während der zweiten Phase *nicht*, finden jedoch trotzdem Verwendung, um Gradienten für die Gewichte der Repräsentationsmodule zu berechnen. Fehlersignale für die letzteren werden demgemäß mittels Fehlerpropagierung *durch* die *Eingabe*knoten der Prediktoren (welche ja gleichzeitig Ausgabeknoten der Repräsentationsmodule sind) gewonnen. Dies erinnert an die Art und Weise, in der in Kapitel 4 (allerdings in höchst unterschiedlichem Kontext) differenzierbare adaptive 'Weltmodelle' zur Berechnung von Gradienten für das eigentlich inter-

essierende 'Steuernetzwerk' verwendet werden. Auch der Subzielgenerator desselben Kapitels bedient sich eines vergleichbaren Tricks.

Es sollte erwähnt werden, daß einige oder auch alle der Repräsentationsmodule sich versteckte Knoten teilen dürfen. Dasselbe gilt für die Prediktionsmodule. Prediktoren mit gemeinsamen versteckten Knoten müssen ihre Ausgaben allerdings sequentiell berechnen – kein Repräsentationsknoten darf dazu verwendet werden, seine eigene Aktivität vorherzusagen.

Obiger Algorithmus stellt eine *'off-line'*-Version dar. Gewichte ändern sich erst nach Präsentation des gesamten Eingabemusterensembles; den Prediktoren wird stets eine 'hinreichende' Anzahl von Trainingsbeispielen angeboten, um mit den Repräsentationsmodulen Schritt zu halten. Die *'off-line'*-Version ist möglicherweise weniger attraktiv als eine *'on-line'*-Version, bei der (1) Eingabemuster zufällig angeboten werden, (2) Gewichtsänderungen sofort nach jeder Musterpräsentation stattfinden, und (3) Prediktoren und Repräsentationsmodule weitgehend simultan lernen. Bei solch einer *'on-line'*-Version führen allerdings sowohl die Prediktoren als auch die Repräsentationsmodule Gradientenabstieg in sich ändernden Funktionen durch. Wieviel derartige *'on-line'*-Interaktion gestattet werden darf, bleibt experimentellen Auswertungen überlassen. Bei den im nächsten Abschnitt zu berichtenden Experimenten verursachte die *'on-line'*-Version keine größeren Schwierigkeiten.

[104] betrachtet auch den Fall stochastischer Repräsentationsknoten, der uns jedoch im Rahmen dieser Arbeit nicht weiter interessieren soll.

6.6 EXPERIMENTE

Alle im folgenden beschriebenen Experimente basieren auf der Definition von $T = V_C$ (siehe Abschnitt 6.4). Die Repräsentationsknoten versuchen also, dasselbe Performanzmaß zu maximimieren, das die Prediktoren minimieren wollen.

Alle Repräsentationsmodule und Prediktoren wurden als 3-lagige BP-Netze implementiert. Alle versteckten Knoten und alle Ausgabeknoten benutzten eine logistische Aktivierungsfunktion und wiesen zudem eine Verbindung von einem stets mit 1 aktivierten 'wahren' Knoten auf. Parameter wie Lernraten und die Anzahl versteckter Knoten wurden nicht systematisch in Bezug auf Lerngeschwindigkeit optimiert – es geht hier vielmehr um die Demonstration, daß Vorhersageminimierung in der Tat praktisch anwendbar ist.

Jeff Rink (Student an der Universität Colorado) und Daniel Prelinger (Diplomand an der TUM) implementierten *on-line* und *'off-line'* Versionen des in Abschnitt 6.5 angegebenen Verfahrens (siehe Details in [104]). Der

Zweck dieser Sektion besteht nicht darin, die *on-line*- mit der *'off-line'*-Version zu vergleichen, sondern zu zeigen, daß beide zu befriedigenden Resultaten führen können.

Bei der *'off-line'*-Version wurde der Wert 5 als 'hinreichend' für die Zahl der in PHASE 1 durchzuführenden Trainingsepochen für die Prediktoren angesehen. Lernraten um 0.3 erwiesen sich sowohl für die Prediktoren als auch für die Repräsentationsmodule als zweckmäßig.

Bei der *'off-line'*-Version wurden die beiden Lernphasen wie folgt modifiziert: Pro Phase wurde nur ein einziges Eingabemuster aus dem Ensemble präsentiert, dasselbe Muster wurde dabei sowohl in PHASE 1 als auch in PHASE 2 angeboten. In Abschnitten 6.6.1 und 6.6.2 wurde eine zusätzliche Modifikation zur Vermeidung gewisser lokaler Minima eingeführt (siehe [104]). Es galt $\eta_P = 1.0, \eta_R = 0.1$ (die Prediktoren lernten also '10 mal schneller' als die Repräsentationsmodule).

Bei allen Experimenten wurde ein Knoten als binär angesehen, wenn die absolute Differenz zwischen jeder von ihm angenommenen Aktivation und entweder der maximalen oder der minimalen von seiner Aktivierungsfunktion gestatteten Aktivation den Wert 0.05 nie überstieg.

Die nächsten Unterabschnitte beschreiben Experimente mit beiden Versionen. Der Ausdruck 'lokale Eingaberepräsentation' soll dabei $dim(x)$ verschiedene binäre Eingabevektoren der Länge 1 implizieren. Der Ausdruck 'verteilte Eingaberepräsentation' bedeutet $2^{dim(x)}$ verschiedene binäre Eingabevektoren,

Bei vielen Experimenten stellte sich heraus, daß die Existenz versteckter Knoten in der Tat zu besserer Performanz führt. Der Grund ist leicht einzusehen: Man betrachte den Fall $dim(y) = 3$. Angenommen, es existiert eine XOR-artige Beziehung zwischen den Aktivationen der ersten beiden Repräsentationsknoten und der Aktivation des dritten Repräsentationsknotens. Ein *linearer* Prediktor wäre außerstande, diese Beziehung aufzudecken. Demzufolge sähen die Repräsentationsmodule keine Veranlassung, die nicht-lineare Redundanz zu beseitigen.

6.6.1 EXPERIMENTE MIT GLEICHVERTEILTEN EINGABEN

Bei den in diesem Unterabschnitt beschriebenen Experimenten besteht das Eingabeensemble aus $2^{dim(y)}$ uniform verteilten Eingabevektoren. Daher sind die erwünschten faktoriellen Codes bei maximalem $T = V_C$ gerade die alle möglichen Binärmuster umfassenden Binärcodes. Der *un*bedingte Erwartungswert der Aktivation jedes Repräsentationsknotens ist im Erfolgsfall gleich 0.5, was dem vom entsprechenden Prediktor als Antwort auf jedes Eingabemuster emittierten *bedingten* Erwartungswert entspricht.

Experiment 1: 'off-line', $dim(y) = 2$, $dim(x) = 4$, lokale Eingaberepräsentation, 3 versteckte Knoten pro Prediktor, 4 allen Repräsentationsmodulen gemeinsame versteckte Knoten. 10 Testläufe mit 20000 Trainingsepochen für die Repräsentationsmodule wurden durchgeführt. In 8 Fällen genügte dies zur Findung eines faktoriellen Binärcodes.

Experiment 2: on-line, $dim(y) = 2$, $dim(x) = 2$, verteilte Eingaberepräsentation, 2 versteckte Knoten pro Prediktor, 4 allen Repräsentationsmodulen gemeinsame versteckte Knoten. 10 Testläufe wurden durchgeführt. Weniger als 3000 Musterpräsentationen (äquivalent zu ca. 700 Epochen) reichten dabei stets aus, um einen faktoriellen Binärcode zu entdecken.

Experiment 3: 'off-line', $dim(y) = 4$, $dim(x) = 16$, lokale Eingaberepräsentation, 3 versteckte Knoten pro Prediktor, 16 allen Repräsentationsmodulen gemeinsame versteckte Knoten. 10 Testläufe mit 20000 Trainingsepochen für die Repräsentationsmodule wurden durchgeführt. In einem Fall fand das System einen invertiblen faktoriellen Code. In 4 Fällen kreierte es einen 'nahezu' faktoriellen Code mit 15 verschiedenen binären Ausgabemustern als Antworten auf die 16 Eingabemuster. In drei Fällen kreierte es einen Code mit 14 Ausgabemustern, und in 2 Fällen einen mit 13. In allen Fällen wurde zumindest eine in hohem Maße nicht-redundante Codierung gefunden.

Experiment 4: on-line, $dim(y) = 4$, $dim(x) = 4$, verteilte Eingaberepräsentation, 6 versteckte Knoten pro Prediktor, 8 allen Repräsentationsmodulen gemeinsame versteckte Knoten. 10 Testläufe wurden durchgeführt. In allen Fällen außer einem genügten weniger als 4000 Musterpräsentationen (äquivalent zu weniger als 300 Epochen) zur Entdeckung eines faktoriellen Binärcodes mit 16 verschiedenen binären Ausgabemustern als Antworten auf die 16 Eingabemuster.

6.6.2 EXPERIMENTE MIT 'OCCAMS RASIERMESSER'

Eingangs wurde erwähnt, daß Methoden zum Finden faktorieller Codes automatisch 'Occams Rasiermesser' verkörpern, welches 'einfache' Umgebungsmodelle komplexeren vorzieht. Das Maß der 'Einfachheit' ist in unserem Fall durch die Anzahl der Repräsentationsknoten definiert, die zur eindeutigen Repräsentation der Umgebungseingaben erforderlich sind. Die hier vorgestellten Experimente sollen die Wirksamkeit von Occams Rasiermesser verifizieren.

Es sei darauf hingewiesen, daß bei Musterensembles, die keine faktorielle Codierung besitzen, Vorhersagbarkeitsminimierung die Anzahl der verwendeten Codesymbole reduziert, anstatt (wie Barlow *et al.*s im vorangegangenen Kapitel erwähnte Methode) die Bitentropiesumme zu minimieren.

Dies wird einsichtig, wenn man eine von Mitchison im Appendix von [5] beschriebene Testaufgabe betrachtet. Bei diesem Beispiel läßt sich die Bitentropiesumme durch *lokale* Repräsentation der Eingaben minimieren. Lokale Repräsentationen *maximieren* jedoch die wechselseitige Vorhersagbarkeit: Jedes Codesymbol läßt sich aus allen anderen vorhersagen. Vorhersagbarkeitsminimierung versucht gerade, dies durch Kreierung distribuierter nicht-expansiver Codes zu vermeiden.

Experiment 4: 'off-line', $dim(y) = 3$, $dim(x) = 4$, lokale Eingaberepräsentation (die Kapazität der Repräsentationsknoten überstieg also die in der Umgebung enthaltene Information um ein bit), 3 versteckte Knoten pro Prediktor, 4 allen Repräsentationsmodulen gemeinsame versteckte Knoten. 10 Testläufe mit 10000 Trainingsepochen für die Repräsentationsmodule wurden durchgeführt. In 7 Fällen genügte dies, um einen faktoriellen Binärcode zu finden: Nach dem Training emittierte einer der Repräsentationsknoten stets einen konstanten binären Wert c. Der vom entsprechenden Prediktor gelieferte bedingte Erwartungswert war natürlich gleich dem unbedingten Erwartungswert c, womit das Kriterium der statistischen Unabhängigkeit auf triviale Weise erfüllt war. In den verbleibenden 3 Fällen war der Code zumindest binär und invertibel.

Experiment 5: 'off-line', $dim(y) = 3$, $dim(x) = 4$, lokale Eingaberepräsentation (die Kapazität der Repräsentationsknoten überstieg also die in der Umgebung enthaltene Information um *zwei* bit), 3 versteckte Knoten pro Prediktor, 4 allen Repräsentationsmodulen gemeinsame versteckte Knoten. 10 Testläufe mit 10000 Trainingsepochen für die Repräsentationsmodule wurden durchgeführt. In 5 Fällen genügte dies, um einen faktoriellen Binärcode zu finden: Nach dem Training emittierten zwei der Repräsentationsknoten stets einen konstanten binären Wert. In den verbleibenden 3 Fällen verbrauchte der Code zwar mehr als die minimale Anzahl an Repräsentationsknoten, war jedoch zumindest binär und invertibel.

Experiment 6: on-line, $dim(y) = 4$, $dim(x) = 2$, verteilte Eingaberepräsentation (die Kapazität der Repräsentationsknoten überstieg die in der Umgebung enthaltene Information wieder um zwei bit), 2 versteckte Knoten pro Prediktor, 4 allen Repräsentationsmodulen gemeinsame versteckte Knoten. 10 Testläufe mit 250000 Musterpräsentationen wurden durchgeführt. Dies genügte stets, um einen faktoriellen Binärcode zu entdecken: Nach dem Training emittierten zwei der Repräsentationsknoten stets einen konstanten binären Wert. In 7 der 10 Fälle erwiesen sich weniger als 100000 Musterpräsentationen (dies entspricht 25000 Trainingsepochen) als erforderlich.

6.6.3 EXPERIMENTE MIT UNGLEICH VERTEILTEN EINGABEN

Das im vorliegenden Unterabschnitt betrachtete Eingabeensemble besteht aus 4 verschiedenen Mustern: x_a, x_b, x_c, und x_d. Die jeweiligen Wahrscheinlichkeiten ihres Auftretens sind

$$P(x^a) = \frac{1}{9}, P(x^b) = \frac{2}{9}, P(x^c) = \frac{2}{9}, P(x^d) = \frac{4}{9}.$$

Die dergestalt definierte Trainingsumgebung erlaubt binäre faktorielle Codes; einer davon ist der folgende

Code F: $y^a = (1,1)^T$, $y^b = (0,1)^T$, $y^c = (1,0)^T$, $y^d = (0,0)^T$.

Code F erfüllt das Unabhängigkeitskriterium, denn die unbedingten Erwartungswerte jedes Repräsentationsknotens gleichen den entsprechenden bedingten Erwartungswerten:

$$E(y_1) = \frac{2}{3} = E(y_1 \mid y_2 = 1) = E(y_1 \mid y_2 = 0),$$

$$E(y_2) = \frac{2}{3} = E(y_2 \mid y_1 = 1) = E(y_2 \mid y_1 = 0).$$

Code F's totales Performanzmaß V_C ist gleich $V_C^F = 2$.

Ein *nicht* faktorieller (wohl aber reversibler und damit informationserhaltender) Code ist der folgende

Code B: $y^a = (0,1)^T$, $y^b = (0,0)^T$, $y^c = (1,0)^T$, $y^d = (1,1)^T$.

Code B erfüllt das Unabhängigkeitskriterium *nicht*, denn nicht alle bedingten Erwartungswerte jedes Repräsentationsknotens gleichen seinem unbedingten Erwartungswert:

$$E(y_1 \mid y_2 = 0) = \frac{1}{2} \neq E(y_1) = \frac{2}{3} \neq E(y_1 \mid y_2 = 1) = \frac{4}{5},$$

$$E(y_2 \mid y_1 = 1) = \frac{2}{3} \neq E(y_2) = \frac{5}{9} \neq E(y_2 \mid y_1 = 0) = \frac{1}{3}.$$

Code B's totales Performanzmaß V_C ist gleich $V_C^B = \frac{19}{10}$, ein Wert, der nur um $\frac{1}{10}$ unterhalb von V_C^F liegt. Dies zeigt bereits, daß gewisse lokale Maxima der Zielfunktion der Repräsentationsknoten den gesuchten globalen Maxima sehr nahe kommen können. Diese Tatsache spiegelt sich in manchen der folgenden Experimente wieder.

Experiment 7: 'off-line', $dim(y) = 2$, $dim(x) = 2$, verteilte Eingaberepräsentation mit $x^a = (0,0)^T$, $x^b = (0,1)^T$, $x^c = (1,0)^T$, $x^d = (1,1)^T$, 1 versteckter Knoten pro Prediktor, 2 allen Repräsentationsmodulen gemeinsame versteckte Knoten. 10 Testläufe mit 2000 Trainingsepochen für

die Repräsentationsmodule wurden durchgeführt. Eine Epoche bestand dabei aus der Präsentation von 9 Mustern: x_a wurde einmal präsentiert, x_b wurde zweimal präsentiert, x_c wurde zweimal präsentiert, x_d wurde viermal präsentiert. In 7 Fällen fand das System einen zu einem globalen Maximum von V_C korrespondierenden faktoriellen Code.

Experiment 8 (Occams Rasiermesser): Wie Experiment 7, jedoch mit $dim(y) = 3$. In 9 von 10 Testläufen fand das System einen faktoriellen Code (mit einem unbenutzten Repräsentationsknoten, der stets denselben konstanten Wert lieferte). Beim verbleibenden Testlauf erwies sich der resultierende Code zumindest als informationserhaltend.

Experiment 9: Wie Experiment 7, jedoch lokale Eingaberepräsentation, 3 versteckte Knoten pro Prediktor, 4 allen Repräsentationsmodulen gemeinsame versteckte Knoten. 10 Testläufe mit 10000 Trainingsepochen für die Repräsentationsmodule wurden durchgeführt. In 5 Fällen fand das System einen zu einem globalen Maximum von V_C korrespondierenden faktoriellen Code. In 2 weiteren Fällen erwies sich der Code als invertibel, in den 3 verbleibenden Fällen nicht.

Wie schon des öfteren erwähnt, ist die wohl einfachste Strategie gegen lokale Maxima folgende: Wiederhole das Experiment unter verschiedenen Initialisierungsbedingungen, bis ein befriedigendes Ergebnis erreicht wird. Jedes neue Experiment entspricht einer neuen lokalen Suche im Suchraum.

Experiment 10 (Occams Rasiermesser): Wie Experiment 9, aber mit $dim(y) = 3$. In 2 aus 10 Testläufen fand das System einen faktoriellen Code (mit einem unbenutzten Repräsentationsknoten, der stets denselben konstanten Wert lieferte). In den verbleibenden 8 Testläufen wurden stets invertible Codes entdeckt. Dies reflektiert einen *'trade-off'* zwischen Redundanz und Reversibilität: Überflüssige Freiheitsgrade unter den Repräsentationsknoten vermögen zwar einerseits die Wahrscheinlichkeit der Auffindung informationserhaltender Codes zu erhöhen, können aber andererseits die Wahrscheinlichkeit der Entdeckung eines faktoriellen Codes drücken.

6.6.4 EXPERIMENTE MIT BUCHSTABENBILDERN MIT UNTERSCHIEDLICHEN AUFTRETENSWAHRSCHEINLICHKEITEN

Um die Skalierungseigenschaften des Verfahrens zu testen, wurden in Zusammenarbeit mit Stefanie Lindstädt (University of Colorado at Boulder) auch aufwendigere Experimente zur Kodierung von Buchstabenbildern (mit den der englischen Sprache entsprechenden Auftretenswahrscheinlichkeiten) durchgeführt.

82 verschiedene Zeichen (Kleinbuchstaben, Großbuchstaben, Ziffern und

Sonderzeichen) wurden jeweils durch ein aus 10×15 Pixeln bestehendes Bild repräsentiert (die Bilder entstammten dem *DEC courier* Datensatz). Jedes Pixel war entweder schwarz oder weiß. Damit ließen sich die Buchstabenbilder als 150-dimensionale Binärvektoren darstellen, deren n-te Komponente gleich 1 war, falls das entsprechende Pixel schwarz war, und 0 sonst. Die 82 Binärvektoren dienten als die Eingabevektoren x^p für die Repräsentationsmodule. Ihre Auftretenswahrscheinlichkeiten sind in Tabelle 6.1 aufgelistet (siehe auch [5]).

Um die Performanz der Methode zur Redundanzreduktion zu analysieren, wurden folgende Maße verwendet:

1. Maß der Informationstransmission: Um zu messen, wieviel Information über das Eingabeensemble nach dem Training durch das Netzwerk an die (binären) Repräsentationsknoten übermittelt wird, mißt man die wechselseitige Information zwischen Eingabe und Codierung

$$\mathcal{I}(x,y) = \mathcal{H}(x) + \mathcal{H}(y) - \mathcal{H}(x,y), \tag{6.11}$$

wobei

$$\mathcal{H}(z) = -\sum_p P(z^p) log P(z^p)$$

die Entropie von z bezeichnet. Da die von den Repräsentationsmodulen geleistete Abbildung deterministisch ist, vereinfacht sich (6.11) zu

$$\mathcal{I}(x,y) = \mathcal{H}(y). \tag{6.12}$$

$\mathcal{I}(x,y)$ kann die Entropie des Eingabeensembles, welche sich aus Tabelle 6.1 zu 4.34 bit errechnet, nicht übersteigen.

2. Abhängigkeitsmaß: Um zu messen, wie sehr die Codekomponenten nach dem Training voneinander abhängen, berechnet man

$$\mathcal{D} = \sum_{i,p} P(x^p) \left(y_i^p - E\left[y_i | \left\{ y_j^p, j \neq i \right\} \right] \right)^2. \tag{6.13}$$

3. Redundanzmaß: Schließlich definieren wir das Maß der Redundanz R einer Binärcodierung durch die Differenz zwischen der tatsächlichen Entropie des Codes und der maximal möglichen Entropie eines Binärcodes mit ebensovielen Komponenten:

$$\mathcal{R} = \frac{-\mathcal{H}(y) + \sum_i P(y^i = 1) log P(y^i = 1) + (1 - P(y^i = 1)) log(1 - P(y^i = 1))}{\mathcal{H}(y)}. \tag{6.14}$$

Betrachten wir unser Eingabeensemble (bestehend aus 82 150-dimensionalen binären Eingabevektoren) als seine eigene Codierung, so ist die Redundanz des Codes hoch – sie beträgt 1341 Prozent.

Wahrsch.	Zeichen	Wahrsch.	Zeichen	Wahrsch.	Zeichen
0.17595	'Blank'	0.00014	!	0.00070	"
0.00005	#	0.00002	%	0.00028	'
0.00150	(	0.00171	)	0.00012	+
0.00753	,	0.00122	-	0.00490	.
0.00047	/	0.00061	0	0.00117	1
0.00096	2	0.00014	3	0.00009	4
0.00007	5	0.00019	6	0.00002	7
0.00007	8	0.00007	9	0.00033	:
0.00098	;	0.00002	<	0.00016	=
0.00007	>	0.00009	?	0.00108	A
0.00117	B	0.00063	C	0.00028	D
0.00045	E	0.00042	F	0.00014	G
0.00026	H	0.00195	I	0.00002	J
0.00007	K	0.00026	L	0.00028	M
0.00253	N	0.00066	O	0.00129	P
0.00019	Q	0.00056	R	0.00117	S
0.00195	T	0.00012	U	0.00148	V
0.00019	W	0.00009	X	0.00002	[
0.00002	]	0.05791	a	0.01768	b
0.02610	c	0.02345	d	0.09843	e
0.01953	f	0.01231	g	0.03604	h
0.06272	i	0.00054	j	0.00251	k
0.03050	l	0.01796	m	0.05773	n
0.06214	o	0.01768	p	0.00176	q
0.04375	r	0.05620	s	0.07817	t
0.02610	u	0.00943	v	0.01023	w
0.00181	x	0.01032	y	0.00021	z
0.00052	—				

Table 6.1: *82 im Experiment verwendete Zeichen und ihre Auftretenswahrscheinlichkeiten in englischsprachigen Texten.*

Die folgenden Experimente zeigen die Performanz des Systems in Abhängigkeit von der Anzahl der Codekomponenten (der Anzahl der Repräsentationsknoten):

Experiment 11: 'on-line', 150 Eingabeknoten, ein zusätzlicher Eingabeknoten mit konstanter Aktivation 1.0 (der *'bias'*-Knoten), 8 Repräsentationsknoten, keine versteckten Knoten für die Repräsentationsmodule, ebensoviele versteckte Knoten pro Prediktor wie Prediktoreingabeknoten, Prediktorenlernrate 1.0, Repräsentationsmodullernrate 0.1, Verzicht auf Fehlerrückpropagierung durch Prediktoreneingabeknoten. 10 Testläufe mit je 10000 Musterdarbietungen (gemäß den Auftretenswahrscheinlichkeiten aus Tabelle 6.1) wurden durchgeführt. *Resultate:* Durchschnittliche Informationstransmission $\mathcal{I}(x,y) = 4.12$ bit (das theoretische Maximum liegt, wie bereits erwähnt, bei 4.34 bit), durchschnittliche Redundanz $\mathcal{R} = 0.82$, durchschnittliche Abhängigkeit $\mathcal{D} = 1.27$.

Experiment 12: Wie Experiment 11, aber mit 10 Repräsentationsknoten. *Resultate:* $\mathcal{I}(x,y) = 4.25$ bit, $\mathcal{R} = 1.1$, $\mathcal{D} = 1.8$.

Experiment 13: Wie Experiment 12, aber mit 12 Repräsentationsknoten. *Resultate:* $\mathcal{I}(x,y) = 4.26$ bit, $\mathcal{R} = 1.6$, $\mathcal{D} = 2.15$.

Experiment 14: Wie Experiment 13, aber mit 14 Repräsentationsknoten. *Resultate:* $\mathcal{I}(x,y) = 4.29$ bit, $\mathcal{R} = 2.05$, $\mathcal{D} = 2.4$.

Experiment 15: Wie Experiment 14, aber mit 16 Repräsentationsknoten. *Resultate:* $\mathcal{I}(x,y) = 4.28$ bit, $\mathcal{R} = 2.5$, $\mathcal{D} = 2.6$.

Aus den Experimenten 12 bis 15 ist ersichtlich, daß sich mit steigender Anzahl der Repräsentationsknoten die Informationsübertragung dem theoretischen Maximum annähert, während gleichzeitig die Redundanz innerhalb des Codes und auch die statistische Abhängigkeit der Codesymbole untereinander zunimmt. Dies verdeutlicht erneut den *'trade-off'* zwischen Informationstransmission und Redundanz: Optimale faktorielle Codierungen des Eingabeensembles wurden praktisch nie erreicht, hohe Informationsübertragung wurde durch den Verlust an Kompaktheit und statistischer Unabhängigkeit der Codekomponenten erkauft.

Dennoch wurde in allen Fällen eine bedeutende Redundanzreduktion erreicht (von 1341 Prozent Redundanz im Eingabeensemble auf deutlich unter 300 Prozent).

Eine der für die Reduktion der im *DEC courier*-Datensatz enthaltenen Redundanz brauchbarsten getesteten Vorgehensweisen war die folgende: Mit 16 Repräsentationsknoten (wie in Experiment 14) wurde das System solange mehrmals hintereinander von neuem zufällig initialisiert und daraufhin durch je 3000 Musterpräsentationen trainiert, bis das theoretische Optimum der Informationstransmission erreicht wurde (da in den Experimenten 12 - 14 Durchschnitte anhand von 10 Testläufen gebildet wurden, taucht die maximal mögliche wechselseitige Information zwischen Ein- und

Ausgabe dort nie auf). Dieses Vorgehen entspricht der einfachsten Strategie zur Behandlung lokaler Maxima und erforderte meist nicht mehr als 5 sukzessive Läufe.

Zusätzliche Addition von uniform verteiltem Rauschen im Intervall $[-1, +1]$ auf die Aktivationen der Eingabeknoten während der Trainingsphase erwies sich ebenfalls zur Vermeidung lokaler Maxima als zweckmäßig.

Welcher Art sind die 'Objekte', die durch Vorhersagbarkeitsminimierung aus dem Eingabeensemble extrahiert werden? Abbildung 6.3 veranschaulicht die Gewichte der Verbindungen zwischen den 10×15 Eingabeknoten und jedem der 16 Repräsentationsknoten nach dem Training. Die 3. Matrix in der 2. Reihe von oben steht beispielsweise für die Eingangsgewichte des 7. Repräsentationsknotens (das einzelne Feld unterhalb jeder Gewichtsmatrix steht für die Stärke der jeweiligen Verbindung vom *'bias'*-Knoten). Je positiver ein Gewicht, desto größer das entsprechende weiße Quadrat. Je negativer ein Gewicht, desto größer das entsprechende schwarze Quadrat.

Aus Abbildung 6.3 geht hervor, daß die zur Redundanzminderung aus dem Datensatz extrahierten 'Objekte' keine für menschliche Anschauung offensichtliche Interpretation besitzen, obwohl die Strukturen gewisse Regelmäßigkeiten aufweisen und gelegentlich bestimmte häufig auftretende Buchstabenteile eine Entsprechung in stark positiven oder stark negativen Eingangsgewichten finden (siehe beispielsweise die Matrix Nummer 13).

6.6.5 EXPERIMENTE ZUR KOMBINATION VON VORHERSAGBARKEITSMAXIMIERUNG MIT VORHERSAGBARKEITSMINIMIERUNG

Im folgenden kombinieren wir Vorhersagbarkeitsminimierung mit dem Ansatz zur Vorhersagbarkeits*maximierung* aus dem letzten Kapitel (Abschnitt 5.5).

STEREO-EXPERIMENT

Das in Abschnitt 5.5 sowie in [11] beschriebene Stereo-Experiment diente zum Vergleich von IMAX (Abschnitt 5.5.2) und Vorhersagbarkeitsmaximierung. Der 'diskriminierende' Zielfunktionsterm D_l war dabei durch Vorhersagbarkeits*minimierung* definiert.

Die beiden Eingabetransformatoren T^1 und T^2 besaßen jeweils 8 Eingabeknoten, 12 versteckte Knoten sowie einen einzigen Ausgabeknoten (da ja die zu extrahierende Eigenschaft (der *'shift'*) ein binäres Merkmal ist). Demgemäß reichte auch ein einziger Prediktor pro Transformer aus, um den Ausgabeknoten (aus einem Knoten mit konstanter Aktivation) vorherzusagen.

Experiment 16: 'on-line', Vorhersagbarkeitsmaximierung gemäß (5.15), separate Gewichtssätze für beide Transformatoren, keine versteckten Knoten in den Prediktoren, Lernraten der Prediktoren gleich 1.0, Lernraten der Transformatoren gleich 0.5, $\epsilon = 0.5$, $\lambda = 1.0$. 10 Testläufe wurden durchgeführt. In allen Fällen genügten wie bei (Becker und Hinton) 100000 Musterpräsentationen zur Extraktion des *'shifts'*. Dem entspricht 1 bit wechselseitiger Information zwischen den Ausgaben der Transformatoren.

Im Gegensatz zu Beckers und Hintons Methode waren dabei weder sukzessive 'bootstrap'-*Trainingszyklen noch Lernratenanpassung oder irgendwelche sonstigen heuristischen Kniffe notwendig.*

Experiment 2: Wie Experiment 1, allerdings teilten sich nun beide Transformatoren denselben Gewichtssatz. Dies führte zu einer signifikanten Reduktion der Anzahl der freien Parameter (siehe auch Abschnitt 5.5), was in deutlicher Beschleunigung des Lernvorgangs resultierte. Bei 10 Testläufen genügten zwischen 20000 und 50000 Musterpräsentationen zur Extraktion des *'shifts'*.

DISTRIBUIERTE REPRÄSENTATIONEN

In Abschnitt 5.5.3 wurde D_l im zweiten Experiment durch einen Autoassoziator definiert. D_l läßt sich jedoch statt dessen auch durch Predaktibilitätsminimierung festlegen.

Experiment 17: 'on-line', Vorhersagbarkeitsmaximierung gemäß (5.15), D_l definiert durch Vorhersagbarkeitsminimierung, ein gemeinsamer Gewichtsatz für beide Transformatoren, 2 versteckte Knoten pro Prediktor, 4 versteckte Knoten pro Transformator, Lernraten der Prediktoren gleich 1.0, Lernraten der Transformatoren gleich 0.5, $\epsilon = 0.5$, $\lambda = 1.0$. 10 Testläufe mit 10000 Musterpräsentationen wurden durchgeführt. Wie schon in Kapitel 5 fand das System stets eine distribuierte nahezu binäre Repräsentation der 4 möglichen vorhersagbaren Eigenschaftskombinationen.

6.7 VORHERSAGBARKEITSMINIMIERUNG UND ZEIT

Wir wollen nun noch kurz auf Eingabe*sequenzen* (im Gegensatz zu stationären Eingaben) eingehen. Dieser Abschnitt beschreibt eine völlig lokale Methode zum Auffinden eindeutiger, nicht-redundanter, reduzierter Sequenzbeschreibungen. Das nächste Kapitel wird verwandte Methoden allerdings wesentlich ausführlicher behandeln.

Der Anfangszustandsvektor $y^p(0)$ der Repräsentationsknoten sei für alle Sequenzen p der gleiche. Der Eingabevektor zur Zeit $t > 0$ der Sequenz p

sei die Konkatenation $x^p(t) \circ y^p(t-1)$ der Eingabe $x^p(t)$ und des vorangegangenen internen Zustands $y^p(t-1)$. Die Repräsentation des bis zum Zeitpunkt t beobachteten Sequenzpräfixes sei durch $y^p(t)$ selbst gegeben.

Wir minimieren und maximieren im wesentlichen dieselben Zielfunktionen wie im stationären Fall. Für den i-ten Repräsentationsknoten, *dem nun rekurrente Verbindungen zu sich selbst und zu den anderen Modulen entspringen* (siehe Abbildung 6.4), gibt es wieder einen adaptiven Prediktor P_i, *der seinerseits nicht rekurrent zu sein braucht.* P_i's Eingabe zur Zeit t ist die Konkatenation der Ausgaben $y_k^p(t)$ aller Repräsentationsknoten $k \neq i$. P_i's eindimensionale Ausgabe $P_i^p(t)$ wird gemäß der Zielfunktion

$$\frac{1}{2} \sum_p \sum_t (P_i^p(t) - y_i^p(t))^2$$

daraufhin trainiert, sich $E(y_i \mid \{y_k(t), k \neq i\})$ anzugleichen. Die Repräsentationsknoten versuchen ihrerseits, das Performanzmaß

$$\bar{E} = \sum_t T(t)$$

zu maximieren, wobei $T(t)$ analog zu den entsprechenden stationären Fällen definiert ist.

Die einzige Möglichkeit, die ein Repräsentationsknoten wahrnehmen kann, um sich selbst vor auf den übrigen Repräsentationsknoten beruhenden Voraussagen zu schützen, besteht darin, mittels rekurrenter Verbindungen Eigenschaften der Eingabesequenzen zu speichern, die von den von den übrigen Knoten gespeicherten Aspekten statistisch unabhängig sind.

Um angemessene Gewichtsänderungen zu bestimmen, wird lediglich Information über den Zustand zum lezten Zeitschritt benötigt. Dies hat einen (im Gegensatz zu den ersten drei Methoden aus Kapitel 2) *lokalen* Algorithmus zur Folge. Trotzdem erlaubt das Verfahren theoretisch, eindeutige Repräsentationen beliebig langer Sequenzen und all ihrer Untersequenzen zu finden – wie sich durch Induktion über die Länge der längsten Eingabesequenz sehen läßt:

1. y kann eindeutige Repräsentationen der Anfänge aller Sequenzen lernen.

2. Angenommen, alle Sequenzen und Untersequenzen der Länge $< k$ sind bereits eindeutig in y repräsentiert. Auch wenn zu jedem Zeitpunkt nur der letzte Zustand, nicht aber frühere Zustände berücksichtigt werden, kann y eindeutige Repräsentationen aller Sequenzen und Subsequenzen der Länge k lernen.

Obige Gedankenführung vernachlässigt allerdings mögliche *'cross-talk'*-Effekte. Die Experimente des nächsten Abschnittes zeigen jedoch die Anwendbarkeit der Methode.

6.7.1 EXPERIMENTE MIT SEQUENZEN

Die folgenden Experimente zur Kodierung von sequentiell (buchstabenweise) dargebotenen Wörtern mit gleichen Endungen sowie von in einem visuellen Feld wandernden Balken (durchgeführt in Zusammenarbeit mit Stefanie Lindstädt, CU) zeigen die prinzipielle Anwendbarkeit des in Abschnitt 6.7 vorgestellten Verfahrens auf einfache Quellencodierungsprobleme.

Der erste verwendete Datensatz bestand aus den 4 englischen Wörtern *'main', 'vain', 'rain',* und *'pain'*. Diese zeichnen sich durch gleiche Endungen aus und unterscheiden sich lediglich im jeweils ersten Buchstaben. Um nach der sequentiellen Beobachtung eines Wortes eine eindeutige Sequenzrepräsentation liefern zu können, muß das System also lernen, 4 Zeitschritte in die Vergangenheit zu blicken.

Da der Datensatz 16 Subsequenzen erlaubt (welche alle eindeutig repräsentiert werden müssen), ergab sich die Eingabeentropie zu 4 bit. Da die Eingabecodierung der Buchstaben dem 7 bit ASCII-Code entsprach, betrug die Redundanz 2.08.

Experiment 18: 'on-line', 7 Eingabeknoten, 8 rekurrente Repräsentationsknoten, keine versteckten Knoten für die Repräsentationsmodule, ebensoviele versteckte Knoten pro Prediktor wie Prediktoreingabeknoten, Verzicht auf Fehlerrückpropagierung durch Prediktoreneingabeknoten, Prediktorenlernrate 1.0, Repräsentationsmodullernrate 0.1. 5 Testläufe mit je 5000 zufällig gewählten Sequenzdarbietungen wurden durchgeführt. *Resultate:* Durchschnittliche Informationstransmission $\mathcal{I}(x, y) = 2.6$ bit (das theoretische Maximum liegt, wie bereits erwähnt, bei 4.00 bit), durchschnittliche statistische Abhängigkeit der Codesymbole $\mathcal{D} = 0.6$.

Experiment 19: Wie Experiment 18, aber mit 6 Repräsentationsknoten. *Resultate:* $\mathcal{I}(x, y) = 3.0$ bit, $\mathcal{D} = 1.25$.

Experiment 20: Wie Experiment 19, aber mit 8 Repräsentationsknoten. *Resultate:* $\mathcal{I}(x, y) = 3.8$ bit, $\mathcal{D} = 1.65$.

Experiment 21: Wie Experiment 20, aber mit 14 Repräsentationsknoten. *Resultate:* $\mathcal{I}(x, y) = 4.00$ bit (theoretisches Maximum), $\mathcal{D} = 3.00$.

Erneut wird der *'trade-off'* zwischen Informationstransmission und statistischer Abhängigkeit der Codesymbole deutlich. Bei 14 Repräsentationsknoten fand das System schließlich immer eine eindeutige Codierung aller Sequenzen und Subsequenzen, was einer deutlichen Redundanzminderung entspricht - 4×7 bit sind doppelt soviel wie 14 bit.

Der zweite verwendete Datensatz ergab sich durch Sequenzen von Eingabevektoren, welche dadurch erzeugt wurden, daß Balken vierer verschiedener Orientierungen nach einer der beiden zum Balken senkrechten Richtungen sequentiell quer über ein aus 8×8 Pixeln bestehendes Pixelfeld wanderten. Die Breite eines Balkens betrug ein Pixel, seine Enden ragten

während seiner Wanderung stets über das Pixelfeld hinaus. Die vier möglichen Balkenorientierungen waren: 0 Grad, 45 Grad, 90 Grad und 135 Grad. Zu jedem gegebenen Zeitpunkt war genau ein Balken sichtbar. Diese Versuchsanordnung zieht 64 mögliche beobachtbare Subsequenzen nach sich, welche alle eindeutig zu repräsentieren sind. Die Eingabeentropie beträgt demnach 6.0 bit. Man beachte, daß jeder Eingabevektor in zwei verschiedenen Sequenzen auftaucht – es kommt zur Disambiguierung also auf den zeitlichen Kontext an. Die Redundanz ist aufgrund der zahlreichen unterschiedlichen denkbaren, sich über 8 Zeitschritte erstreckenden Sequenzen sehr hoch.

Experiment 22: 'on-line', 64 Eingabeknoten, 6 rekurrente Repräsentationsknoten, keine versteckten Knoten für die Repräsentationsmodule, ebensoviele versteckte Knoten pro Prediktor wie Prediktoreingabeknoten, Prediktorenlernrate 1.0, Repräsentationsmodullernrate 0.1. 5 Testläufe mit je 10000 zufällig gewählten Sequenzdarbietungen wurden durchgeführt. *Resultate:* Durchschnittliche Informationstransmission $\mathcal{I}(x, y) = 5.4$ bit (das theoretische Maximum liegt bei 6.00 bit), durchschnittliche statistische Abhängigkeit der Codesymbole $\mathcal{D} = 0.5$.

Experiment 23: Wie Experiment 22, aber mit 14 Repräsentationsknoten. *Resultate:* $\mathcal{I}(x, y) = 6.00$ bit (theoretisches Maximum), $\mathcal{D} = 3.2$.

Bei 14 Repräsentationsknoten fand das System erneut fast immer eine eindeutige Codierung aller Sequenzen und Subsequenzen, was wiederum eine deutlichen Redundanzminderung zur Folge hat - 8×64 bit sind ein Vielfaches von 14 bit.

Trotz ihrer prinzipiellen Anwendbarkeit ist die in Abschnitt 6.7 vorgestellte Methode allerdings gelegentlich weniger brauchbar wie die Verfahren des nächsten Kapitels, welches tiefer in das unüberwachte Lernen eindeutiger Sequenzrepräsentationen einsteigen wird.

6.8 ABSCHLIESSENDE BEMERKUNGEN

Man darf von auf Vorhersagbarkeitsminimierung beruhenden Gradientenabstiegsmethoden nicht erwarten, bei gegebener Trainingsumgebung stets beim ersten Versuch einen faktoriellen Code zu finden. Vorhersagbarkeitsminimierung erlaubt jedoch, Redundanzen zu entdecken und zu eliminieren, die von älteren linearen Methoden nicht eliminierbar waren.

In vielen realistischen Fällen würde man sich mit *Approximationen* nichtredundanter Codes begnügen. Das Experiment mit den unterschiedlich wahscheinlichen Buchstabenbildern zeigt, daß Vorhersagbarkeitsminimierung zu bedeutender Redundanzminimierung führen kann. Es bleibt zu untersuchen, ob das Verfahren auch dazu taugt, weitgehend redundanzfreie Re-

präsentationen von Eingaben aus der 'realen' Welt zu entdecken. Weiterführende Arbeiten sollen die in diesem Kapitel vorgestellten Methoden auf das Problem unüberwachter Bildsegmentation anwenden.

Zwischen Vorhersagbarkeitsminimierung und den konventionelleren *kompetitiven* Lernschemata besteht eine Verwandtschaft: In einem gewissen Sinne treten Repräsentationsknoten miteinander in Wettbewerb, um bestimmte 'abstrakte' Transformationen der Umgebungseingaben repräsentieren zu dürfen. Der Wettbewerb beruht dabei nicht auf physikalischen Nachbarschaftskriterien (siehe z.B. [42]), sondern auf wechselseitiger Vorhersagbarkeit. Im Gegensatz zu den von sogenannten *'winner-take-all'*-Netzwerken erlaubten Repräsentationen gestatten durch Vorhersagbarkeitsminimierung geformte distribuierte Repräsentationen im allgemeinen *mehr* als einen 'Gewinner', solange alle 'Sieger' für statistisch voneinander unabhängige aus der Umgebung extrahierte abstrakte 'Objekte' stehen.

Man könnte darüber spekulieren, ob das Gehirn ein ähnliches Prinzip benützt, um 'Konzepte' aus den Umgebungseingaben herauszufiltern. Möglicherweise wäre es eine lohnende Aufgabe, in biologischen Organismen nach 'Vorhersage-Neuronen' und den Vorhersagen ausweichenden 'Repräsentationsneuronen' zu forschen[2].

Das vorliegende Kapitel nimmt einen allgemeinen Standpunkt ein, indem es sich auf den generellen Fall nicht-linearer Vorhersagbarkeitsminimierung konzentriert. Gelegentlich mag es jedoch von Vorteil sein, die theoretischen Möglichkeiten der Prediktoren und Repräsentationsmodule einzuschränken, indem man sie linear oder semilinear macht. So mag sich zum Beispiel ein hierarchisches System mit von der Berechnungskapazität her limitierten, sich in aufeinanderfolgenden Stufen der Hierarchie befindlichen Modulen als geeignet erweisen, um hierarchische Strukturen spezieller Umgebungseingaben zu reflektieren.

6.9 APPENDIX

6.9.1 EINE NICHT PREDIKTORBASIERTE ZIELFUNKTION ZUR AUFFINDUNG FAKTORIELLER CODES

Untenstehendes Performanzmaß beruht auf dem in Kapitel 5 ausgeführten Konzept minimaler Bitentropiesummen. Ist Code B binär, so ist seine

[2] Die *'winner-take-all'*-Netze erschienen vielen Forschern noch nie als wirklich plausible Kandidaten für die Modellierung interner Repräsentationen biologischer Gehirne.

Bitentropiesumme $e(B)$ durch

$$e(B) = -\sum_i E(y_i) log E(y_i) - \sum_i (1 - E(y_i)) log(1 - E(y_i))$$

definiert. Existiert ein faktorieller Code, so ist die Minimierung von $e(C)$ über alle möglichen Binärcodes C (wie in [5] ausgeführt) äquivalent zur Auffindung eines der faktoriellen Codes. Dies können wir ausnützen, um folgende zu maximierende Zielfunktion für unsere Repräsentationsknoten zu definieren:

$$\alpha \sum_i \sum_p P(x^p)(y_i^p - E(y_i))^2 - \beta \sum_p P(x^p)(z^p - x^p)^T (z^p - x^p) + \sum_i (E(y_i) - \frac{1}{2})^2.$$

Dabei sei z^p wie in Abschnitt 6.3.3 definiert, und α und β stellen wieder positive Konstanten dar. Der erste Term zwingt jeden Knoten dazu, in Antwort auf ein bestimmtes Eingabemuster entweder an oder aus zu sein. Der zweite Term erzwingt Reversibilität. Der dritte Term zwingt den Mittelwert der Aktivationen jedes Knotens, sich nahe bei entweder 0 oder 1 zu befinden. Damit wird minimale Bitentropie ermutigt.

Die Methode weist Verwandschaft zu einem älteren Ansatz zur Auffindung nichtredundanter Codes auf [64]. Sie muß sich jedoch ebenfalls der in Abschnitt 6.3.7 erwähnten Kritik (betreffs des Parameterwahlproblems) beugen.

6.9.2 ZUR VERMUTUNG 6.4.1

Die Maximierung von V_C aus Abschnitt 6.4 ist äquivalent zur Maximierung von

$$Q_C = \sum_i \sum_p P(x^p)(P_i^p - y_i^p)^2 = E[E(y_i \mid \{y_k, k \neq i\}) - y_i]^2, \qquad (6.15)$$

wobei p über alle *unterschiedlichen* Muster (statt über *alle* Muster) rangiert, und wobei wieder angenommen wird, daß $P_i^p = E(y_i \mid \{y_k^p, k \neq i\})$ gilt.

Definieren wir nun α_j^i als das j-te unterschiedliche Ereignis der Form $\{y_k, k \neq i\}$. Es ist

$$E[E(y_i \mid \{y_k, k \neq i\}) - y_i]^2 \leq E\left[E(y_i) - y_i\right]^2, \qquad (6.16)$$

wobei das Gleichheitszeichen nur dann gilt, wenn der folgende Ausdruck wahr ist:

$$\forall i, j: \quad E(y_i \mid \alpha_j^i) = E(y_i).$$

Im Falle eines quasi-binären Codes läßt sich Q_C wie folgt umformen:

$$Q_C = \sum_i \sum_p P(x^p)(P_i^p - y_i^p)^2 =$$

$$= \sum_{i:\forall p:\ y_i^p = E(y_i) = const.} \sum_j P(\alpha_j^i)(E(y_i \mid \alpha_j^i) - E(y_i))^2 \ +$$

$$+ \sum_{i\ bin\ddot{a}r} \sum_j P(\alpha_j^i) \left[\sum_{p:y_i^p=1} P(x^p)(E(y_i \mid \alpha_j^i) - 1)^2 + \sum_{p:y_i^p=0} P(x^p)(E(y_i \mid \alpha_j^i) - 0)^2 \right] =$$

$$= 0 + \sum_{i\ bin\ddot{a}r} \sum_j P(\alpha_j^i) \left[(E(y_i \mid \alpha_j^i) - 1)^2 \sum_{p:y_i^p=1} P(x^p) + (E(y_i \mid \alpha_j^i))^2 \sum_{p:y_i^p=0} P(x^p) \right] =$$

$$= \sum_{i\ bin\ddot{a}r} \sum_j P(\alpha_j^i) \left[(E(y_i \mid \alpha_j^i) - 1)^2 E(y_i \mid \alpha_j^i) + (E(y_i \mid \alpha_j^i))^2 (1 - E(y_i \mid \alpha_j^i)) \right] =$$

$$= \sum_{i\ bin\ddot{a}r,\ j} P(\alpha_j^i) E(y_i \mid \alpha_j^i)(1 - E(y_i \mid \alpha_j^i)). \qquad (6.17)$$

Ist der quasi-binäre Code faktoriell, so wird (6.13) zu[3]

$$\sum_i E(y_i)(1 - E(y_i)). \qquad (6.18)$$

Die Maximierung von Q_C ermutigt quasi-binäre Codes. Betrachten wir einen quasi-binären faktoriellen Code F. Es ist

$$Q_C^F = \sum_{i\ bin\ddot{a}r} E^F(y_i)(1 - E^F(y_i)),$$

wobei zusätzliche hochgestellte Indices die Zugehörigkeit zu einem bestimmten Code bezeichnen. Jeder Code B mit

$$\sum_{i\ bin\ddot{a}r} E^B(y_i)(1 - E^B(y_i)) \leq Q_C^F$$

kann aufgrund von (6.12) und (6.14) keinen größeren Gesamtprediktionsfehler als F verursachen.

[3]Definiert man eine sehr ähnliche Zielfunktion $\bar{Q}_C = \sum_i \sum_p P(x^p) \mid P_i^p - y_i^p \mid$ (basierend auf Absolutwerten statt MSE), so findet man interessanterweise $\bar{Q}_C = 2Q_C$ im Falle eines quasi-binären faktoriellen Codes.

Was aber, wenn

$$\sum_{i \ bin\ddot{a}r} E^B(y_i)(1 - E^B(y_i)) > Q_C^E \quad ?$$

Intuitiv scheint dies nahezulegen, daß die Codekapazität die im Eingabeensemble enthaltene Entropie übersteigt, was intra-repräsentationelle Redundanz nach sich zieht, was wiederum kleineres Q_C^B zur Folge hat. Es wurde zwar versucht, auch letzteren Fall unter Ausnützung von

$$E(y_i \mid \alpha_j^i) = \frac{P(\alpha_j^i \mid y_i = 1)}{P(\alpha_j^i)} E(y_i),$$

formal zu fassen, die Vermutung 6.4.1 bleibt allerdings für den allgemeinen Fall unbewiesen.

6.9.3 ÄQUIVALENZ VON V_C UND $V - H$

Peter Dayan und Richard Zemel wiesen in persönlicher Kommunikation darauf hin, daß das Verfahren aus Abschnitt 6.3.6 und Vorhersagbarkeitsminimerung gemäß Abschnitt 6.4 für $\alpha = 1, \gamma = 1$ Kraft folgender Argumentation im wesentlichen äquivalent sind:

(6.11) läßt sich umschreiben als

$$\sum_{i,p} P(x^p) \left(y_i^p - E\left[y_i | \left\{y_j^p, j \neq i\right\}\right]\right)^2 =$$

$$\sum_{i,p} P(x^p)(y_i^p - \bar{y_i})^2 + \sum_{i,p} P(x^p) \left(E\left[y_i | \left\{y_j^p, j \neq i\right\}\right] - \bar{y_i}\right)^2 -$$

$$-2 \sum_{i,p} P(x^p)(y_i^p - \bar{y_i}) \left(E\left[y_i | \left\{y_j^p, j \neq i\right\}\right] - \bar{y_i}\right). \qquad (6.19)$$

Da

$$E\left[E\left[y_i | \left\{y_j, j \neq i\right\}\right]\right] = E\left[y_i\right] = \bar{y_i},$$

ist die dritte Zeile von (6.15) gleich

$$-2COV\left[y_i, E\left[y_i | \left\{y_j, j \neq i\right\}\right]\right].$$

Wie nun von Alex Pouget bemerkt wurde[4], gilt

$$COV\left[y_i, E\left[y_i | \left\{y_j, j \neq i\right\}\right]\right] = VAR\left[E\left[y_i | \left\{y_j, j \neq i\right\}\right]\right] =$$

[4]Alex Pouget wies darauf hin, daß für Zufallsvariablen Y und Z folgendes gilt:

$$COV(Y, E(Y|Z)) = \sum_{y,z} (y - \bar{y})(E(Y|z) - \bar{y}) P(y, z)$$

$$= \sum_{i,p} P(x^p) \left(E\left[y_i | \left\{ y_j^p, j \neq i \right\} \right] - \bar{y_i} \right)^2 \tag{6.20}$$

und damit auch

$$\sum_{i,p} P(x^p) \left(y_i^p - E\left[y_i | \left\{ y_j^p, j \neq i \right\} \right] \right)^2 = \sum_{i,p} P(x^p)(y_i^p - \bar{y_i})^2 - \sum_{i,p} P(x^p) \left(E\left[y_i | \left\{ y_j^p, j \neq i \right\} \right] - \bar{y_i} \right)^2,$$

was zu zeigen war.

$$= \sum_z [(E(Y|z) - \bar{y}) P(z) \sum_y (y - \bar{y}) P(y|z)]$$

$$= \sum_z (E(Y|z) - \bar{y}) P(z) * (E(Y|z) - \bar{y}) = VAR(E(Y|Z)),$$

wobei y über alle möglichen Werte von Y, und z über alle möglichen Werte von Z rangiert.

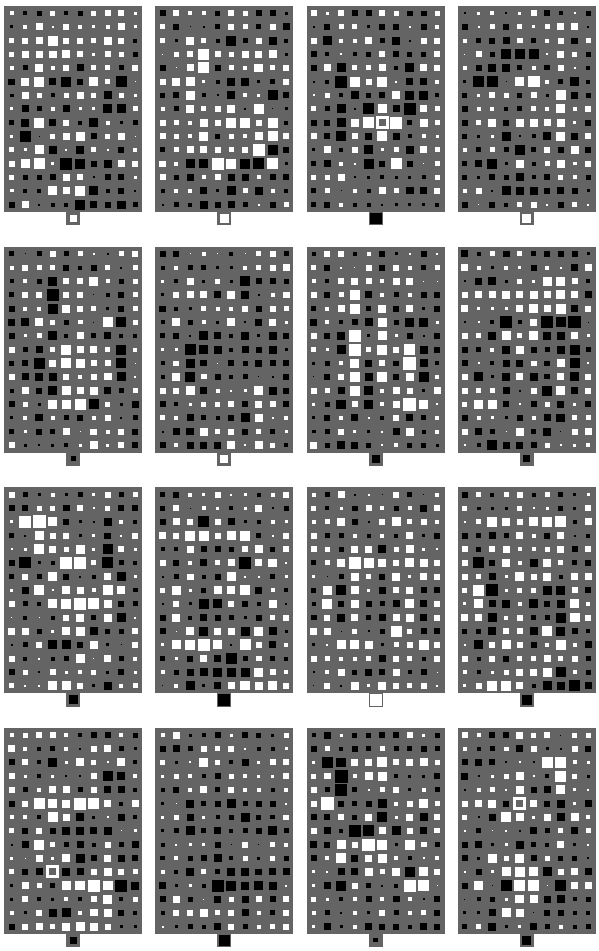

Abbildung 6.3: Veranschaulichung der Verbindungsgewichte zwischen den 10×15 Eingabeknoten und jedem der 16 Repräsentationsknoten nach dem Training mit dem Buchstabenbilderdatensatz. Je exzitatorischer ein Gewicht, desto größer das entsprechende weiße Quadrat. Je inhibierender ein Gewicht, desto größer das entsprechende schwarze Quadrat.

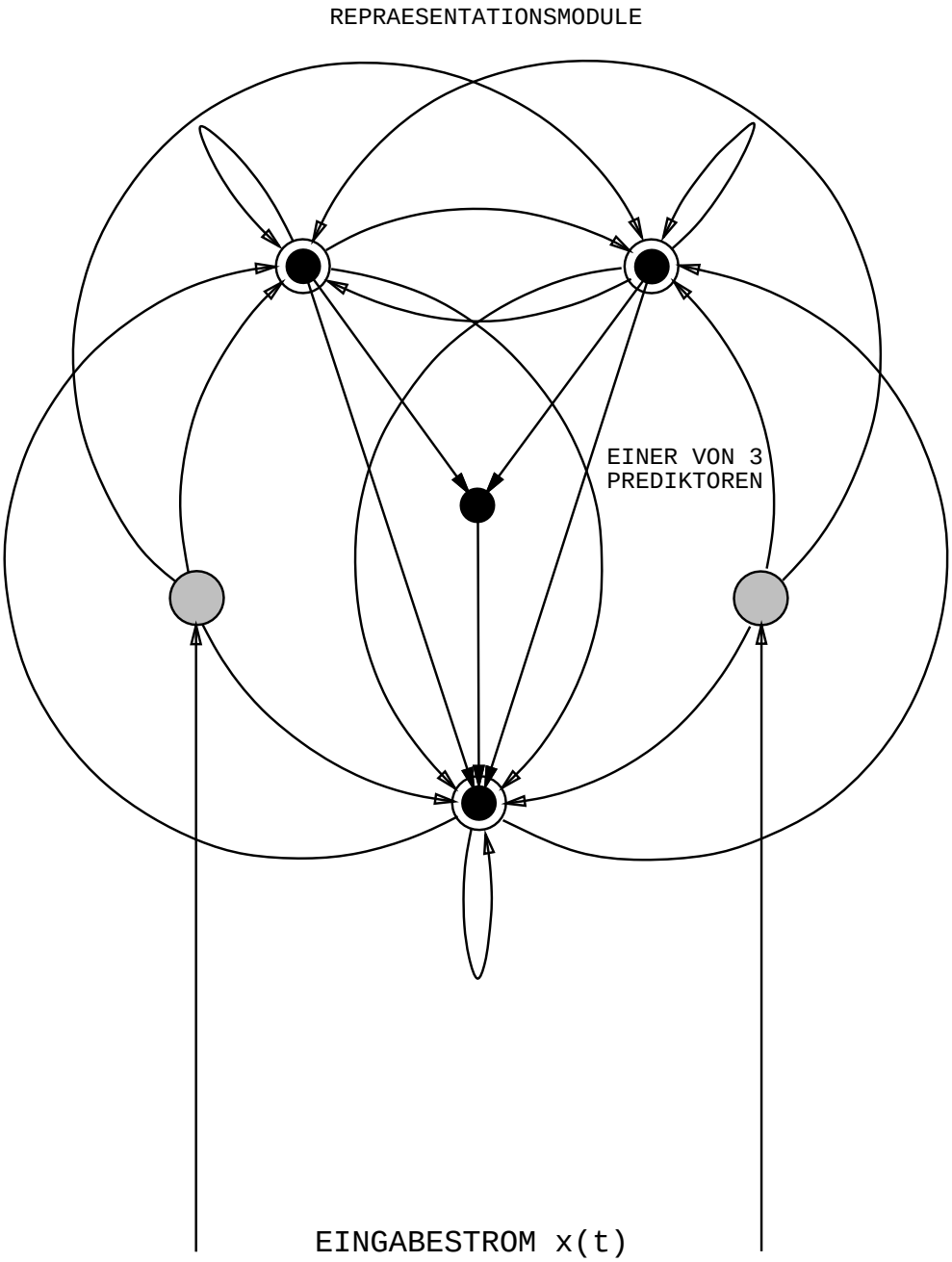

Abbildung 6.4: Ein rekurrentes Netz mit zwei Eingabeknoten (grau) und 3 vollständig vernetzten internen Knoten (weisse Kreise mit dem Durchmesser der grauen Kreise). Einer von drei Prediktoren mit 4 Knoten (kleine schwarze Kreise, ein versteckter Knoten) versucht, einen internen Knoten des rekurrenten Netzes aus den beiden anderen vorherzusagen. Das rekurrente Netz versucht seinerseits, derartige Vorhersagbarkeit zu minimieren.

Kapitel 7

UNÜBERWACHTE GESCHICHTS-KOMPRESSION

Alle mir bekannten in der Literatur bisher aufgetauchten unüberwachten Performanzmaße wurden im wesentlichen für statische Umgebungen entworfen. Auch die beiden vorangegangenen Kapitel befaßten sich schwerpunktmäßig mit unüberwachtem Lernen bei stationären Eingaben. Die Vorteile unüberwachter Performanzmaße kommen allerdings erst in dynamisch veränderlichen, nicht-stationären Umgebungen voll zum Ausdruck.

Das vorliegende Kapitel (der sechste bedeutende originäre Beitrag dieser Arbeit) zeigt an den Beispielen Sequenzrepräsentation und Sequenzklassifikation, daß die Algorithmen aus Kapitel 2 in gewissen typischen dynamischen Umgebungen durch Einführung zusätzlicher unüberwachter Performanzmaße und Architekturen zur Entdeckung *kausaler* Regelmäßigkeiten im Eingabestrom enorm beschleunigt werden können.

Zunächst wird zur Motivation an die praktischen Probleme der bisher besprochenen Algorithmen für den Fall erinnert, daß alle Trainingssequenzen lange zeitliche Lücken zwischen relevanten korrelierten Ereignissen aufweisen (siehe auch Kapitel 2).

Daraufhin wird ein einfaches und dennoch neuartiges Prinzip zum unüber-

wachten Finden eindeutiger reduzierter Darstellungen ausgedehnter Sequenzen angegeben. Dieses allgemeine *Prinzip der Geschichtskompression* benötigt zur Erklärung weder die Kettenregel noch eine spezifische Architektur. Es beruht auf der Einsicht, daß nur *unvorhersagbare* Eingaben nicht-redundante Information enthalten – erwartete Eingaben dürfen im wesentlichen ignoriert werden.

Aufbauend auf dem Prinzip der Geschichtskompression wird eine hierarchische Netzwerkarchitektur und eine zugehörige Serie von Performanzmaßen konstruiert. Ziel ist dabei das informationsverlustfreie unüberwachte Finden kompakter Repräsentationen eventuell im Eingabestrom vorhandener 'kausaler Regularitäten'. Diese kompakten Repräsentationen können Aufgaben wie Sequenzklassifikation gewaltig erleichtern, wie auch experimentell gezeigt wird. In der Tat ist es mit der neuartigen Architektur möglich, Sequenzen korrekt einzuordnen, bei denen die konventionelleren 'neuronalen' Algorithmen trotz ihrer theoretischen Allgemeinheit in der Praxis vollständig scheitern.

Die Kettenregel kommt dabei nur lokal auf jeder Ebene der Hierarchie ins Spiel. In der Tat ist das System ein Beispiel dafür, daß Lernalgorithmen wie die bisher behandelten, die ausschließlich durch sture rekursive Anwendung der Kettenregel in einer differenzierbaren Architektur hergeleitet wurden, nicht unbedingt das allein selig machende Hilfsmittel zum Entwurf neuronaler Lernalgorithmen sind. Um entsprechender Kritik gleich im voraus vorzubeugen, sei hier bemerkt, daß die Grundlage vorliegender Schrift durch diese Aussage nicht in Frage gestellt wird. Im Gegenteil: Neue Einsichten können Anlaß zu neuen Architekturen geben, in deren Untermodulen die Kettenregel den ihr angemessenen Platz finden kann. Das Entwerfen von solchen neuartigen, gewisse hilfreiche Prinzipien verkörpernden Architekturen und Zielfunktionen ist aus der von dieser Arbeit vertretenen Perspektive der eigentlich kreative, (zumindest im Moment) nicht automatisierbare Prozeß.

Ein weiterer Beitrag dieses Kapitels besteht in einer auf informationstheoretischen Erwägungen basierenden Modifikation der hierarchischen Prediktorenarchitektur.

Der Rest des Kapitels beschäftigt sich schließlich mit Architekturen (samt zugehörigen Zielfunktionen), die das Multiebenensystem mit der Zeit in ein einzelnes rekurrentes Netz kollabieren. Diese Kollabiermethoden stellen die einzigen 'schmutzigen' Verfahren dieser Arbeit dar – schmutzig insofern, als sie Gradientenabstieg in einer sich langsam ändernden Funktion betreiben

und damit mathematischer Exaktheit entbehren. Ich brachte es nicht übers Herz, diese Methoden zu ignorieren, vor allem deswegen, weil sie eine schöne Analogie zu antropomorphen Konzepten wie 'Bewußtsein' und 'Unterbewußtsein' bieten. Experimente illustrieren die Arbeitsweise der Verfahren und erweisen in gewissen Fällen eine mindestens 1000-fache Überlegenheit auch der 'schmutzigen' neuartigen Methoden über 'konventionellere' Algorithmen.

Ein Nebenprodukt der experimentellen Untersuchungen ist ein neuartiges, hybrides, gegenwärtig noch in der Erprobung befindliches Textkompressionsverfahren, welches bei gewissen Zeitungstexten bereits zu besserer Datenverdichtung führte als der weitverbreitete, in einem gewissen informationstheoretischen Sinne optimale Lempel-Ziv Textkompressionsalgorithmus, der u.a. auch die Grundlage der *compress*-Funktion des Betriebssytems *UNIX* bildet.

7.1 NACHTEILE DER REINEN GRADIENTENABSTIEGSVERFAHREN

In Kapitel 2, Abschnitt 6 haben wir u. a. durch die Experimente mit der *Standardaufgabe* gesehen, daß herkömmliche gradientenbasierte rekurrente Netze in der Praxis Schwierigkeiten bekommen, wenn alle Trainingssequenzen so gestaltet sind, daß sie lange zeitliche Verzögerungen zwischen Eingabeereignissen und korrelierten gewünschten Ausgabeereignissen beinhalten. Es stellte sich heraus, daß es rekurrenten Netzen bei regelmäßigen Verzögerungen von 20 Zeitschritten schlechterdings unmöglich ist, die Standardaufgabe bei vertretbarem Zeitaufwand zu lösen. Zwei Probleme mit existierenden gradientenbasierten sequenzverabeitenden Netzen wurden identifiziert:

1. Je mehr Zeit zwischen dem Auftreten eines Ereignisses und der Generierung eines durch dieses Ereignis bedingten Fehlersignals verstreicht, desto stärker wird das Fehlersignal i.a. auf seiner 'Reise in die Vergangenheit' gestreut, und desto unsignifikanter werden die durch den Fehler hervorgerufenen Gewichtsänderungen.

2. Aktivationen jedes Eingabeknotens werden zu jedem Zeitpunkt in unspezifischer Weise gleich stark berücksichtigt, um das Problem des *'temporal credit assignment's* [55] zu lösen. Keiner der bisher behandelten Algorith-

men versucht in irgendeiner Weise, die von einem Eingabesignal übermittelte *Information* zu messen. Keiner der bekannten (oder bisher in dieser Arbeit behandelten) Algorithmen lernt, unsignifikante (redundante) Eingaben von potentiell signifikanten Eingaben zu unterscheiden und sich selektiv auf informationstragende Ereignisse zu konzentrieren, um so Zeit und Ressourcen zu sparen.

[60], [61] and [80] haben ansatzweise das erste Problem angegriffen, nicht jedoch das zweite. [62] und [77] haben ansatzweise in höchst unterschiedlichen Kontexten auch das zweite Problem behandelt, den von ihnen verwendeten *ad-hoc* Methoden fehlt jedoch die solide theoretische Grundlage.

Es erheben sich folgende Fragen: Kann ein sequenzverarbeitendes System in theoretisch zu rechtfertigender Weise in unüberwachter Manier selbst lernen, seine Aufmerksamkeit auf relativ wenige signifikante Ereignisse im Eingabestrom zu lenken und 'unwichtige' Ereignisse zu ignorieren? Läßt sich dadurch seine Performanz bei Sequenzklassifikationsaufgaben entscheidend verbessern?

In (realistischen) Umgebungen, die einen Eingabestrom mit gewissen kausalen Abhängigkeiten produzieren, sind obige Fragen zu bejahen. Bei der Trainingsgrammatik T_1 aus Abschnitt 2.6 fällt zum Beispiel auf, daß die Sequenz $b_1 \ldots b_{100}$ eine Kausalkette darstellt: Das zweite Element der Kette kann aus dem ersten vorhergesagt werden, das dritte aus dem zweiten, und so fort. In gewisser noch zu präzisierender Weise trägt nur der Anfang einer Kausalkette nicht-redundante Information. Dies gibt Anlaß zu reduzierten Sequenzbeschreibungen, die, wie wir sehen werden, die Lösung mancher Klassifikationsaufgabe überhaupt erst ermöglichen.

Der Schwerpunkt dieses Kapitels liegt auf der unüberwachten Filterung redundanter Information aus nicht-deterministischen Eingabeströmen. Das allen Architekturen dieses Kapitels zugrundeliegende zentrale (und doch einfache) Prinzip ist

7.2 DAS PRINZIP DER GESCHICHTSKOMPRESSION

Man betrachte einen *deterministischen* Prediktor (nicht notwendigerweise ein KNN), dessen Zustand zur Zeit t der Sequenz p durch einen reellen Ein-

gabevektor $x^p(t)$ aus der Umgebung, einen internen Zustandsvektor $h^p(t)$, und einen Ausgabevektor $z^p(t)$ gegeben ist.

Die Eingaben aus der Umgebung selbst dürfen *nichtdeterministisch* sein. Zum Zeitpunkt 0 beginnt der Prediktor mit $x^p(0)$ und einem internen Initialisierungszustand $h^p(0)$. Zum Zeitpunkt $t \geq 0$ berechnet er

$$z^p(t) = f(x^p(t), h^p(t)). \tag{7.1}$$

Weiterhin berechnet der Prediktor zur Zeit $t > 0$

$$h^p(t) = g(x^p(t-1), h^p(t-1)). \tag{7.2}$$

Es ist nun entscheidend, einzusehen, daß die *gesamte* Information über die Eingabe zu jedem beliebigen Zeitpunkt t_x in

$$t_x, f, g, x^p(0), h^p(0), \{(t_s, x^p(t_s)) \;\; mit \;\; 0 < t_s \leq t_x, z^p(t_s - 1) \neq x^p(t_s)\} \tag{7.3}$$

enthalten ist [102]. Denn falls zum Zeitpunkt t $\quad z^p(t) = x^p(t+1)$ gilt, kann der Prediktor die neue Eingabe aus den alten Eingaben vorhersagen. Die neue Eingabe ist aus den alten mittels f und g *deduzierbar* und trägt damit lediglich redundante Information. Die gesamte Eingabesequenz kann aus der *reduzierten Beschreibung* (7.4) rekonstruiert werden [113].

Dabei spielt es keine Rolle, ob die Eingabesequenz durch einen deterministischen oder durch einen nicht-deterministischen Prozeß generiert wurde. Es geht nur um die Kompression der Beschreibung einer bereits beobachteten Sequenz – auf welche Weise die Sequenz entstanden ist, ist egal. Beispiel: Ein trivialer Prediktor, der bei einer Sequenz von Münzwürfen stets 'Wappen' prophezeiht, wird in etwa der Hälfte der Fälle recht behalten – damit läßt sich die Zahl der zur Beschreibung der beobachteten Sequenz benötigten Eingaben um etwa die Hälfte reduzieren[1].

Die Information über eine beobachtete Eingabesequenz läßt sich i.a. jedoch noch weiter komprimieren: Es genügt vollständig, wenn wir uns in (7.4) ausschließlich diejenigen *Komponenten* der Vektoren $x^p(t_s)$ merken, die nicht korrekt vorhergesagt wurden. Ich nenne dies das *Prinzip der Geschichtskompression.*

[1] Aus der algorithmische Informationstheorie wissen wir natürlich, daß sich wahrhaft zufällige Sequenzen prinzipiell nicht signifikant komprimieren lassen [17][43]. Bei der durch den trivialen Prediktor prozessierten Münzwurfserie steckt die vermeintlich 'hinwegkomprimierte' Information in Wirklichkeit in den zur Rekonstruktion der Serie erforderlichen, mit den unvorhergesagten Münzwurfresultaten assoziierten Zeitrepräsentanten. Wahre Kompressionserfolge lassen sich nur mit nicht zufälligen, 'Regularitäten' enthaltenden Sequenzen erzielen.

Typischerweise wollen wir beobachtete Sequenzen gar nicht vollständig rekonstruieren, sondern nur korrekt *klassifizieren*. Aus dem Prinzip der Geschichtskompression folgt, daß wir zwei oder mehr Sequenzen schon dann voneinander unterscheiden können, wenn wir nichts über die Sequenzen wissen außer den *falsch vorhergesagten Eingaben samt den Zeitschritten, zu denen sie auftraten*. Durch das Ignorieren erwarteter Eingaben geht keine Information verloren. *Zur Sequenzklassifikation müssen wir nicht einmal f und g kennen* – es genügt, zu wissen, daß sowohl f als auch g deterministisch sind.

Vom theoretischen Standpunkt aus ist es wirklich notwendig, den Zeitschritt zu kennen, zu dem eine unerwartete Eingabe auftrat. Ohne solches Wissen eröffnet sich Raum für Ambiguitäten: Zwei verschieden Eingabesequenzen können zur selben kürzeren Sequenz falsch prophezeihter Eingaben führen. Bei vielen praktischen Aufgaben ist es jedoch nicht unbedingt erforderlich, die 'kritischen' Zeitschritte zu kennen (siehe Abschnitt 7.4.1).

7.3 EINE SELBSTORGANISIERENDE PREDIKTORENHIERARCHIE

Aufbauend auf dem Prinzip der Geschichtskompression konstruieren wir eine sich selbst organisierende hierarchische Prediktorenarchitektur. Die Eingabe für jede Ebene der Hierarchie stammt aus der nächst niedrigeren Ebene. Das System entdeckt in unüberwachter Manier kausale Abhängigkeiten im Eingabestrom und lernt, signifikante informationstragende Ereignisse von anderen zu unterscheiden. Dabei lernt es, sowohl lokalere als auch globalere zeitliche Regularitäten auf verschiedenen Ebenen der Architektur widerzuspiegeln[2].

Die Aufgabe des Systems kann als Prediktionsaufgabe formuliert werden. Zu jedem Zeitpunkt besteht das Ziel darin, die neue Eingabe aus bereits beobachteten Eingaben vorherzusagen. Falls ein zusätzlicher Lehrer zu bestimmten Zeitpunkten bestimmte Ausgaben fordert, werden diese gewünschten Ausgaben als Teil der vorherzusagenden Eingabe betrachtet. Die Methode vereinfacht das in [101] vorgestellte Verfahren.

[2] [76] beschrieb eine alternative ad-hoc Methode zum Bau einer entfernt verwandten Hierarchie im Kontext des R-Lernens.

7.3.1 PRINZIP

$P_i, i = 1, 2, \ldots$ bezeichnet das *Prediktornetzwerk* der i-ten Ebene. Zur Implementierung von P_i eignen sich im Prinzip beliebige rekurrente Netze für dynamische Umgebungen (e.g [40], [23], [14], [74], [85], [144], [149], [59], [79], [3], [150], [148], [152], [109], [112], [29], [68], [72], [28], [147], [90], [97][3]). Zu jedem Zeitpunkt einer Sequenz p besteht die Eingabe für die unterste Ebene aus $x^p(t)$. Wir nehmen an, daß die Umgebung folgender (in allen vernünftigen Fällen erfüllten) Bedingung Genüge leistet:

Bedingung 7.3: Zu jeder reellwertigen Prediktion eines Eingabevektors läßt sich unter allen möglichen Eingabevektoren einer angeben, dessen euklidischer Abstand zur Prediktion minimal ist.

Wann immer ein P_i daran scheitert, seine eigene nächste Eingabe vorherzusagen (u.a. auch nach einem 'nullten' Initialisierungszeitschritt zu Beginn jeder Sequenz), ergibt sich P_{i+1}'s Eingabe durch die Konkatenation der tatsächlich beobachteten Eingabe und einer eindeutigen Repräsentation des entsprechenden Zeitschritts[4]. Die Aktivationen von P_{i+1}'s versteckten Knoten und Ausgabeknoten werden nur zu den 'kritischen' Zeitpunkten der nächstniedrigeren Ebene aktualisiert. Diese Prozedur stellt sicher, daß P_{i+1} ohne Informationsverlust mit einer eindeutigen reduzierten Beschreibung der Sequenz p gefüttert wird[5]. Das Prinzip der Geschichtskompression liefert hierfür die theoretische Grundlage. Siehe Abbildung 7.1.

Im allgemeinen wird P_{i+1} weniger Eingaben pro Zeiteinheit erhalten als P_i. Nach Sektion 7.1 sollte P_{i+1} damit i.a. weniger Schwierigkeiten als P_i haben, zu lernen, P_i's 'kritische Eingaben' vorherzusagen. Dies darf man aber *nur dann* erwarten, wenn der Eingabestrom globale zeitliche Regularitäten enthält, die P_i noch nicht entdeckt hat. In Umgebungen *ohne* hierarchische zeitlichen Strukturen bietet das Multiebenensystem keine Vorteile. Typische Eingabeströme scheinen allerdings hierarchisch aufgebaut zu sein –

[3] P_i könnte aber auch ein beliebiger anderer adaptiver Mechanismus zur Vorhersage von Sequenzeingaben sein. So könnte man z.B. ein azyklisches BP-Netz und den 'Zeitfensteransatz' verwenden. Mit dieser in ihren Möglichkeiten beschränkten Methode bleibt die Anzahl der vergangenen Eingaben, die für die Vorhersage der neuen Eingabe berücksichtigt werden können, konstant.

[4] Eine eindeutige Repräsentation der seit dem letzten unerwarteten Ereignis verstrichenen Zeitspanne tut es genauso.

[5] Im Gegensatz hierzu sind die von Mozer [61] zitierten 'reduzierten Beschreibungen' nicht eindeutig.

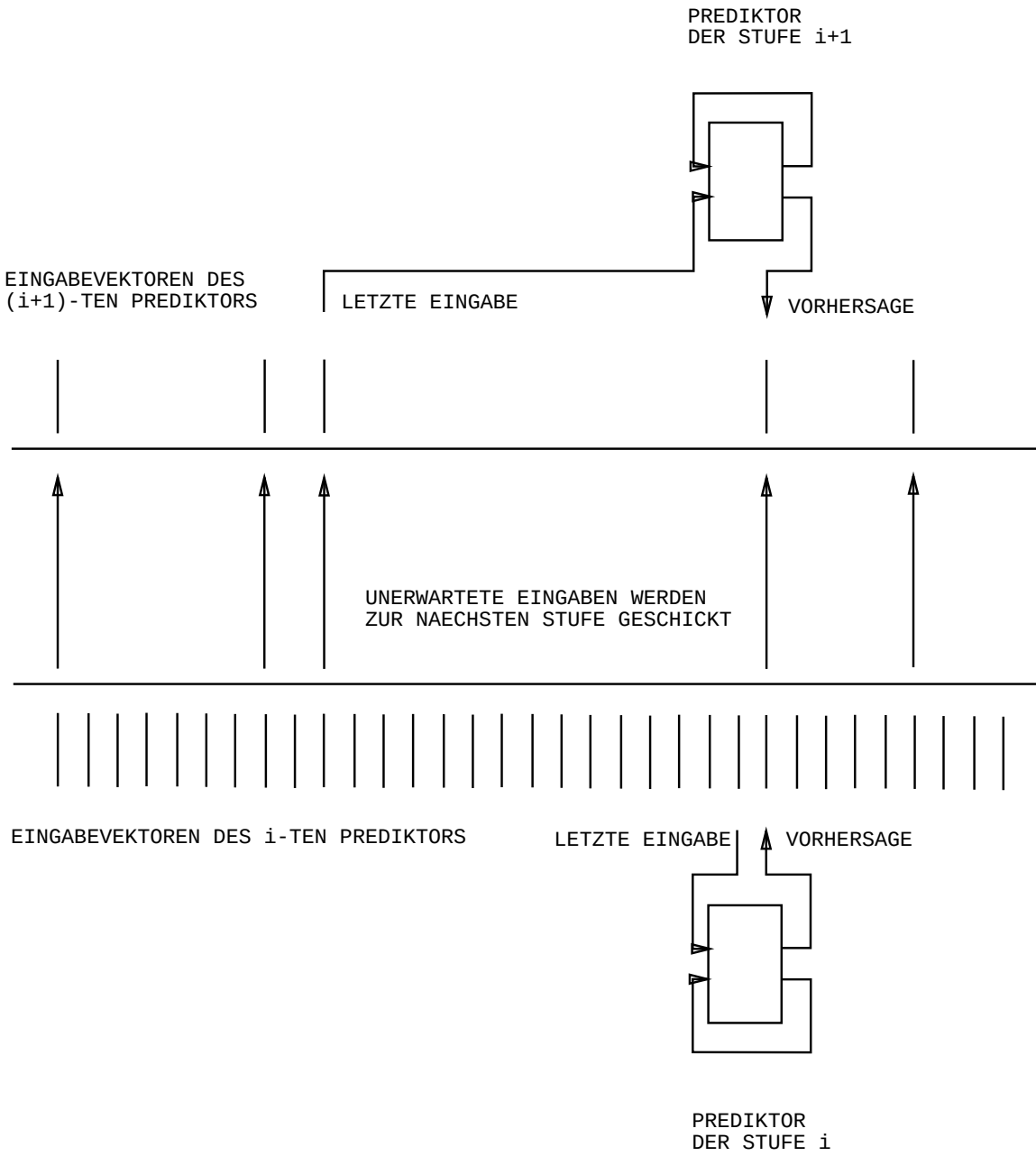

Abbildung 7.1: Ausschnitt aus der Prediktorenhierarchie. Der Prediktor der i-ten Stufe versucht, seine eigene nächste Eingabe aus vorangegangenen Eingaben vorherzusagen. Gelingt ihm das irgendwann einmal nicht, so wurde eine informationstragende Eingabe gefunden, welche samt einer eindeutigen Repräsentation des Zeitschrittes, zu welchem sie stattfand, an die nächste Stufe weitergereicht wird.

man denke nur an Sprachsignale. (Siehe [77] für einen verwandten ad-hoc Ansatz zur Lösung von Aufgaben aus dem Bereich R-Lernen.)

7.3.2 DETAILS: EINE SERIE VON ZIELFUNKTIONEN

Wie bereits erwähnt, wird P_{i+1} i. a. auf einer 'langsameren' Zeitskala als P_i arbeiten; die Zeitskala der Ebene $i+1$ wird durch die von P_i nicht korrekt vorhergesagten Eingaben definiert. Der t-te Zeitschritt der zur i-ten Ebene gehörigen Zeitskala sei mit t_i oder auch mit $(t)_i$ bezeichnet. P_i aktualisiert die Aktivationen seiner Knoten ausschließlich zu den Zeitpunkten $t_i, t = 0, 1, \ldots$. Bei gegebener Sequenz p stehe $x^{p,i}(t_i)$ für P_i's Eingabevektor zum Zeitpunkt t_i. Der Ausgabevektor trage den Namen $z^{p,i}(t_i)$. Wir nehmen an, daß Bedingung 7.3 erfüllt ist. Als aktuelle Vorhersage $\bar{z}^{p,i}(t_i)$ wird derjenige der möglichen Eingabevektoren angesehen, der den geringsten euklidischen Abstand zu $z^{p,i}(t_i)$ aufweist (siehe Bedingung 7.3). Gibt es mehrere derartige Eingabevektoren, so wird derjenige mit der kleinsten Nummer (bei beliebiger Ordnung auf der Menge der möglichen Eingabevektoren) herausgepickt. So erhalten wir eine eindeutige deterministische Vorhersage. Wir gehen inkrementell vor: Zunächst wird P_1 trainiert, nach P_1's Trainingsphase werden alle Gewichte in P_1 'eingefroren' und P_2's Trainingsphase beginnt, etc.

P_i's Zielfunktion ist gleich

$$E_i = \frac{1}{2} \sum_p \sum_{t_i} \sum_k (z_k^{p,i}(t_i) - (x_k^{p,i}((t+1)_i))^2. \tag{7.4}$$

Für die unterste Ebene gilt dabei

$$\forall t : t_1 = t, x^{p,1}(t_1) = x^p(t),$$

wobei $x^p(t)$ wie stets der t-te Eingabevektor der Sequenz p ist. Jeder Sequenz p wird dabei eine für alle Sequenzen gleichbleibende Initialisierungseingabe $x^p(0)$ vorangestellt.

Ist $i > 1$ (höhere Level), so gilt

$$x^{p,i}(t_i) = x^{p,i-1}(t_{i-1}) \circ rep(t_{i-1}),$$

wobei t_i auf der i-ten Zeitskala den Zeitpunkt des t-ten *unerwarteten* Ereignisses der Form

$$\bar{z}^{p,i-1}((t-1)_{i-1}) \neq x^{p,i-1}(t_{i-1})$$

bezeichnet und $rep(t_i)$ eine eindeutige vektorwertige Repräsentation von t_i (oder der seit $(t-1)_i$ verstrichenen Zeitspanne) darstellt.

Falls die Menge der potentiellen Eingabevektoren überabzählbar unendlich viele Elemente enthält (z. B. alle möglichen reellen Vektoren mit der Dimension des Eingaberaums), so macht es Sinn, sich ein 'Epsilon' ϵ definieren, welches 'akzeptable Vorhersagefehler' definiert: Gilt

$$\sum_k (z_k^{p,i}(t_i) - (x_k^{p,i}((t+1)_i)))^2 < \epsilon,$$

so wird die Vorhersage als korrekt angesehen. Dies ist allerdings eine Heuristik, die auf der (häufig gemachten) Annahme basiert, daß eng benachbarte Eingabevektoren 'ähnliche' Bedeutung besitzen.

7.4 EXPERIMENTE MIT PREDIKTORENHIERARCHIEN

Das Multiebenensystem erlaubt, Sequenzen korrekt zu klassifizieren, die in der Praxis mit den zwar jungen, doch bereits mit dem Adjektiv 'herkömmlich' zu apostrophierenden KNN-Verfahren (Kapitel 2) trotz ihrer theoretischen Allgemeinheit *nicht* bei vernünftigem Zeitaufwand klassifizierbar sind.

Eine aus zwei Ebenen bestehende Prediktorenhierarchie produzierte innerhalb weniger hundert Trainingsepisoden eine korrekte Lösung der Standardaufgabe (für $m = 10$, siehe Abschnitt 2.6.4). Damit war sie mindestens 10^3 mal schneller als das konventionelle Verfahren. Um die Vorteile der selbstorganisierenden Prediktorenhierarchie noch besser würdigen zu können, machen wir beim nächsten Experiment einen quantitativen Sprung und beschäftigen uns mit sehr viel längeren zeitlichen Verzögerungen als denjenigen der Standardaufgabe.

7.4.1 EXPERIMENT: VERZÖGERUNGEN MIT MEHR ALS 1000 ZEITSCHRITTEN

Wir konstruieren eine stochastische Grammatik, i.e. eine Grammatik, bei der die Anwendung jeder Produktionsregel mit einer von vornherein fest-

gelegten Wahrscheinlichkeit stattfindet. Produziert Zeichenkette α die Zeichenkette β mit Wahrscheinlichkeit q, so schreiben wir formal

$$\alpha \rightarrow^{q} \beta.$$

N bezeichnet die Menge der Nichtterminalzeichen, T steht für die Menge der Terminalzeichen, λ stellt die leere Zeichenkette dar, S repräsentiert das Startsymbol.

Die Grammatik T_3 sei wie folgt definiert (Regeln werden durchnumeriert):

$$T = \{a, b, x, y, 1, 2\},$$

$$N = \{S, S', S'', A, A', A'', B, B', B'', C, C', C'', D, D', D''\},$$

$$(1)\ S \rightarrow^{0.25} xCCCS'1, \quad (2)\ S \rightarrow^{0.25} yCCCS'2,$$

$$(3)\ S \rightarrow^{0.25} xDDDS''1, \quad (4)\ S \rightarrow^{0.25} yDDDS''2,$$

$$(5)\ S' \rightarrow^{0.5} C, \quad (6)\ S' \rightarrow^{0.5} \lambda,$$

$$(7)\ S'' \rightarrow^{0.5} D, \quad (8)\ S'' \rightarrow^{0.5} \lambda,$$

$$(9)\ C \rightarrow^{1.0} ABABABABABABABABABABC',$$

$$(10)\ C' \rightarrow^{0.5} ABC', \quad (11)\ C' \rightarrow^{0.5} \lambda,$$

$$(12)\ D \rightarrow^{1.0} BABABABABABABABABABAD',$$

$$(13)\ D' \rightarrow^{0.5} BAD', \quad (14)\ D' \rightarrow^{0.5} \lambda,$$

$$(15)\ A \rightarrow^{1.0} abababababababababababA',$$

$$(16)\ A' \rightarrow^{0.5} abA', \quad (17)\ A' \rightarrow^{0.5} \lambda,$$

$$(18)\ B \rightarrow^{1.0} babababababababababaB',$$

$$(19)\ B' \rightarrow^{0.5} baB', \quad (20)\ B' \rightarrow^{0.5} \lambda. \qquad (7.5)$$

T_3 erzeugt eine unendliche Menge von Zeichenreihen, von denen jede mindestens die Länge $3*20*20+1+1 = 1202$ aufweist. Das erste Terminalsymbol jeder Zeichenkette ist entweder x oder y. Dann folgt eine sehr lange aus a's und b's bestehende Unterkette. Das letzte Terminalzeichen ist 1, falls das erste Terminalzeichen x war, und 2 sonst.

Wir nennen die am Ende jeder Zeichenreihe auftretenden Symbole 1 und 2 'Sequenzklassifikatoren'. Ziel unseres Systems soll die richtige Vorhersage der Sequenzklassifikatoren sein, ohne sie tatsächlich beobachtet zu haben.

Um dies zu erreichen, muß das System Ereignisse berücksichtigen, die zumindest 1200 Zeitschritte in der Vergangenheit zurückliegen.

Zunächst wurde wieder ein konventionelles rekurrentes Netz getestet. Die 6 verschiedenen Terminalzeichen wurden jeweils durch einen Bitvektor der Länge 6 mit einer einzigen Komponente $\neq 0$ repräsentiert. Auch die Netzwerkausgaben (die Vorhersagen) waren demgemäß 6-dimensional.

Dem konventionellen Netzwerk war es mit verschiedenen Anzahlen versteckter Knoten, verschiedenen Lernraten, und verschiedenen Gewichtsinitialisierungen zwischen -0.5 und 0.5 (wie nach den Experimenten aus dem letzten Unterabschnitt erwartet) *nicht* möglich, innerhalb von 10^6 Trainingssequenzpräsentationen die Sequenzklassifikatoren korrekt vorherzusagen. Ich vermute, daß auch 10^9 Trainingssequenzpräsentationen nicht ausgereicht hätten – mangelnde Rechenzeit verhinderte allerdings die Bestätigung dieses Verdachts.

Ein Multiebenensystem hingegen traf bei der Lösung der Aufgabe auf keine nennenswerten Schwierigkeiten. Es wurde eine aus drei Ebenen bestehende Prediktorenhierarchie verwendet. Alle drei Prediktoren P_1, P_2, P_3 ließen sich als rekurrente Netze implementieren, die durch 'abgeschnittenes BPTT' (siehe Kapitel 2) trainiert wurden, wobei die zu jedem Zeitschritt jeder Zeitskala auftretenden Fehlersignale lediglich 3 Zeitschritte (!) 'in die Vergangenheit' zurückpropagiert wurden (mehr waren bei diesem Experiment nicht nötig). Alle 3 Prediktoren besaßen jeweils 6 Eingabeknoten, je einen für die 6 verschiedenen Terminalzeichen (für das Experiment waren *keine* eindeutigen Zeitschrittrepräsentationen erforderlich). Als deterministische Ausgabe jedes Prediktors P_i wurde derjenige Bitvektor der Länge 1 angesehen, der der eigentlich reellwertigen Ausgabe von P_i am nächsten war. Jeder Prediktor verfügte über 3 versteckte Knoten. Alle Gewichte wurden zufällig zwischen -0.1 und 0.1 initialisiert.

Zunächst wurde P_1 mittels 333 gemäß (7.6) generierter Trainingssequenzen trainiert (weniger hätten auch gereicht). Nach Einfrieren von P_1's Gewichten fand P_2's Trainingsphase mit ebenfalls 333 Sequenzpräsentationen statt. Schließlich dienten 333 Sequenzen zur Adjustierung von P_3's Gewichten. Alle drei Netze verwendeten eine Lernrate von 0.333. Die insgesamt 999 Sequenzpräsentationen genügten, dem System beizubringen, die Sequenzklassifikatoren korrekt vorherzusagen.

P_1 lernte im wesentlichen, daß a normalerweise von b gefolgt wird, und umgekehrt. Dies entspricht der lokalen in den Regeln (15) - (20) enthaltenen

Struktur. Da allerdings dank der Regeln (1) - (14) die von P_1 vorhergesagten Werte glegentlich falsch sind (manchmal treten 2 b's oder 2 a's in Folge auf, oder auch eines der Zeichen $x, y, 1, 2$), bekam P_2 durchschnittlich alle 21 Zeitschritte (auf P_1's Skala) eine von P_1 unerwartete Eingabe. P_2 lernte nun seinerseits die in seinen Eingaben enthaltene durch die Regeln (9) - (14) gegebene 'globalere Struktur'. Die einzigen von P_2 unerwarteten Ereignisse bestanden nun in die Anfängen und Enden der präsentierten Sequenzen. Für P_3 war es schließlich einfach, die nach den Regeln (1) - (8) existierende Beziehung zwischen Sequenzanfängen und -enden zu entdecken und die Sequenzklassifikatoren korrekt vorherzusagen. Weitere vergleichbare Experimente finden sich in [37].

Testaufgaben wie die soeben beschriebene zeigen, daß auf dem Prinzip der Geschichtskompression basierende Systeme in gewissen Trainingsumgebungen mindestens 1000 mal schneller lernen können als konventionelle rekurrente Netze. Dabei ist dies noch eine sehr vorsichtige Abschätzung – aufgrund mangelnder Rechnerzeit war es nicht möglich, herauszufinden, wie lange ein konventionelles Netz zur Lösung obiger Aufgabe braucht. Weiterhin kann es sein, daß (wie im obigen Experiment) die an der Hierarchie beteiligten Prediktoren wesentlich weniger Operationen *pro Zeitschritt* benötigen als ein konventionelles rekurrentes Netzwerk – einfach deswegen, weil sie auf ihrer gedehnten Zeitskala Fehler häufig nur relativ wenige Zeitschritte in die Vergangenheit zurückpropagieren müssen. In gewissen Umgebungen reichen also *lokale* Lernalgorithmen aus, um in der Hierarchie beliebig lange Zeitspannen zu überbrücken. In solchen Fällen lassen sich demnach zwei Fliegen mit einer Klappe schlagen.

Es sollte nun klar sein, daß man durch Wahl komplexerer Grammatiken als (7.6) die Vorteile einer Prediktorenhierarchie im Vergleich zu den 'konventionellen' Verfahren aus Kapitel 2 *beliebig* deutlich herausstellen kann – die hier an einem speziellen Beispiel demonstrierte mindestens 1000-fache Überlegenheit der neuen Methodik stellt bei weitem keine obere Schranke dar.

7.4.2 EXPERIMENTE ZUR TEXTKOMPRESSION

Die von der künstlichen stochastischen Grammatik aus obigem Experiment erzeugten Zeichenreihen lassen sich mit wenig Aufwand stark komprimieren. Der Grund hierfür liegt in der Tatsache, daß die in den Zeichenreihen enthaltene (Shannon-) Information angesichts der Sequenzlängen als sehr

niedrig zu bewerten ist.

Für auf natürlichem Wege (z.B. durch Zeitungsjournalisten) generierte Zeichenreihen trifft derartiges nicht (notwendigerweise) zu. Aus diesem Grunde eignen sich Experimente zur Kompression natürlichsprachlicher Texte mit relativ hohem Informationsgehalt auch zur Aufweisung der Schranken des in diesem Kapitel verfolgten Ansatzes.

In Zusammenarbeit mit Stefan Heil (Student an der TUM) wurden (und werden gegenwärtig) daher Versuche zur adaptiven Komprimierung deutscher Sprache durchgeführt. Der zugrundeliegende Datensatz bestand aus 25 Zeitungstexten aus der Zeitung 'Frankenpost'. Ein Text verfügte über zwischen 3000 und 10000 Zeichen.

Zur Bewertung der Ergebnisse bedienen wir uns des Begriffs des durchschnittlichen Kompressionsfaktors eines Textkompressionsalgorithmus. Darunter versteht man das durchschnittliche Verhältnis zwischen den Längen von Originaltexten und den Längen der entsprechenden komprimierten Texte. Um eine Vorstellung davon zu bekommen, was ein 'guter' Textkompressionsalgorithmus ist, sei ein Satz aus [34] zitiert:

'In general, good algorithms can be expected to achieve an average compression ratio of 1.5, while excellent algorithms based upon sophisticated processing techniques will achieve an average compression ratio exceeding 2.0.'

Der wohl am weitesten verbreitete Textkompressionsalgorithmus ist der Lempel-Ziv Algorithmus [158], welcher u.a. auch die Grundlage der *compress*-Funktion des Betriebssytems *UNIX* bildet. Der Lempel-Ziv Algorithmus ist in einem gewissen informationstheoretischen Sinne optimal [154]. Sein durchschnittlicher Kompressionsfaktor beträgt für englischen Text etwas mehr als 2.0.

In den Experimenten wurde wie folgt vorgegangen: Alle Kleinbuchstaben eines jeden Textes wurden in die entsprechenden Großbuchstaben umgewandelt. Als Prediktor P_1 der untersten (und bei diesem Experiment einzigen Stufe) wurde statt einem rekurrenten Netz der Einfachheit ein azyklisches BP-Netz und der 'Zeitfensteransatz' verwendet. Dabei versucht P_1 aus je n aufeinanderfolgenden Zeichen das n+erste vorherzusagen. 42 Zeichen eines jeden Textes (Buchstaben, Interpunktionszeichen und Ziffern) besaßen dabei jeweils eine eindeutige lokale Eingaberepräsentation als 43-dimensionale Binärvektoren, bei denen jeweils eine einzige Komponente gleich 1 und alle anderen Komponenten gleich 0 waren. Der dreiundvierzigste mögliche

Binärvektor der Länge 1 diente zur Repräsentation aller anderen Zeichen. Der Nullvektor diente zur Repräsentation des 'Defaultzeichens' – jedem Text wurden am Anfang n derartige Defaultzeichen vorangestellt.

P_1 verfügte über $43n$ Eingabeknoten, $43n$ versteckte Knoten und 43 Ausgabeknoten. Die Eingabelage war vollständig mit der 'versteckten Lage' vernetzt, die versteckte Lage war vollständig mit der Ausgabelage vernetzt. Zur Textverarbeitung wurde das Eingabefenster sequentiell über den Text geschoben – P_1 erhielt also zunächst die ersten n Zeichen als Eingabe, daraufhin das zweite bis zum n+ersten Zeichen, und so fort bis zum Textende. Als deterministische Vorhersage des Prediktors galt jeweils dasjenige Zeichen, dessen Repräsentation den geringsten euklidischen Abstand zum rellwertigen Ausgabevektor von P_1 besaß. Während der Textverarbeitung wurde ein komprimiertes File erzeugt, in das nur die vom Prediktor nicht vorhergesagten Zeichen sowie eine Repräsentation der Zahl der Zeichen, die seit dem jeweils letzten unvorhersagbaren Zeichen aufgetreten waren, geschrieben wurden. Aus dem komprimierten File ließ sich das Originalfile mit Hilfe des Prediktors durch eine entsprechende inverse 'uncompress'-Funktion also wieder vollständig rekonstruieren.

20 der 25 Textfiles dienten zum Training, 5 ausschließlich zum Testen der Performanz. In andauernden Versuchen sollen verschiedene Zeitfensterbreiten n getestet werden. Signifikante Resultate existieren wegen der zeitaufwendigen Trainingsphase bisher erst für $n = 5$: Bei einer Lernrate von 0.2 sowie bei 50 Durchgängen durch die Trainingsfiles lernte P_1, 60 Prozent der Buchstaben der Trainingstexte und 47 Prozent der Buchstaben der Testtexte korrekt vorherzusagen (interessant ist natürlich vor allem der Wert für die unbekannten Testtexte). Aufgrund des zusätzlichen Speicheraufwands für die Repräsentationen der Zahl der Zeichen, die seit dem jeweils letzten unvorhersagbaren Zeichen aufgetreten waren, entsprach dies allerdings nicht einer Kompression auf 53 Prozent der Originaltextlänge, sondern lediglich einer Verkürzung auf 64 Prozent. Durch anschließendes einfaches *'Huffman-Coding'* (siehe z.B. [34]) des komprimierten Files mit Hilfe der UNIX Funktion *pack* konnte jedoch für unbekannte Texte ein Kompressionsfaktor von insgesamt über 2 erreicht werden.

Schon für $n = 5$ schnitt dieses hybride Verfahren damit so gut oder gar besser ab als der Lempel-Ziv Algorithmus, welcher bei den Zeitungstexten auf einen Kompressionsfaktor von durchschnittlich nur knapp 2.0 kam.

Die durch das Netzwerk erzielten Resultate wurde auch verglichen mit durch Stefan Heil als menschlichem Testsubjekt erhaltenen Werten. Heil

versuchte sich darin, aus n aufeinanderfolgenden zufällig aus dem Datensatz ausgewählten Buchstaben den n+ersten zu prophezeihen. Er erhielt die folgenden Resultate: Für $n = 5$ 50 Prozent korrekte Vorhersagen, für $n = 10$ 60 Prozent korrekte Vorhersagen, für $n = 15$ 68 Prozent korrekte Vorhersagen.

Für $n = 5$ erreichte das adaptive Netzwerk also nicht ganz die Performanz des menschlichen Testers, die Ergebnisse waren jedoch zumindest vergleichbar. Es besteht die Hoffnung, daß KNN für $n > 5$ ähnlich gut mit der menschlichen Performanz mithalten können, was als Nebenprodukt dieser Untersuchungen ein neuartiges, für praktische Anwendungen interessantes Textkompressionsverfahren zur Folge hätte, welches manche Texte wesentlich besser komprimieren könnte als der Lempel-Ziv Algorithmus.

Obige Experimente zeigen, daß natürlichsprachliche deutsche Texte zwar einen bedeutenden Teil lokal entdeckbarer Redundanz aufweisen, daß die durch Entfernung lokal entdeckbarer Redundanz gewonnenen Kompressionserfolge jedoch erheblich hinter denen mit gewissen einfacheren künstlichen Satzgeneratoren zu erzielenden Resultaten (siehe den vorangegangenen Abschnitt) zurückstehen können. Dieses Ergebnis war zu erwarten: Je mehr algorithmische Information (siehe e.g. [17]) in gewissen Zeichenreihen enthalten ist, desto weniger lassen sie sich durch Geschichtskompression oder irgendein anderes Kompressionsverfahren komprimieren. Das Extrem ist natürlich durch auf rein zufälligem Wege generierte Zeichenreihen gegeben: In diesem Falle lassen sich auf Grund der maximalen algorithmischen Information i.a. überhaupt keine Kompressionserfolge erzielen.

Es bleibt zu untersuchen, welche Domänen der 'Echtwelt' sich besonders zur Redundanzreduzierung durch Geschichtskompression eignen. Ein hohes Maß an Redundanz ist beispielsweise in Schwarzweißbildern typischer visueller Szenen (und erst recht in typischen Sequenzen aufeinanderfolgender derartiger Schwarzweißbilder) enthalten. Nach kleineren, auf die Zweidimensionalität von Bilddaten zugeschnittenen Modifikationen des Prinzips der Geschichtskompression könnten sich hier vielversprechende Möglichkeiten ergeben.

7.5 KOLLAPS DER HIERARCHIE

Normalerweise stellt eine Prediktorenhierarchie mit mehreren Ebenen den sichersten und schnellsten Weg dar, um den Umgang mit Sequenzen mit

hierarchischer zeitlicher Struktur zu erlernen. Ein gewisser Nachteil des Multiebenensystems besteht jedoch darin, daß gewöhnlich nicht von vornherein bekannt ist, wieviele Ebenen benötigt werden. Ein weiterer Nachteil mag in gewissen Fällen darin gesehen werden, daß die Ebenen explizit voneinander getrennt sind, statt in ein und demselben Netzwerk repräsentiert zu werden.

Es ist jedoch bis zu einem gewissen Grad möglich, die Hierarchie in ein einzelnes Netz zu *kollabieren*. Der vorliegende Abschnitt liefert dazu Architektur und Zielfunktionen. Es soll allerdings gleich vorausgeschickt werden, daß wir hier zum ersten Mal ein mathematisch nicht exaktes Verfahren bekommen werden – es wird auf Gradientenabstieg in einer sich ändernden Funktion basieren und damit ein Potential für Instabilitäten aufweisen [110]. Ein Experiment mit unserer Standardaufgabe wird jedoch zeigen, daß auch die kollabierende Architektur konventionellen rekurrenten Netzen bei weitem überlegen sein kann.

7.5.1 PRINZIP

Hier sei nur die grundlegende Idee beschrieben – die Details finden sich im nächsten Abschnitt.

Wir benötigen zwei konventionelle rekurrente Netzwerke: den *Automatisierer* A und den *'Chunker'* C. Zu jedem Zeitschritt sieht A die gegenwärtige externe Eingabe. A's Zielfunktion enthält drei Terme: Ein Term zwingt A, gewisse von einem externen Lehrer zu bestimmten Zeitpunkten vorgegebene gewünschte Zielwerte auszugeben. Falls solche Zielwerte existieren, werden sie Teil der nächsten Eingabe. Der zweite Term zwingt A zu jedem Zeitpunkt, seine Umgebungseingabe (nicht die eventuell durch Zielwerte bestimmte Eingabe) vorherzusagen. Der dritte Term ist der eigentlich interessante und wird im übernächsten Absatz erklärt.

Dann und nur dann, wenn A eine nicht zutreffende Voraussage bezüglich des ersten und zweiten Terms seiner Zielfunktion macht, wird die falsch vorhergesagte Eingabe (einschließlich eines möglicherweise vorhandenen gewünschten Zielvektors) zusammen mit einer eindeutigen Repräsentation des gegewärtigen Zeitpunkts als neue Eingabe an C weitergeliefert. Bevor sie jedoch zur Aktualisierung von C führt, wird C daraufhin trainiert, die neue Eingabe aus seinem gegenwärtigen internen Zustand und seiner letzten Eingabe (die vor langer Zeit stattgefunden haben mag) vorherzusagen. Dazu

wird einer der konventionellen Algorithmen für rekurrente Netze verwendet (siehe Kapitel 2). Anschließend wird C gemäß der gewöhnlichen Aktivationsausbreitungsregeln aktualisiert, was zur Repräsentation der Geschichte aller vergangener Eingaben beiträgt. Man beachte, daß C nach dem Prinzip der Geschichtskompression eine eindeutige reduzierte Beschreibung der vergangenen Eingaben gewahrt. (Der Anfang einer Trainingsepisode ist i.a. nicht vorhersagbar, daher muß er ebenfalls an die zweite Ebene weitergeliefert werden).

Da die von C's Lernalgorithmus zu überbrückenden Zeitlücken im Vergleich zu denen von A oft kurz sind, wird C häufig nützliche interne Repräsentationen früherer unerwarteter Ereignisse entwickeln (hier wieder die Annahme, daß der aus der Umgebung kommende Eingabestrom sowohl lokale als auch globale zeitliche Struktur besitzt). Dank dem dritten Term seiner Fehlerfunktion wird A nun gezwungen, C's interne Repräsentationen zu *rekonstruieren.* Daher wird A selbst schon in einer frühen Phase der Verarbeitung einer gegebenen Sequenz nützliche interne Repräsentationen entwickeln können; A wird häufig bedeutungstragende Fehlersignale erhalten, lange bevor externe durch den ersten oder zweiten 'konventionellen' Term verursachte Fehler auftreten. Diese internen Repräsentationen tragen die diskriminierende Information über C's Zustand. Damit tragen sie auch die diskriminierende Information, die notwendig ist, um A's Vorhersagen auf der niedrigeren Ebene zu verbessern. Demzufolge wird C allerdings weniger und weniger Eingaben erhalten, da mehr und mehr Eingaben vom Automatisierer vorhersagbar werden. Dies ist die *Kollapsoperation.* Im Idealfall wird C nach einiger Zeit überflüssig werden.

Die Kollapsoperation ist mit einem Nachteil behaftet. Während bei der inkrementellen Kreierung einer Multiebenenhierarchie keine Ebene Einfluß auf die Vorgänge in einer niedrigeren Ebene ausüben kann (und damit kein Potential für Instabilitäten auftreten kann), ist die Kollapsarchitektur nicht frei von möglichen Fluktuationen. C's interne Repräsentationen führen zur Änderung von A's Gewichten, was seinerseits in der Regel zur Verminderung der Eingaben für C und damit meist auch zur Änderung C's interner Repräsentationen führt. Es sind nun Situationen denkbar, in denen A aufgrund des dritten Terms seiner Zielfunktion früher gelernte Vorhersagen wieder 'verlernt'. Ein *ad-hoc* Heilmittel bestünde in der sorgfältigen relativen Gewichtung der drei Terme in A's Zielfunktion. In dem weiter unten beschriebenen Experiment war keine relative Gewichtung notwendig.

Vektor	Beschreibung (bzgl. Zeitpunkt t)	Dimension
$x(t)$	'normale' Umgebungseingabe	n_I
$d(t)$	Zielvektor (vom Lehrer definiert)	n_D
$i_A(t) = x(t) \circ d(t)$	A's Eingabe	$n_I + n_D$
$h_A(t)$	A's versteckte Aktivationen	n_{H_A}
$d_A(t)$	A's Vorhersage von $d(t)$	n_D
$p_A(t)$	A's Vorhersage von $x(t)$	n_I
$time(t)$	eindeutige Repräsentation von t	n_{time}
$h_C(t)$	C's versteckte Aktivationen	n_{H_C}
$d_C(t)$	C's Vorhersage von C's nächster Zieleingabe	n_D
$p_C(t)$	C's Vorhersage von C's nächster 'normalen' Eingabe	n_I
$s_C(t)$	C's Vorhersage von C's nächster *'time'* Eingabe	n_{time}
$o_C(t)$	$d_C(t) \circ p_C(t) \circ s_C(t)$	$n_{O_C} = n_D + n_I + n_{time}$
$q_A(t)$	A's Vorhersage von $h_C(t) \circ o_C(t)$	$n_{H_C} + n_{O_C}$
$o_A(t)$	$d_A(t) \circ p_A(t) \circ q_A(t)$	$n_{O_A} = n_D + n_I + n_{H_C} + n_{O_C}$

Tabelle 7.1: *Definitionen von Symbolen, die zeitabhängige Aktivationsvektoren bezeichnen. $h_A(t)$ und $o_A(t)$ basieren auf vergangenen Eingaben und werden ohne Wissen über $d(t)$ und $x(t)$ berechnet.*

7.5.2 ARCHITEKTURDETAILS

Das hier beschriebene System sollte als *on-line* Repräsentant einer Menge von Variationen des in Abschnitt 7.5.1 beschriebenen Prinzips angesehen werden.

Tabelle 7.1 liefert einen Überblick über verschiedene zeitabhängige Aktivationsvektoren, die für die Beschreibung von Architektur und Algorithmus relevant sind. $\delta_d(t) = 1$, falls ein externer Lehrer zur Zeit t einen Zielvektor $d(t)$ bereitstellt, und 0 sonst. Falls $\delta_d(t) = 0$ gilt, nimmt $d(t)$ einen *'Defaultwert'* an, z.B. den Nullvektor.

A besitzt $n_I + n_D$ Eingabeknoten, n_{H_A} versteckte Knoten, und n_{O_A} Ausgabeknoten (siehe Tabelle 7.1). Bei reinen Vorhersageproblemen ist $n_D = 0$. C verfügt über n_{H_C} versteckte Knoten und n_{O_C} Ausgabeknoten. Alle Nichtausgabeknoten von A weisen gerichtete Verbindungen zu allen Nichteingabeknoten von A auf. A's Eingabeknoten besitzen gerichtete Verbindungen zu allen Nichteingabeknoten von C. Dies ermöglicht A's Eingabeknoten, zu bestimmten (kritischen) Zeitpunkten als Eingabeknoten für C zu fungieren. Weitere n_{time} Eingabeknoten für C dienen zur eindeutigen Repräsentation

'kritischer' Zeitschritte. Diesen zusätzlichen Eingabeknoten entspringen gerichtete Verbindungen zu allen Nichteingabeknoten von C. Schließlich sind alle versteckten Knoten von C Quellen gerichteter Verbindungen zu allen Nichteingabeknoten von C. Siehe Abbildung 7.2.

Falls $\delta_d(t) = 1$ gilt, versucht A, seine Vorhersage $d_A(t)$ dem Wert $d(t)$ anzugleichen. Weiterhin bemüht sich A, $p_A(t) = x(t)$ zu erreichen und damit $x(t)$ vorherzusagen. Hierbei sehen wir das Zielvorhersageproblem wieder als speziellen Fall eines Eingabevorhersageproblems an. Ist $\delta_d(t) = 1$ und konnte A $d(t)$ nicht korrekt vorhersagen, so versucht C, $d_C(t) = d(t)$ zu erreichen. C legt es weiterhin darauf an, $p_C(t) \circ s_C(t)$ der nächsten nicht vom Lehrer definierten Eingabe für C gleichzusetzen. Diese Eingabe mag unter Umständen noch weit in der Zukunft liegen. Schließlich versucht A, $q_A(t) = h_C(t) \circ o_C(t)$ zu erreichen und damit den Zustand von C zu rekonstruieren. Dies ermöglicht die Kollapsoperation. Die Aktivationen von C's Ausgabeknoten werden dabei als Teil von C's Zustand angesehen.

Sowohl C als auch A werden gleichzeitig durch einen konventionellen Algorithmus für rekurrente Netze trainiert. Häufig (siehe z.B. das in Abschnitt 7.4.1 besprochene Experiment) bietet sich 'abgeschnittenes BPTT' an (siehe Kapitel 2).

Die (inzwischen geläufige) Aktualisierungsprozedur für jedes der beteiligten rekurrenten Netze sieht wie folgt aus:

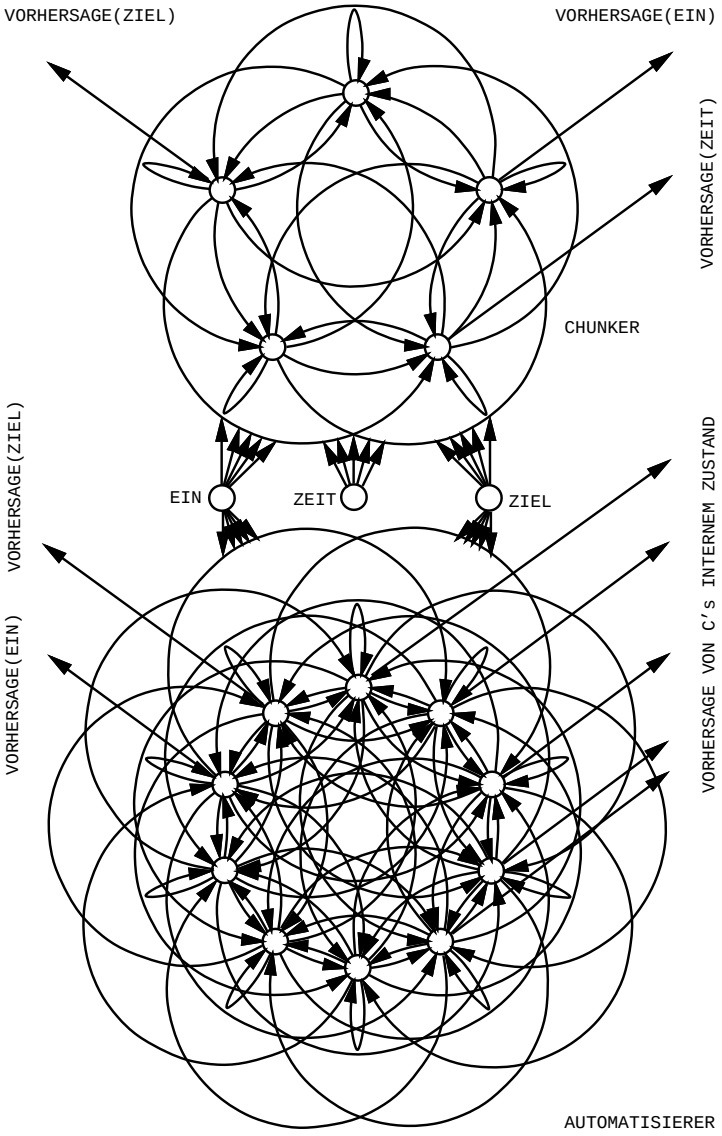

Abbildung 7.2: Zur Illustration der 2-Netz-Architektur: Der 'Automatisierer' A und der 'Chunker' C. Der besseren Übersichtlichkeit halber (sowie aus ästhetischen Gründen) sind nicht alle Verbindungen eingezeichnet. Siehe nähere Erläuterungen im Text.

Wiederhole für eine konstante Iterationsanzahl (typischerweise 1 oder 2):

1. Für jeden Nichteingabeknoten j aus N berechne $\hat{a}_j = f_j(\sum_i a_i w_{ij})$, wobei a_j die gegenwärtige Aktivation des j-ten Knotens, f_j eine semilineare differenzierbare Funktion und w_{ji} das Gewicht der gerichteten Verbindung vom Knoten i zum Knoten j ist.
2. Für alle Nichteingabeknoten j: Setze $a_j \leftarrow \hat{a}_j$.

Nun die Details des Ein-/Ausgabeverhaltens und der zu minimierenden Zielfunktionen (in pseudo-algorithmische Form gefaßt):

INITIALISIERUNG: Alle Gewichte werden zufällig initialisiert. Zu Beginn (zum Zeitpunkt 0) setze $h_C(0) \leftarrow 0$, $h_A(0) \leftarrow 0$, $i_A(0) \leftarrow d(0) \circ x(0)$. *Repräsentiere Zeitschritt 0 in* $time(0)$. *Aktualisiere* C, *um* $h_C(1)$ *und* $o_C(1)$ *zu erhalten.*

FR ALLE ZEITPUNKTE $t > 0$ *BIS ZUR EXTERNEN UNTERBRECHUNG:*
1. Aktualisiere A, *um* $h_A(t)$ *und* $o_A(t)$ *zu erhalten. A's Fehler* $e_A(t)$ *ist*

$$e_A(t) = \frac{1}{2}\|p_A(t) \circ q_A(t) - x(t) \circ h_C(t) \circ o_C(t)\|^2 + \delta_d(t)\frac{1}{2}\|d_A(t) - d(t)\|^2$$

Verwende einen der Gradientenabstiegsalgorithmen aus Kapitel 2, um jedes Gewicht w_{ji} *von* A *proportional zur Approximation von* $-\frac{\partial e_A(t)}{\partial w_{ji}}$ *zu ändern. Setze* $i_A(t) \leftarrow d(t) \circ x(t)$. *Repräsentiere* t *eindeutig in* $time(t)$.
2. Falls A's nicht *auf* C *bezogener Fehler*

$$e_P(t) = \frac{1}{2}\|p_A(t) - x(t)\|^2 + \delta_d(t)\frac{1}{2}\|d_A(t) - d(t)\|^2$$

kleiner oder gleich einer kleinen Konstate $\beta \geq 0$ *ist, setze* $h_C(t+1) = h_C(t)$, $o_C(t+1) = o_C(t)$.
Andernfalls definiere C's Vorhersagefehler als $e_C(t)$:

$$e_C(t) = \frac{1}{2}\|p_C(t) - x(t)\|^2 + \delta_d(t)\frac{1}{2}\|d_C(t) - d(t)\|^2 + \frac{1}{2}\|s_C(t) - time(t)\|^2,$$

verwende einen der Gradientenabstiegsalgorithmen aus Kapitel 2, um jedes Gewicht w_{ji} *von* A *proportional zur Approximation von* $-\frac{\partial e_C(t)}{\partial w_{ji}}$ *zu ändern, und aktualisiere* C, *um* $h_C(t+1)$ *und* $o_C(t+1)$ *zu erhalten.*

7.5.3 EXPERIMENT MIT DER KOLLAPS-ARCHITEKTUR

Josef Hochreiter (Diplomand an der TUM) implementierte Varianten des obigen Algorithmus und testete sie an der in Abschnitt 2.6.4 beschriebenen Standardaufgabe mit zeitlichen Verzögerungen der Länge $2m$. Zwar erfordert die Kollapsoperation mehr Trainingsaufwand, als zur Adjustierung einer nicht kollabierenden Multiebenenhierarchie notwendig ist. Auch

die Kollapsarchitektur kann jedoch 'konventionellen' Algorithmen für rekurrente Netze in zweifacher Hinsicht überlegen sein. Bei den Experimenten stellte sich heraus: (a) Die Anzahl der erforderlichen Rückpropagierungsschritte pro Zeitschritt war bei der Kollapsarchitektur deutlich geringer. (b) Die Anzahl der erforderlichen Trainingssequenzen betrug nur einen kleinen Bruchteil der entsprechenden Anzahl für die konventionelle Architektur.

Es galt wieder $m = 10$. A besaß einen versteckten Knoten, einen Eingabeknoten für jedes der Eingabesymbole $a, x, b_1, \ldots, b_{20}$, einen Eingabeknoten für den letzten extern vorgegebenen Zielwert, und einen stets mit 1 aktivierten Eingabeknoten, dessen ausgehende Verbindungen modifizierbare Schwellwerte für alle Nichteingabeknoten bereitstellten. C wurde aktualisiert, wann immer der an den von A nicht für die Vorhersage von C's Zustand verwendeten Ausgabeknoten beobachtete maximale Fehler den Wert 0.2 überstieg. C besaß einen versteckten Knoten und 23 Ausgabeknoten. Es waren *keine* eindeutigen Zeitschrittrepräsentationen $time(t)$ notwendig. A verfügte über 23 Ausgabeknoten, um die nächste Umgebungseingabe vorherzusagen, sowie 23 Ausgabeknoten, um den internen Zustand von C zu rekonstruieren. Pro Netzwerkaktualisierung wurde eine Iteration ausgeführt. Beide Netzwerke wurden durch 'abgeschnittenes BPTT' (Kapitel 2) trainiert. *Bei der Fehlerrückpropagierung wurden lediglich 3 Iterationen pro Zeitschritt ausgeführt, obwohl es darum ging, eine Lücke von 20 Zeitschritten zu überbrücken.* Die Lernraten beider Netze waren gleich 1.0.

17 Testläufe wurden durchgeführt. Bei 13 Testläufen erwiesen sich weniger als 5000 Trainingssequenzen als erforderlich, um den Fehler von A's Zielknoten stets kleiner als 0.12 zu machen. Für die übrigen 4 Testläufe ergaben sich folgende Zahlen als obere Schranken für die Zahl der Trainingssequenzen, die benötigt wurden, um den Fehler des Zielknotens unter den Wert 0.06 zu drücken: 30.000, 35.000, 25.000, 15.000.

Man vergleiche dies mit den in Abschnitt 2.6.4 gewonnenen Resultaten: Innerhalb von 10^6 Trainingsepisoden gelang es dem konventionellen Algorithmus nicht, dasselbe Problem zu lösen.

Typischerweise lernte A schnell, die leicht vorhersagbaren Symbole $b_2, \ldots, b_{20}$ zu prophezeihen. Dies führte zu einer stark reduzierten Eingabe für das C-Netz, welches nun bei der Aufgabe, die korrekten Zielwerte am Ende der Trainingssequenzen zu lernen, auf wenig Schwierigkeiten traf. Nach geraumer Zeit sah sich A imstande, C's interne Repräsentationen nachzuahmen, was A wiederum die Basis dafür lieferte, die korrekten Zielwerte selbst zu lernen.

Die endgültige Gewichtsmatrix von A verfügte über die Eigenschaften, die man erwarten würde: Typischerweise schaltete sich ein versteckter Knoten in dem Moment ein, in dem das Terminalzeichen a erschien. Eine starke rekurrente Verbindung dieses Knotens auf sich selbst hielt ihn für die folgenden 21 Zeitschritte 'am Leben', danach wurde der Knoten im Falle des Auftretens eines x's inhibiert. (Es traten auch ähnliche symmetrische Lösungen auf.)

Bemerkenswerterweise evolvierte diese 20 Zeitschritte überbrückende Struktur trotz der Tatsache, daß, wie erwähnt, nur 3 Fehlerrückpropagierungsiterationen pro Zeitschritt ausgeführt wurden.

7.5.4 SCHRANKEN DER KOLLAPS-ARCHITEKTUR

Mit der Prediktorenhierarchie ließ sich das Standardproblem natürlich wesentlich schneller (innerhalb weniger 100 Trainingsequenzen) lösen als mit der Kollapsarchitektur, welche die meiste Zeit der Trainingsphase für den Kollabiervorgang selbst verwendete. Für die in Abschnitt 7.4.1 beschriebene Aufgabe (mit zeitlichen Verzögerungen von mehr als 1000 Zeitschritten) fand die Kollapsarchitektur innerhalb von 10^6 Trainingssequenzen keine Lösung – damit lernte sie bei diesem Problem mindestens 1000 mal langsamer als die hierarchische Architektur. Ferner ist die hierarchische Architektur, wie bereits des öftern erwähnt, im Gegensatz zur Kollapsarchitektur *sicher*, da ein Prediktor der Hierarchie keinen potentiell destabilisierenden Einfluß auf Prediktoren niederer Ebenen nehmen kann. Ein Nachteil der Hierarchie besteht allerdings darin, daß die Informationsverarbeitung nach der Trainingsphase mehr als ein Netzwerk erfordert – was die Kollapsarchitektur trotz ihrer Schranken interessant erscheinen läßt.

7.6 KONTINUIERLICHE GESCHICHTSKOMPRESSION

Das oben verwendete Prinzip zur Geschichtskompression definiert Vorhersagefehler oder *'Mismatche'* in einer binären Art und Weise: Jeder Eingabeknoten, dessen Aktivation zu einem bestimmten Zeitpunkt nicht vorhersagbar war, zeigt ein unerwartetes Ereignis an. Jedes unerwartete Ereignis provoziert die Aktualisierung des internen Zustandes eines Prediktors auf

höherer Ebene. Es gibt kein Konzept eines partiellen *'Mismatches'*, oder einer 'nahezu richtigen' Vorhersage. Es ist nicht möglich, ein Netz der höheren Ebene in Antwort auf ein *'beinahe erwartetes'* Ereignis *'nur ein klein wenig'* zu aktualisieren.

'Kontinuierliche Geschichtskompression' [119] begegnet dieser Kritik durch die folgenden Ideen (wir halten uns an die Notation aus Abschnitt 7.2):

Wir bedienen uns lokaler Eingaberepräsentation. Die Komponenten von $z^p(t)$ werden so normiert, daß ihre Summe 1 ergibt (siehe z.B. Abschnitt 5.3). $z^p(t)$ wird als Vorhersage der Wahrscheinlichkeitsverteilung der möglichen Eingaben $x^p(t+1)$ interpretiert: $z_j^p(t)$ ist die Vorhersage der Wahrscheinlichkeit, daß $x_j^p(t+1)$ gleich 1 ist.

Die *Entropie* der Ausgabeknoten zur Zeit t,

$$-\sum_j z_j^p(t) log\ z_j^p(t), \tag{7.6}$$

läßt sich als Maß der Sicherheit des Prediktors ansehen. (7.7) ist maximal, wenn jedes mögliche Ereignis mit gleicher Wahrscheinlichkeit erwartet wird (höchste Unsicherheit des Prediktors). (7.7) ist minimal, wenn ein einziges Ereignis nahezu mit Wahrscheinlichkeit 1 erwartet wird (höchste Sicherheit des Prediktors).

Wieviel Information (relativ zum gegenwärtigen Prediktor) wird dem lernenden System durch das Ereignis $x_j^p(t+1) = 1$ übermittelt? Nach Shannon [125] lautet die Antwort

$$-log\ z_j^p(t).$$

Im folgenden werden wir eine Architektur entwerfen, die (im Shannonschen Sinne) *informative* Ereignisse stärkeren Einfluß auf die Geschichtsrepräsentation nehmen läßt als *weniger informative* Ereignisse. Die 'Stärke' einer Netzwerkaktualisierung ist dabei eine monoton wachsende Funktion der vom beobachteten Ereignis übermittelten Information.

7.6.1 KONTINUIERLICHE GESCHICHTSKOMPRESSION MIT RAAMs

Die in den Abschnitten 2.3, 2.4 und 2.5 behandelten Verfahren berechnen den exakten Gradienten der Zielfunktion bezüglich der Gewichtsmatrix. Sie

zielen darauf ab, nur diejenigen Eingabeereignisse zu berücksichtigen, die für die Minimierung der Fehlertrajektorie relevant sind.

Als interessanten Kontrast benützen wir im folgenden eine alternative Methode zur Sequenzspeicherung, die zwar in limitierter Weise Gradienteninformation verwendet, allerdings nicht den exakten Performanzgradienten berechnet (und damit weniger mathematische Rechtfertigung besitzt als die ersten drei Verfahren).

Was die Methode hier jedoch interessant macht, ist die Tasache, daß sie uns erlaubt, im Gegensatz zur Kollapsarchitektur aus Abschnitt 7.5 mit nur einem einzigen Netzwerk (statt mit zwei Netzwerken) zur Ereignisspeicherung auszukommen (eine weitere interessante Eigenschaft der Methode besteht darin, daß die Berechnungskomplexität pro Verbindung und pro Zeitschritt einer Trainingssequenz unabhängig von der Netzgröße und der Sequenzlänge konstant ist). Ein Experiment wird zeigen, daß dieses alternative System in der Tat einen noch schnelleren Lernprozeß erlauben kann als die ohnehin schon relativ schnelle Kollapsarchitektur.

Das Verfahren beruht auf einem erstmals von Pollack vorgeschlagenem Prinzip [74], dem rekursiven auto-assoziatives Gedächtnis (RAAM – für *'recursive auto-associative memory'*):

REKURSIVE AUTO-ASSOZIATIVE GEDÄCHTNISSE

Ein noch zu spezifizierender 'Autoassoziator' A wird daraufhin trainiert, jede Eingabesequenz sowie all ihre Anfangssequenzen in eindeutiger Weise zu repräsentieren (dies ist eine Form der Quellenkodierung). Ein zusätzliches überwacht lernendes *azyklisches* Netzwerk erhält A's eindeutige Repräsentationen als Eingabe und lernt, ein durch allgemeine Fehlertrajektorien definiertes Performanzmaß zu minimieren[6].

Wir fokussieren uns hier auf den Autoassoziator A. A's einziges Ziel besteht in der Kreierung unterschiedlicher interner Zustandsvektoren in Antwort auf unterschiedliche Eingabesequenzen.

Architektur und Zielfunktion. Es gibt drei Klassen von Knoten: Eingabeknoten, 'versteckte' Knoten, und Ausgabeknoten. Alle Eingabeknoten von A

[6]Diese für überwachtes Lernen notwendige Modifikation wurde von Pollack allerdings nicht berücksichtigt.

sind mit allen versteckten Knoten verbunden. Alle versteckten Knoten weisen Verbindungen zu allen Ausgabeknoten auf. A's interner Zustandsvektor $h^p(t)$ ist der Aktivationsvektor seiner versteckten Knoten zum Zeitpunkt t der sich über n_p Zeitschritte erstreckenden Sequenz p. A's Eingabe zur Zeit $t > 1$ ist $h^p(t-1) \circ x^p(t)$. Für alle p nimmt der interne Initialzustand $h^p(0)$ zum Zeitpunkt 0 einen *'Defaultwert'* an, z.B. den Nullvektor. Zur Zeit $0 < t \leq n_p$, berechnet A

$$h^p(t) = g(x^p(t), h^p(t-1)),$$

wobei g durch die konventionellen Aktivationsausbreitungsregeln für BP-Netze definiert ist (siehe Kapitel 1). A's Ausgabevektor $z^p(t)$ wird aus $h^p(t)$ ebenfalls gemäß der BP-Aktivationsausbreitungsregeln bestimmt. Die Anzahl von A's Ausgabeknoten ergibt sich zur Summe der Anzahlen seiner Eingabeknoten und versteckten Knoten. $z^p(t)$ wird nun als 'Rekonstruktion' von $x^p(t) \circ h^p(t-1)$ interpretiert. Siehe Abbildung 2.5.

Durch konventionelles BP wird g so modifiziert, daß die $h^p(t), 0 < t < n_p$ Werte annehmen, die es erlauben, $h^p(t-1)$ und $x^p(t)$ zu rekonstruieren. A's Zielfunktion zur Zeit t der Sequenz p ist dabei

$$E_A(t) = \frac{1}{2}\left[x^p(t) \circ h^p(t-1) - z^p(t)\right]^T \left[x^p(t) \circ h^p(t-1) - z^p(t)\right].$$

Zur Minimierung von $E_A(t)$ wird *nicht* etwa wie bei BPTT bis zum Beginn der gegenwärtig behandelten Sequenz zurückpropagiert, sondern lediglich bis zu $h^p(t-1)$. Damit bekommen wir zwar keinen exakten Gradientenabstieg in $E_A = \sum E_A(t)$, wohl aber einen Algorithmus, (im wesentlichen den von Pollack vorgeschlagenen[7]), dessen Berechnungskomplexität pro Verbindung und Zeitschritt unabhängig von der Netzgröße konstant ist. Warum sollte dieser Algorithmus A dazu zwingen, eindeutige interne Zustände für theoretisch beliebig lange Sequenzen und all ihre Subsequenzen zu generieren?

Die Antwort läßt sich leider nur informell durch Induktion über die Länge der längsten Trainingssequenz sehen (hier haben wir ein Beispiel für einen Lernalgorithmus, der *nicht* allein aus der Kettenregel gerechtfertigt werden kann):

1. Nehmen wir an, es gibt s verschiedene Trainingssequenzen $1, \ldots, s$. Die Länge der Sequenz p beträgt $n_p > 0$. Für alle $p = 1, \ldots, s$ wird $h^p(1)$

[7] In seiner Arbeit erwähnte Pollack dieses Potential für lokale Algorithmen nicht – der Fokus seiner Untersuchungen war ein ganz anderer.

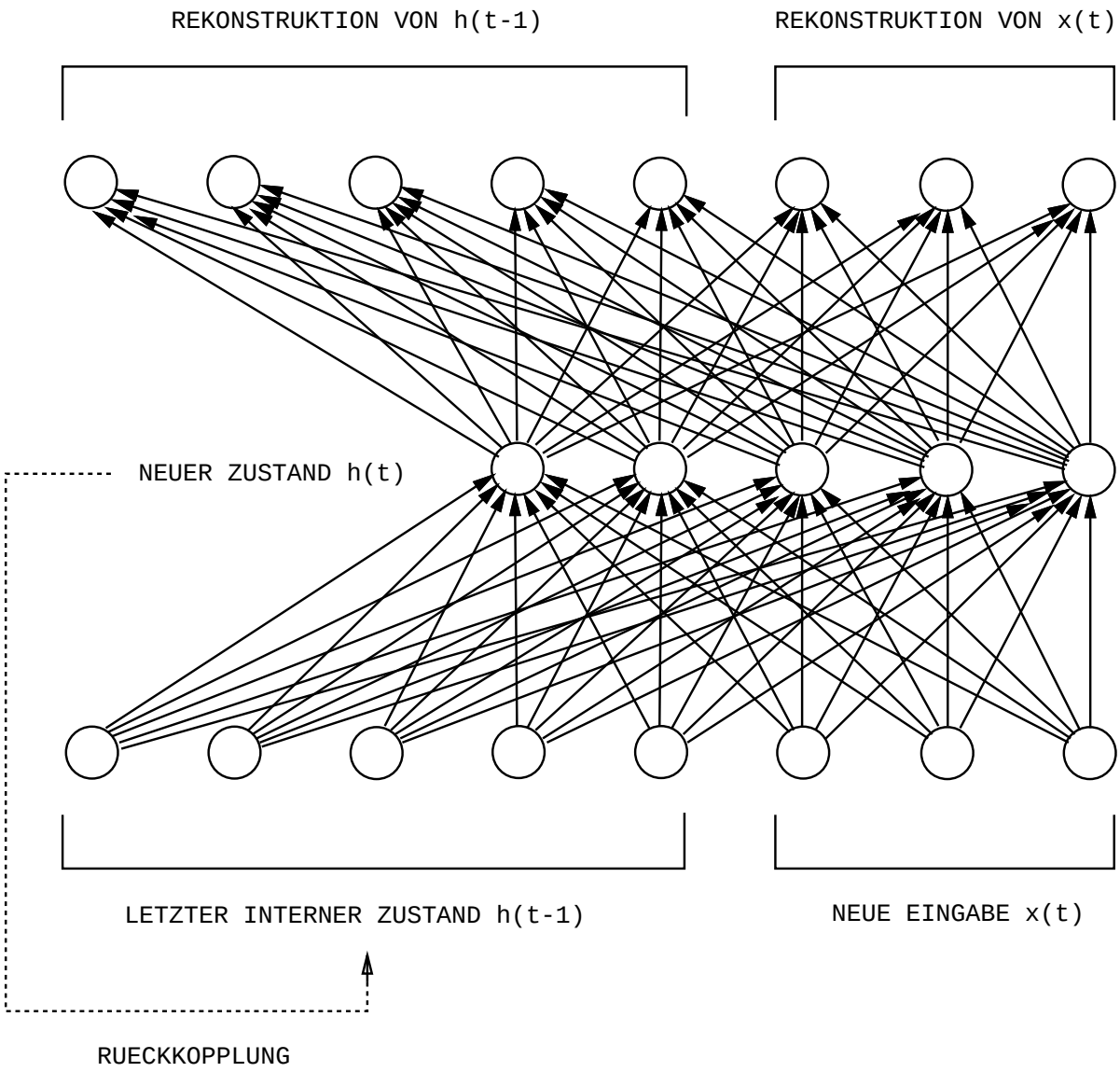

Abbildung 7.3: Ein rekursives auto-assoziatives Gedächtnis assoziiert die Kombination des letzten internen Zustands und der neuen Eingabe mit sich selbst – dadurch wird eine Beschreibung der bisher beobachteten Eingabesequenz in den neuen internen Zustand 'hineinkomprimiert'.

daraufhin trainiert, die Rekonstruktion von $h^p(0)$ und $x^p(1)$ zu ermöglichen. Demzufolge werden die Anfänge aller Sequenzen eindeutig in h repräsentiert sein.

2. Setzen wir nun voraus, daß alle Sequenzen und Subsequenzen der Länge $< k$ bereits eindeutige Repräsentationen in h verursachen. Für alle Sequenzen und Subsequenzen p mit Länge k zwingen wir $h^p(k)$, die Rekonstruktion von $h^p(k-1)$ and $x^p(k)$ zu ermöglichen. Damit werden alle Sequenzen und Subsequenzen mit Länge $< k+1$ eindeutige Repräsentationen in h nach sich ziehen. □

Obiges Argument vernachlässigt allerdings ein Potential für sogenannten *'crosstalk'*, z.B. die durch den zweiten Schritt eröffnete Möglichkeit, daß Sequenzen der Länge $< k$ durch Eintrainieren der Sequenzen der Länge k wieder 'vergessen' werden. In Experimenten zeigt sich jedoch, daß RAAMs zur Sequenzcodierung durchaus geeignet sind.

AUCH RAAMS HABEN PROBLEME MIT LANGEN ZEITLÜCKEN

Daniel Prelinger (Diplomand an der TUM) führte mit RAAMs zum Standardproblem (Abschnitt 2.6.4) analoge Experimente durch. Letztere erwiesen, daß RAAMs Zeitlücken zwischen korellierten Ereignissen in den Trainingssequenzen noch überbrücken können, wenn die Anzahl der Zeitschritte 5 nicht übersteigt. Auch RAAMs scheitern in praktischen Anwendungen jedoch an der Aufgabe, in vernünftiger Zeit hinreichend unterschiedliche Repräsentationen von Sequenzen (samt all ihrer Subsequenzen) zu kreieren, wenn diese mehr als 10 Zeitschritte umfassen [75].

Dies liefert die Motivation dafür, RAAMs gemäß dem Prinzip der kontinuierlichen Geschichtskompression dergestalt zu modifizieren, daß sie wesentlich längere Zeitspannen zwischen signifikanten Ereignissen überbrücken können. Der folgende Unterabschnitt zeigt, wie zur Erreichung dieses Zwecks vorzugehen ist.

7.6.2 ARCHITEKTUR UND ZIELFUNKTIONEN

Zum Zeitpunkt t der Sequenz p ist $H^p(t) \circ X^p(t)$ die (weiter unten zu definierende) Eingabe eines Autoassoziators A mit n_H versteckten Knoten,

$n_H + n_I$ Eingabeknoten, und $n_H + n_I$ Ausgabeknoten. A's interner Zustandsvektor (ablesbar von einem 'Flaschenhals' versteckter Knoten) heißt $h^p(t)$. A's $n_H + n_I$-dimensionaler Ausgabevektor wird mit $A^p(t)$ bezeichnet. Zu jedem Zeitschritt $t > 0$ versucht A mittels BP, seine eigene Eingabe zu rekonstruieren. A's Zielfunktion ist dabei

$$E_A(t) = \frac{1}{2}(H^p(t) \circ X^p(t) - A^p(t))^T(H^p(t) \circ X^p(t) - A^p(t)).$$

Der n_h-dimensionale Vektor $H^p(t)$ (aus gleich offensichtlich werdenden Gründen A's 'reduzierter Zustand' genannt) nimmt für $t = 0$ einen *'Defaultwert'* an, z.B. den Nullvektor. Das gleiche gilt für den n_i-dimensionalen Vektor $X^p(t)$.

Zum Zeitpunkt t der Sequenz S_p erhält ein azyklisches BP-Netz P den Vektor $H^p(t) \circ x^p(t)$ als Eingabe. Um die Dinge nicht über Gebühr zu verkomplizieren, nehmen wir auf (theoretisch eigentlich notwendige) eindeutige Zeitrepräsentationen keine Rücksicht. P's n_I-dimensionaler Ausgabevektor $P^p(t)$ soll nach der Trainingsprozedur die Wahrscheinlichkeitsverteilung der möglichen $x^p(t+1)$ approximieren. Daher wird $P^p(t)$ so normalisiert, daß stets $\sum_i P_i^p(t) = 1$ gilt (siehe z.B. Abschnitt 5.3). Für $t > 1$ sind $H^p(t)$ und $X^p(t)$ folgendermaßen definiert:

$$H_i^p(t) = (1 - \tau^p(t))H_i^p(t-1) + \tau^p(t)h_i^p(t-1)$$

und

$$X_i^p(t) = (1 - \tau^p(t))X_i^p(t-1) + \tau^p(t)x_i^p(t),$$

wobei $\tau^p(t)$ eine monoton wachsende Funktion von $-log\ P_j^p(t-1)$ ist, z.B. $\tau^p(t) = P_j^p(t-1)$, wobei j so gewählt ist, daß $P_j^p(t-1)$ die von P vermutete Wahrscheinlichkeit von $x^p(t)$ ist. Betrachte hierzu Abbildung 7.3.

Der Effekt dieses Vorgehens ist: Solange sich die nächste Eingabe aus der vorherigen Eingabe und dem vorherigen reduzierten Zustand als (nahezu) vorhersagbar erweist, bleibt die Eingabe des Autoassoziators im wesentlichen invariant. Nur die wirklich unerwarteten Ereignisse generieren neue Zielwerte für A – damit wird A dazu angehalten, ausschließlich informationstragende Ereignisse in seine internen Repräsentationen einzubinden. Es sollte erwähnt werden, daß obige Methode wiederum nur eine von mehreren Möglichkeiten darstellt, das grundlegende Prinzip zu implemetieren.

7.6.3 EXPERIMENT MIT DER RAAM-ARCHITEKTUR

Vorbemerkung: Verwendet man die in Abschnitt 7.6 geforderten lokalen Eingaberepräsentationen, bereitet das Ablesen der von einem Ereignis über-

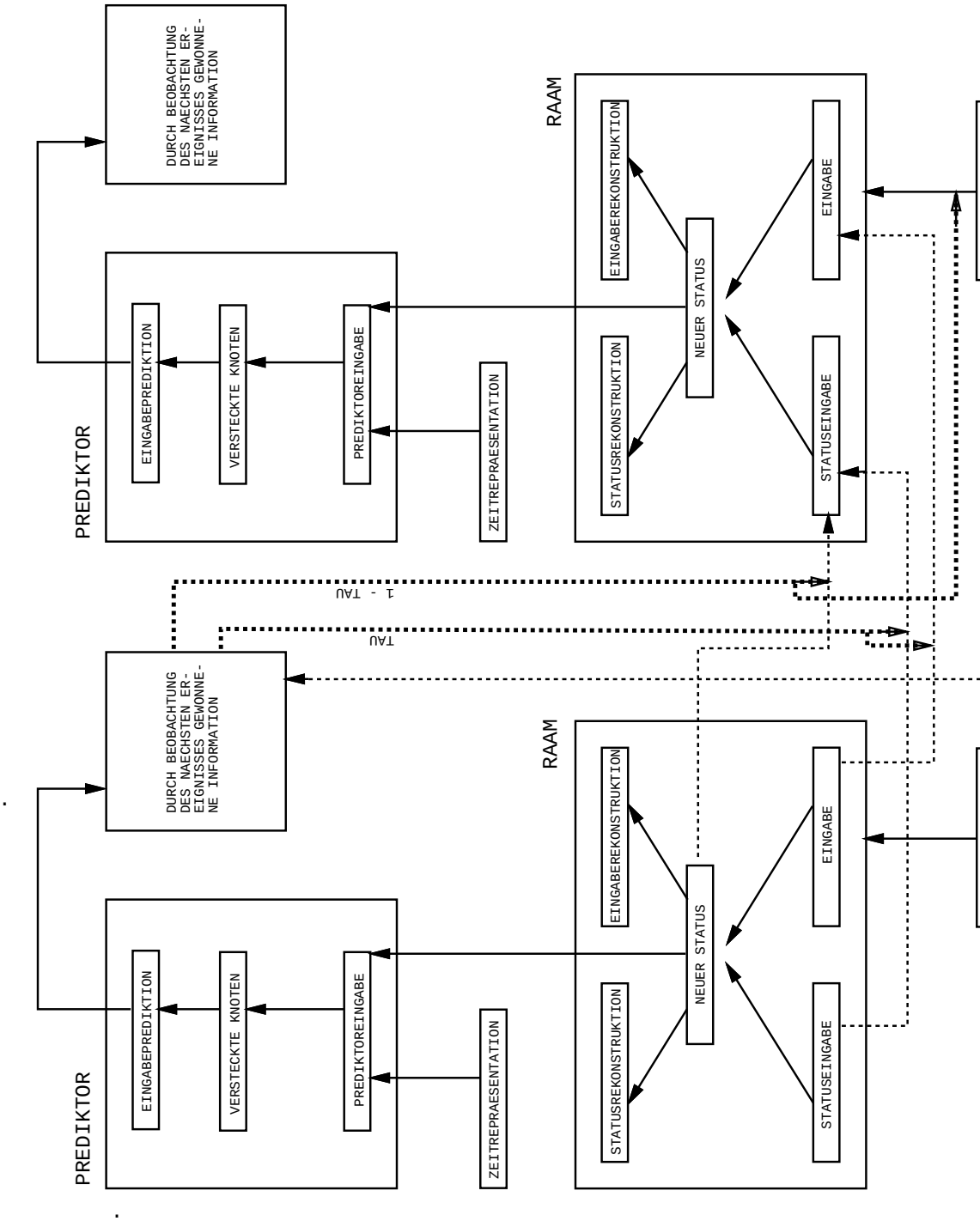

Abbildung 7.4: Kontinuierliche Geschichtskompression: Informationsfluß während zweier sukzessiver Zeitschritte. Siehe Text für Details.

mittelten Information aus der vom Prediktor vorhergesagten Wahrscheinlichkeitsverteilung keine Schwierigkeiten. Im Falle verteilter reellwertiger Eingaben ist etwas Derartiges i.a. nicht möglich.

Eine für solche Fälle hier vorgeschlagene Heuristik besteht darin, die 'Stärke' einer Netzwerkaktualisierung durch eine lineare monoton wachsende Funktion des gegenwärtigen MSE-Fehlers eines alle Ereignisse vorhersagenden Prediktors bestimmen zu lassen. Dieser Heuristik liegt erneut die Annahme zugrunde, daß zwei (im Sinne euklidischer Distanz) ähnliche Eingaben auch 'ähnliche' Bedeutung haben.

Bei dem im folgenden beschriebenen Experiment wurden P's Ausgabeknoten nicht normalisiert, und $\tau^p(t)$ war einfach eine monoton wachsende Funktion des gegenwärtigen Prediktionsfehlers. Diese heuristische Vereinfachung wurde durch brauchbare Resultate gerechtfertigt.

Das System wurde von Daniel Prelinger (Diplomand an der TUM) an unserer im Abschnitt 2.6.4 beschriebenen Standardaufgabe mit Zeitverzögerungen der Länge m getestet [119][75].

Das Fehlerkriterium wurde der Natur des verwendeten RAAMs angepaßt: Die Aufgabe galt als gelöst, wenn die euklidische Distanz zwischen den internen Repräsentationen der beiden verschiedenen Sequenzen größer als 0.5 war. Der Prediktor verfügte über keine versteckten Knoten, n_H war gleich 20. Alle Gewichte wurden zufällig zwischen -0.3 und 0.3 initialisiert. Alle Lernraten besaßen den Wert 1.0.

Das System benötigte bei $m = 20$ (dem Wert, bei dem die 'konventionellen' Netze bereits scheiterten) weniger als 600 Trainingssequenzen, um die Aufgabe zu lösen. Bemerkenswerterweise liegt dieser Wert noch deutlich unter dem Durchschnittswert für die Kollapsarchitektur (siehe Abschnitt 7.5.3). Die Beschreibung weiterer Experimente findet sich in [119] und [75].

Schranken des RAAM-basierten Ansatzes. Für die in Abschnitt 7.4.1 beschriebene Aufgabe (mit zeitlichen Verzögerungen von mehr als 1000 Zeitschritten) fand auch die RAAM-Architektur (ebenso wie die Kollapsarchitektur aus Abschnitt 7.5) innerhalb von 10^6 Trainingssequenzen keine Lösung. Für praktische Anwendungen größeren Maßstabs ist also die weniger elegante, aber erfolgsträchtigere Prediktorenhierarchie aus Abschnitt 7.3 vorzuziehen.

7.7 SCHLUSSBEMERKUNGEN

Es scheint, daß menschliche Subjekte sich vor allem auf unerwartete und atypische Ereignisse konzentrieren. Häufig versucht man, neue atypische Ereignisse durch früher beobachtete atypische Ereignisse zu erklären. Im Lichte des Prinzips der Geschichtskompression erscheint dies als eine höchst sinnvolle Verhaltensweise.

Sobald ehemals unerwartete Ereignisse zu erwarteten Ereignissen werden, tendieren sie dazu, ins 'Unterbewußte' abzugleiten. Man könnte hier eine Analogie zum Ablauf der verschiedenen in diesem Kapitel besprochenen Algorithmen ziehen: Die 'Aufmerksamkeit' der höchsten Ebene wird von den erwarteten Ereignissen abgelenkt und fokussiert sich auf die als neuartig empfundenen Ereignisse – erstere werden dabei 'automatisiert' und erlauben somit der obersten Schicht (entfernt dem 'Bewußtsein' vergleichbar) 'Abstraktionen' auf höherer Ebene. Eine derartige Betrachtung lieferte in der Tat die Inspiration für die Bezeichnung 'Automatisierer' in der Kollapsarchitektur.

Im Kontext sequenzverarbeitender KNNs macht das Prinzip der unüberwachten Geschichtskompression die Kettenregel zwar nicht obsolet, weist ihr jedoch eine weniger zentrale Rolle zu. Das Prinzip gibt Anlaß zu neuen Architekturen und Zielfunktionen, die die Kettenregel in einer eher lokalen (statt wie früher globalen) Manier zum Einsatz bringen. Welche Ereignisse von auf der Kettenregel basierenden Gradientenabstiegsalgorithmen überhaupt noch berücksichtigt werden, wird mit Hilfe des Kompressionsprinzips entschieden. Dadurch können, wie gesehen, unter Umständen gewaltige Ressourceneinsparungen erzielt werden. Die zentrale Botschaft dieses Kapitels lautet: Unüberwachte Performanzmaße zur Entdeckung kausaler Regelmäßigkeiten im Eingabestrom können zielgerichtetes Lernen in dynamischen Umgebungen unterstützen.

Die Anwendbarkeit des Prinzips der Geschichtskompression beschränkt sich nicht auf neuronale Netzwerke. Jeder adaptive sequenzverarbeitende Mechanismus könnte (und sollte vielleicht auch) davon Gebrauch machen.

Kapitel 8

AUSBLICK: 'SELBS-TREFERENTIELLE' NEURONALE NETZE

Ein wesentlicher Unterschied zwischen 'menschlichem' Lernen und existierenden Methoden für maschinelles Lernen (einschließlich der Methoden aus den vorangegangenen Kapiteln) besteht darin, daß Menschen über ihr eigenes Lernverhalten reflektieren und ihre Lernstrategien gegebenenfalls ändern können, um sie den von der Umgebung typischerweise gestellten Lernaufgaben anzupassen[1]. Aus diesem Grunde macht das vorliegende Kapitel (der letzte originäre Beitrag dieser Arbeit) im Rahmen eines Gedan-

[1] Wir können Aussagen über den Effekt unserer eigenen Lernprozeduren machen, wie beispielsweise in dem Satz "Heute werde ich mich hinsetzen und 20 japanische Wörter lernen." Wir reden nicht nur darüber, wie wir lernen, sondern verfügen auch über die Fähigkeit, unser Lernverhalten explizit zu ändern und den Gegebenheiten der Umgebung anpassen. Wir lernen beispielsweise, unsere eigenen Trainingsbeispiele auszusuchen. Wir lernen, Einfluß darauf zu nehmen, welche Assoziationen wir abspeichern. Für viele Problemklassen entwickeln wir spezifische interne Problemrepräsentationen und effiziente Lernstrategien. Diese Strategien mögen bestimmte Formen des 'Analogielernens', *'chunkings'*, *'one-shot'*-Lernens, etc. beinhalten. Es gibt jedoch Grund dafür, anzunehmen, daß sich viele spezifische Lernstrategien *nicht* adäquat durch eine dieser geläufigen Bezeichnungen beschreiben lassen. Nach einigen Jahren der Selbstbeobachtung gelangte der Autor zu der Überzeugung, daß Prozesse für 'Lernen lernen' bereits auf sehr niedrigem kognitiven Niveau für das Skalierungsverhalten großer Lernsysteme von essentieller Bedeutung sind. Dieses Kapitel will seine Motivation jedoch nicht aus den dubiosen introspektiven Erfahrungen des Autors schöpfen.

kenexperiments einige anfängliche theoretische Schritte[2] in Richtung *'selbstreferentielles'* und *'introspektives'* maschinelles Lernen.

Wir haben in den Kapiteln 2 bis 7 gesehen, daß sich gradientenbasierte Gewichtsänderungsalgorithmen für sehr unterschiedliche Problemstellungen aus den Bereichen überwachtes Lernen, unüberwachtes Lernen und 'Reinforcement'-Lernen herleiten lassen. Allen bisher in dieser Arbeit (oder in der KNN-Literatur) aufgetauchten KNN ist allerdings eines gemeinsam: Zumindest einige ihrer adaptiven Parameter lassen sich ausschließlich durch fest 'vorverdrahtete' Lernalgorithmen ändern. Dies gilt auch für das System aus Kapitel 3. Das vorliegende Kapitel geht nun einen Schritt weiter und beantwortet in konstruktiver Weise folgende Fragen: (1) Ist es möglich, ein KNN zu entwerfen, das im Prinzip alle eigenen Gewichtsparameter selbst aktiv modifizieren kann? (2) Läßt sich das Konzept des Gradientenabstiegs im Prinzip auch zur Auffindung geeigneter *Lernalgorithmen* anwenden?

Das in den folgenden Abschnitten als Gedankenexperiment vorgestellte erste 'selbstreferentielle' rekurrente neuronale Netzwerk repräsentiert seine eigenen Gewichtsänderungsalgorithmen dergestalt, daß sie im Prinzip beliebiger 'Selbst-Manipulation' zugänglich sind. Theoretisch kann das Netzwerk eine Suche nicht nur im Raum möglicher Ein-/Ausgabe-Algorithmen, sondern in der Tat *auch im Raum möglicher Gewichtsänderungsalgorithmen selbst* veranstalten.

Dies wird u.a. dadurch erreicht, daß dem Netzwerk Gelegenheit gegeben wird, seine eigenen Erfolge und Mißerfolge zu beobachten. Weiterhin besitzt das Netzwerk spezielle Eigenschaften, die es ihm erlauben, *alle* eigenen Gewichtsparameter zu analysieren und explizit zu manipiulieren, und zwar einschließlich derjenigen Parameter, die für die Analysier- und Manipulierprozesse zuständig sind. Dank der Allgemeinheit der Architektur existieren keine theoretischen Begrenzungen für die Komplexität der im Netzwerk implementierbaren, zur Selbstmodifikation fähigen Gewichtsänderungsalgo-

[2] Intuitiv formuliert besteht das ultimate Ziel 'selbstreferentiellen' und 'introspektiven' maschinellen Lernens (so wie ich es sehe) darin, einen sogenannten *'Lernkeim'* zu entwerfen. Der *'Keim'* ist ein hypothetischer ursprünglich möglichst einfacher, doch allgemeiner Lernalgorithmus für realistische endliche 'Hardware'. Einer gegebenen Umgebung (welche die Lösung bestimmter spezifischer Probleme fordert) ausgesetzt, soll der Keim lernen, unter 'optimaler' Ausnutzung der limitierten Zeit- und Speicherressourcen sich graduell selbst in *'bootstrap'*-Manier zu verfeinern und sich auf typische Lernprobleme zu spezialisieren. Der hypothetische Keim könnte dieses Ziel durch Sammlung von Informationen über Beziehungen zwischen typischen Problemen und zugehörigen effizienten Lösungsstrategien erreichen. Das vorliegende Kapitel behauptet nicht, das ultimate Ziel zu erreichen. Aus diesem Grunde ist das Wort 'selbstreferentiell' durchwegs zwischen Anführungsstriche gesetzt.

rithmen (abgesehen von unvermeidbaren, durch die Endlichkeit der Architektur bedingten Zeit- und Speicherbegrenzungen). Theoretisch vermag das Netz nicht nur seinen eigenen Gewichtsänderungsalgorithmus zu ändern, sondern auch die Art und Weise, in der es den Gewichtsänderungsalgorithmus ändert, sowie die Art und Weise, in der es den Algorithmus ändert, der den Gewichtsänderungsalgorithmus ändert, und so fort *ad infinitum*. Alle '*Meta*-Ebenen' lassen sich in das ursprüngliche Netzwerk einbetten [115]. Dies führt allerdings *nicht* zu endloser Rekursion.

Abschnitt 8.1 spezifiziert zunächst die Interaktion zwischen Umgebung und Netzwerk, beschreibt daraufhin die Details der 'selbstreferentiellen' Netzwerkarchitektur, gliedert sie in konventionelle und unkonventionelle Aspekte, und macht eine Aussage über die Allgemeinheit der Klasse der auf der Architektur berechenbaren 'selbstreferentiellen' Funktionen. Abschnitt 8.2 leitet schließlich für eine geeignete Zielfunktion mit Hilfe der Kettenregel einen *'vorverdrahteten'* Gewichtsänderungsalgorithmus zur Auffindung 'günstiger' selbst-modifizierender Gewichtsmatrixen ab.

Ein Nachteil dieses vorverdrahteten Lernalgorithmus besteht allerdings in seiner hohen Berechnungskomplexität pro Zeitschritt, welche unabhängig von der Sequenzlänge $O(n_{conn}^2 log n_{conn})$ beträgt, wobei n_{conn} die Anzahl der Verbindungen im Netz bezeichnet. Ein weiterer Nachteil ist die hohe Anzahl lokaler Minima der ungewöhnlich komplexen Zielfunktion. Der Zweck dieses letzten Kapitels der Arbeit besteht jedoch nicht in der Auffindung des effizientesten oder praktikabelsten gradientenbasierten 'selbstreferentiellen' Lernalgorithmus, sondern in der Demonstration, daß solche Algorithmen überhaupt theoretisch möglich sind[3]. Insofern will sich das Kapitel als Ausblick verstanden wissen. Weiterführende Fragen (siehe die Diskussion im abschließenden Abschnitt 8.3) sowie experimentelle Auswertungen sollen zukünftigen Forschungen vorbehalten bleiben.

8.1 EIN 'SELBSTREFERENTIELLES' REKURRENTES NETZ

Die Netzwerkeingabe zum Zeitpunkt t heiße $x(t)$, die entsprechende Netzwerkausgabe sei mit $o(t)$ bezeichnet. Gelegentlich liefert die Umgebung eine

[3]Diese gradientenbasierten 'selbstreferentiellen' Lernalgorithmen stellen einen Spezialfall allgemeinerer 'selbstreferentieller' Lernalgorithmen dar, wie sie in [114] behandelt werden.

Performanzevaluation $eval(t)$. $x(t), o(t)$ und $eval(t)$ seien o.B.d.A. durch reellwertige Vektoren repräsentiert. Die Menge der Algorithmen, die das System ausführen kann, heiße S. Das Ziel besteht in der Auffindung eines Algorithmus in S, welcher zu 'wünschenswerten' Evaluationen $eval(t)$ führt (Abschnitt 8.2 wird den Begriff 'wünschenswert' wie stets durch ein Performanzmaß formal spezifizieren).

Trainingsintervalle. Um Indices zu sparen, betrachten wir ein einzelnes begrenztes Zeitintervall diskreter Zeitschritte, während derer das (später zu beschreibende) Netzwerk mit seiner Umgebung interagiert. Es sei angenommen, daß die vom Netz während dieses Intervalls beobachtete Eingabesequenz über $n_{time} = n_s n_r$ (mit $n_s, n_r \in \mathbf{N}$) Zeitschritte verfügt und aus n_s gleich großen 'Blöcken' der Länge n_r zusammengesetzt ist, während derer sich die Eingabevektoren $x(t)$ nicht ändern. Dies impliziert keinen Verlust an Allgemeinheit, sondern bedeutet lediglich, beliebige Eingabesequenzen durch Beschleunigung aller Netzwerkoperationen um einen konstanten Faktor dergestalt 'aufzublasen', daß jedes Eingabemuster n_r mal hintereinander präsentiert wird, bevor das nächste Eingabemuster beobachtet werden kann. Dies verschafft dem Netzwerk Zeit für sequentielle Informationsverarbeitung.

Ein Trainingsintervall läßt sich u.a. durch Konkatenation mehrerer entsprechender 'konventioneller' Trainingssequenzen für 'konventionelle' rekurrente Netze (Kapitel 2) gewinnen. Letztere Möglichkeit wird dem Netzwerk prinzipiell erlauben, wiederkehrende Regelmäßigkeiten bei der Lösung *verschiedener* Lernaufgaben zu entdecken und sich zunutze zu machen.

Informationsverarbeitende Architektur. Offenbar beschränkt die Architektur des Netzwerkes die Menge S der Algorithmen, die es ausführen kann. Wir wollen uns auf eine endliche, jedoch allgemeine Architektur (auf der Basis der schon aus Kapitel 2 bekannten dynamischen rekurrenten Netze) konzentrieren, die Elementen der Menge S außer unvermeidlichen Zeit- und Speicherbegrenzungen keine zusätzlichen Schranken auferlegt.

Wie bereits einführend erwähnt, unterscheidet sich die Methode des vorliegenden Kapitels von Verfahren vorangehender Kapitel sowie von Ansätzen anderer Autoren dadurch, daß das Netzwerk u.a. auch $eval(t)$ als Eingabe verarbeiten kann sowie theoretisch in der Lage ist, alle eigenen Gewichtsparameter ohne prinzipielle Einschränkung explizit zu analysieren und zu ändern[4]. Dies wird erreicht durch (1) Einführung spezieller Eingabekno-

[4]Bei den mir bekannten älteren Ansätzen zum maschinellen Lernen lassen sich die adaptiven Komponenten der Algorithmen ausschließlich durch *vorverdrahtete* Lernal-

ten zur Beobachtung externer Performanzevaluationen, (2) Einführung von Adressen für alle Netzwerkverbindungen, (3) Einführung spezieller Ausgabeknoten zur sequentiellen Adressierung *aller* Verbindungen (einschließlich derjenigen Verbindungen, die gegenwärtig zur Verbindungsadressierung verwendet werden), (4) Einführung spezieller Eingabeknoten zur Analyse durch spezielle Ausgabeknoten adressierter Gewichte, (5) Einführung spezieller Ausgabeknoten zur Modifikation von durch spezielle Ausgabeknoten adressierten Gewichten. Die Architektur erlaubt die Implementierung von Algorithmen aus S, deren Ausgaben u.a. im Prinzip beliebige Änderungen von Komponenten der Algorithmen aus S sein können, so daß die Ergebnisse der Änderungen wiederum Algorithmen aus S darstellen.

Abschnitt 8.1.1 beschreibt die konventionellen Aspekte der Netzwerkarchitektur. Abschnitt 8.1.2 beschreibt die neuartigen 'selbstreferentiellen' Aspekte des Netzes. Der Leser mag gelegentlich auf die kompakte Zusammenfassung relevanter Notation in Tabelle 1 sowie auf die illustrative Abbildung 1 zurückgreifen.

8.1.1 'KONVENTIONELLE' ASPEKTE DER ARCHITEKTUR

$o(t)$ wird aus $x(\tau), \tau < t$ durch ein rekurrentes Netz mit $n_I > n_x$ Eingabeknoten und n_y Nichteingabeknoten berechnet. Eine Teilmenge der Nichteingabeknoten wird als die Menge der 'normalen' Ausgabeknoten (mit Kardinalität $n_o < n_y$) bezeichnet.

Der notationellen Bequemlichkeit halber werde ich im folgenden verschiedene Bezeichner für die reellwertige Aktivation ein und desselben Knotens vergeben.

z_k steht für den k-ten Knoten im Netz. y_k repräsentiert den k-ten Nichteingabeknoten. x_k stellt den k-ten 'normalen' Eingabeknoten dar. o_k bezeichnet den k-ten 'normalen' Ausgabeknoten. Steht u für einen Knoten,

gorithmen ändern. Lenats komplexes 'symbolisches' EURISKO-System stellt möglicherweise eine Ausnahme dieser Regel dar (Lenat berichtet, daß sein System Heuristiken zur Auffindung von Heuristiken fand [48]). EURISKOs Mechanismus wurde allerdings häufig wegen seiner Undurchsichtigkeit kritisiert, sowie dafür, daß er gelegentliche in ihrer Bedeutung schwer einzuschätzende Eingriffe seitens des Programmierers notwendig machte. Das viel einfachere 'subsymbolische' System des vorliegenden Kapitels ist dagegen höchst transparent, verfügt über ein klar definiertes Performanzmaß, und erfordert keinerlei Interventionen eines externen Programmierers.

so denotiert $u(t)$ seine Aktivation zum Zeitpunkt t. Stellt $v(t)$ einen Vektor dar, so bezeichnet $v_k(t)$ seine k-te Komponente (dies steht nicht im Widerspruch zum vorangegangenen Satz). Einer der 'normalen' Eingabeknoten heißt $bias$. Es gilt $bias(t) = 1$ für alle t. Der Zweck der verbleibenden Eingabeknoten wird sich im Abschnitt 8.1.2 klären, welcher die 'selbstreferentiellen' Aspekte der Architektur beschreibt.

Jeder Eingabeknoten ist Quelle gerichteter Verbindungen zu allen Nichteingabeknoten. Jeder Nichteingabeknoten weist ebenfalls gerichtete Verbindungen zu allen Nichteingabeknoten auf. Offensichtlich gibt es $(n_I + n_y)n_y = n_{conn}$ Verbindungen im Netz. Die Verbindung vom Knoten j zum Knoten i trägt den Namen w_{ij}. Einer der Namen der Verbindung vom j-ten 'normalen' Eingabeknoten zum k-ten 'normalen' Ausgabknoten ist demzufolge $w_{o_k x_j}$. $w_{ij}(t)$s rellwertiges Gewicht zum Zeitpunkt t wird mit $w_{ij}(t)$ bezeichnet. Vor Beginn der Trainingsphase werden alle Anfangsgewichte $w_{ij}(1)$ zufällig initialisiert.

Die folgenden Definitionen werden den Leser an Kapitel 2 erinnern: Für 'normale' Eingabeknoten x_k ist $x_k(t)$ als die k-te Komponente des Eingabevektors $x(t)$ definiert. Die Spezifizierung der Aktivationen der verbleibenden Eingabeknoten bleibt Abschnitt 8.1.2 vorbehalten. Für Nichteingabeknoten y_k definieren wir

$$net_{y_k}(1) = 0, \;\; \forall t \geq 1 : y_k(t) = f_{y_k}(net_{y_k}(t)), \;\; \forall t > 1 : \;\; net_{y_k}(t) = \sum_l w_{y_k l}(t-1) l(t-1), \tag{8.1}$$

Der gegenwärtige Algorithmus des Netzes ist durch seine augenblickliche Gewichtsmatrix (sowie die gegenwärtigen Aktivationen) gegeben. Man beachte jedoch, daß die Berechnung der $w_{ij}(t)$ noch nicht spezifiziert wurde. In früher besprochenen Systemen wurden Gewichtsänderungen mindestens eines beteiligten Netzes ausschließlich durch einen festgeschriebenen Lernalgorithmus verursacht. Nicht so jedoch im vorliegenden Fall, wie der nächste Abschnitt zeigen wird.

8.1.2 'SELBSTREFERENTIELLE' ASPEKTE DER ARCHITEKTUR

Dieser Abschnitt beschreibt bisher unspezifizierte Interpretationen gewisser Ein- und Ausgaben, welche unser Netzwerk zum ersten 'selbstreferentiellen'

Netzwerk mit expliziter Kontrolle über *alle* sein Verhalten steuernde Parameter machen. Da wir nicht alle potentiell nützlichen Arten der Selbstmodifikation kennen können, und da geeignete Selbstmodifikationsprozeduren beliebige Komplexität aufweisen könnten, werde ich die 'selbstreferentiellen' Aspekte dergestalt definieren, daß die Selbstmodifikationsprozeduren die Form beliebiger berechenbarer Abbildungen von *Algorithmuskomponenten* und *Performanzevaluationen* auf *Algorithmusmodifikationen* annehmen können (modulo Zeit- und Speicherbegrenzungen).

Im folgenden werde ich vier unkonventionelle Aspekte auflisten. Die resultierende Architektur sollte als *Repräsentant* einer Vielzahl ähnlicher Architekturen verstanden werden.

1. Das Netz 'sieht' Performanzevaluationen. Eine Teilmenge der Eingabeknoten, die keine 'normalen' Eingabeknoten enthält, wird als Menge der *Evalknoten* (mit Kardinalität n_{eval}) bezeichnet. $eval_k$ steht für den k-ten Evalknoten. Obwohl diese Modifikation konventioneller Netze im Vergleich mit den folgenenden Modifikationen relativ einfach ist, repräsentiert sie doch den möglicherweise bedeutsamsten Beitrag zur Erzielung 'selbstreferentieller' Architekturen, wie Fußnote[8], Abschnitt 8.1.3 noch etwas detaillierter ausführen wird.

2. Jede adaptive Komponente des Netzes erhält eine Adresse. Für jede Verbindung w_{ij} führen wir eine Adresse $adr(w_{ij})$ ein. Dies wird dem Netz helfen, über seine eigenen Verbindungen zu 'reden', wie die nächsten beiden Punkte klarmachen werden. O.B.d.A. nehmen wir im folgenden an, daß $adr(w_{ij})$ als Binärvektor repräsentiert sei (dies stellt jedoch nur eine von vielen Möglichkeiten dar[5]).

3. Das Netz vermag all seine eigenen Gewichte zu analysieren. Eine Untermenge der Nichteingabeknoten, welche keine 'normalen' Ausgabeknoten enthält, wird als *Analyseknoten* bezeichnet. ana_k bezeichnet den k-ten von n_{ana} Analyseknoten. Die Analyseknoten dienen dazu, sequentiell Verbindungen anzusprechen, deren gegenwärtige Gewichte das Netzwerk 'in Erfahrung bringen möchte'. Es bereitet keine Schwierigkeiten, die Analyseknoten mit genügend Kapazität zur Adressierung jeder beliebigen Verbindung auszustatten, *einschließlich jener Verbindungen, die zu Analyseknoten führen.* Dies läßt sich beispielsweise durch

$$n_{ana} = up(log_2 n_{conn}) \tag{8.2}$$

erreichen, wobei $up(x)$ die kleinste ganze Zahl $\geq x$ liefert. Ein spezieller Eingabeknoten, der weder 'normaler' Eingabeknoten noch Evalknoten ist,

wird mit *val* bezeichnet. $val(t)$ berechnet sich gemäß

$$val(1) = 0, \quad \forall t \geq 1 : \; val(t+1) = \sum_{i,j} g[\|ana(t) - adr(w_{ij})\|^2] w_{ij}(t), \quad (8.3)$$

wobei g eine differenzierbare Funktion mit Wertebereich $[0 \ldots 1]$ ist. g bestimmt, wie nahe eine Verbindungsadresse[5] den Aktivationen der Analyseknoten sein muß, um einen signifikanten Beitrag für $val(t)$ zu ermöglichen. Ich schlage eine Funktion g vor, die nahezu überall fast Null ist, um den Ursprung herum jedoch eine enge Spitze aufweist. Dies wird dem Netz im Prinzip erlauben, sich zu jedem Zeitpunkt eine einzelne Verbindung herauszupicken und ihr gegenwärtiges Gewicht zu analysieren, ohne *'cross-talk'* von anderen Gewichten in Kauf nehmen zu müssen.

Es ist ohne weiteres möglich, alternative Schemata zur gleichzeitigen Adressierung mehr als eines Gewichtes zu entwerfen (siehe auch Fußnote[5]).

4. Das Netz vermag all seine eigenen Gewichte zu modifizieren. Eine Untermenge der Ausgabeknoten, welche weder 'normale' Ausgabeknoten noch Analyseknoten enthält, wird als die Menge der *Modifizierknoten* bezeichnet. mod_k bezeichnet den k-ten von n_{mod} Modifizierknoten. Die Modifizierknoten dienen dazu, sequentiell Verbindungen anzusprechen, deren gegenwärtige Gewichte das Netzwerk 'ändern möchte'. Es bereitet von neuem keine Schwierigkeiten, die Modifizierknoten mit genügend Kapazität zur Adressierung jeder beliebigen Verbindung auszustatten, *einschließlich jener Verbindungen, die zu Modifizierknoten führen.* Dies läßt sich wiederum durch

$$n_{mod} = up(log_2 n_{conn}) \quad (8.4)$$

erreichen. Ein spezieller Nichteingabeknoten, der weder 'normaler' Ausgabeknoten, Analyseknoten, noch Modifizierknoten ist, wird mit $\triangle$ bezeichnet. $f_{\triangle}$ sollte sowohl positive als auch negative Aktivationen $\triangle(t)$ gestatten.

[5] Man beachte die *Notwendigkeit* kompakter Adressierschemata. Man könnte ja in Anlehnung an Architektur 1 aus Kapitel 3 auf den alternativen Gedanken kommen, einen Analyseknoten pro Verbindung einzuführen, doch offensichtlich kann das nicht funktionieren: Am Ende hätten wir immer mehr Gewichte als Analyseknoten, und damit mehr Gewichte, als adressiert werden könnten. 'Selbstreferenz' wäre auf diese Weise nicht möglich. Es sollte jedoch angemerkt werden, daß das binäre Adressierungsschema bei weitem nicht das kompakteste Schema darstellt. Im Prinzip erlauben reelle Zahlen ja die Darstellung unbeschränkter Informationsmengen durch eine einzige Zahl. Daher ist es beispielsweise möglich, beliebige simultane Gewichtsänderungen für alle Gewichte durch die möglichen reellwertigen Aktivationen eines einzelnen Knotens zu repräsentieren. Unbegrenzte Präzision ist jedoch unrealistisch für praktische Anwendungen. Und der Zweck dieses Kapitels ist ja nicht das Auffinden der kompaktesten 'selbstreferentiellen' Adressierungsmethode, sondern lediglich wenigstens *einer* solchen Methode – welche als Repräsentant vieler vergleichbarer Ansätze gesehen werden sollte.

$mod(t)$ und $\triangle(t)$ arbeiten zusammen, um explizite Gewichtsänderungen gemäß

$$w_{ij}(t+1) = w_{ij}(t) + \triangle(t)\ g[\ \|adr(w_{ij}) - mod(t)\|^2\ ] \tag{8.5}$$

hervorzurufen. Ist g nahezu überall fast Null, weist aber um den Ursprung herum eine enge Spitze auf, so kann sich das Netz zu jedem Zeitpunkt eine einzelne Verbindung herauspicken und ihr gegenwärtiges Gewicht ändern, ohne andere Gewichte zu beeinflussen. Es ist wiederum ohne weiteres möglich, alternative Schemata zur gleichzeitigen Modifikation mehr als eines Gewichtes zu entwerfen (siehe erneut Fußnote[5]).

Die Gleichungen (8.1), (8.3), und (8.5) beschreiben im wesentlichen die vorverdrahtete Systemdynamik – die vom Performanzmaß abhängigen Belegungen der Evalknoten werden allerdings erst im Abschnitt 8.2 spezifiziert werden.

Es gibt viele im obigen Sinne 'selbstreferentielle' Netze. Der Begriff *'versteckte Knoten'* bezeichne Knoten, die weder 'normale' Eingabeknoten, 'normale' Ausgabeknoten, Evalknoten, Analysierknoten, Modifizierknoten, *val* oder $\triangle$ sind. Es existieren $n_h = n_y - n_o$ derartige versteckte Knoten. Es gibt eine unendliche Anzahl von Möglichkeiten, die Bedingung

$$n_{conn} \geq (n_h+n_o+2up(log_2 n_{conn})+1)(n_h+n_o+2up(log_2 n_{conn})+1+n_x+1+n_{eval}). \tag{8.6}$$

zu erfüllen. Ein Beispiel, bei dem das Gleichheitszeichen in (8.6) erfüllt ist, ist durch

$$n_x = 27, n_o = n_{eval} = 4, n_{ana} = n_{mod} = 11, n_h = 5$$

gegeben.

8.1.3 MÄCHTIGKEIT DER KLASSE DER VOM NETZ BERECHENBAREN FUNKTIONEN

Mit geeigneten konstanten (zeit-invarianten) $w_{ij}(t)$, einfachen konventionellen Aktivationsfunktionen f_k, genügend n_h 'versteckten' Knoten und ausreichender Blockgröße n_r kann das 'selbstreferentielle' Netz durch wiederholte Anwendung von (8.1) jede beliebige Funktion

$$f : \{0,1\}^{n_x+1+n_{eval}+n_o+n_{ana}+n_{mod}+1} \rightarrow \{0,1\}^{n_o+n_{ana}+n_{mod}+1} \tag{8.7}$$

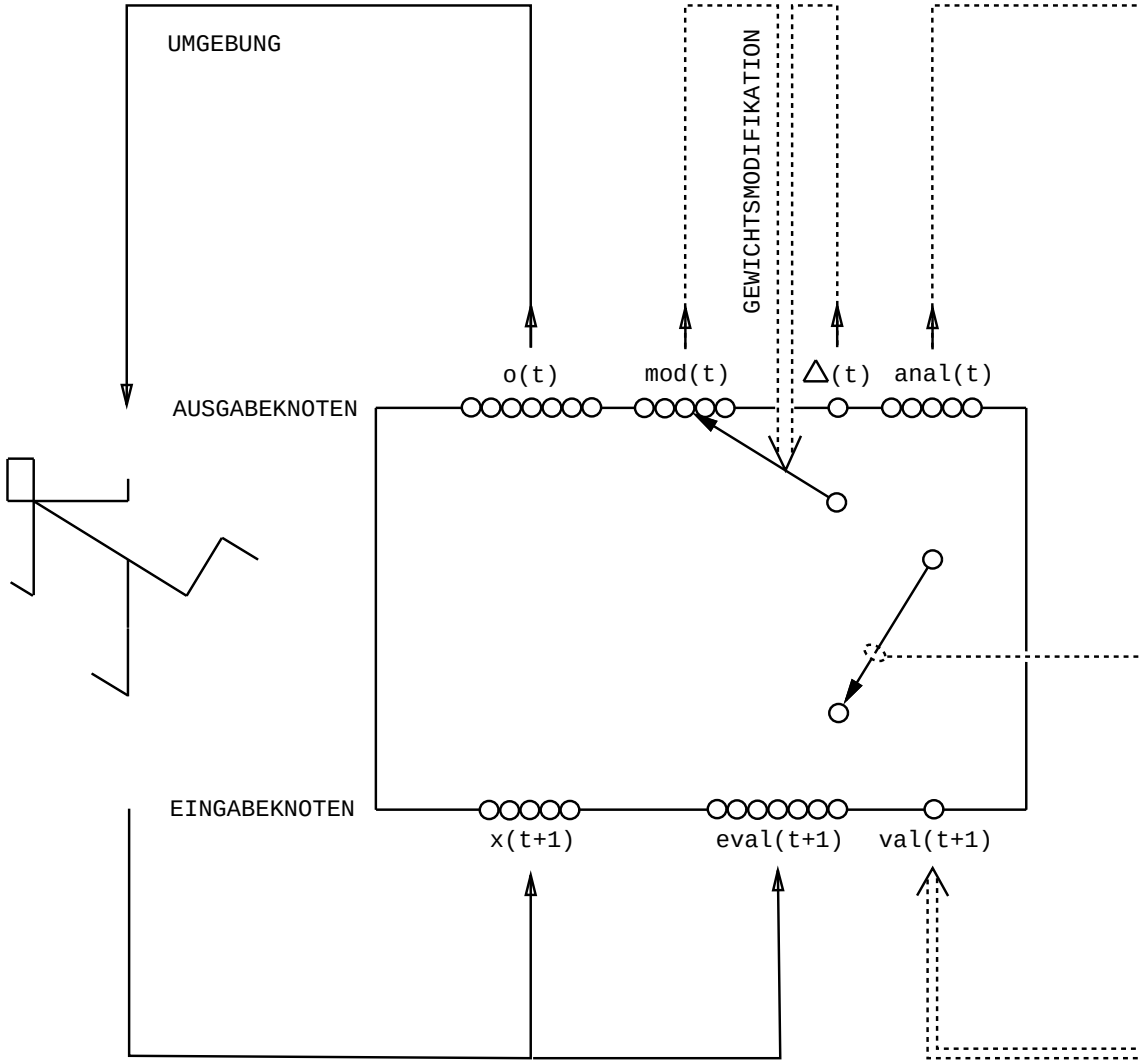

Abbildung 8.1: Der spezielle Eingabevektor $eval(t)$ informiert das vollständig rückgekoppelte Netzwerk (nur zwei Verbindungen sind eingezeichnet) über externe Performanzevaluationen. Der spezielle Ausgabevektor $ana(t)$ adressiert eine zu analysierende Verbindung, deren Gewicht in den speziellen Eingabeknoten val geschrieben wird. Der spezielle Ausgabevektor $mod(t)$ adressiert eine zu modifizierende Verbindung, deren Gewicht in Proportion zur gegenwärtigen Aktivation des speziellen Ausgabeknotens $\triangle$ geändert wird. Siehe Text für Details.

Symbol	Beschreibung
t	Diskreter Zeitindex aus $\{1, \ldots, n_{time}\}$
x_k	k-ter 'normaler' Eingabeknoten
y_k	k-ter Nichteingabeknoten
o_k	k-ter 'normaler' Ausgabeknoten
z_k	k-ter Knoten
$eval_k$	k-ter Evalknoten (zur Beobachtung von Performanzevaluationen)
ana_k	k-ter Analyseknoten (um Verbindungen zu adressieren)
mod_k	k-ter Modifizierknoten (um Verbindungen zu adressieren)
val	spezieller Eingabeknoten zur Analyse von Gewichten
$\triangle$	spezieller Ausgabeknoten, um Gewichte zu ändern
$u(t)$	Aktivation von u zur Zeit t, falls u Knoten bezeichnet
$v_k(t)$	k-te Komponente von $v(t)$, falls $v(t)$ Vektor (konsistent mit vorangegangener Zeile)
$x(t)$	'normaler' Eingabevektor zum Zeitpunkt t
$o(t)$	'normaler' Ausgabevektor zum Zeitpunkt t
$eval(t)$	Performanzevaluationsvektor zum Zeitpunkt t
$ana(t)$	spezieller Ausgabevektor zur Adressierung von Verbindungen
$mod(t)$	spezieller Ausgabevektor zur Adressierung von Verbindungen
n_x	$dim(x(t))$, Anzahl der normalen Eingabeknoten
n_o	$dim(o(t))$, Anzahl der 'normalen' Ausgabeknoten
n_{eval}	$dim(eval(t))$, Anzahl der Evalknoten
n_I	$n_x + n_{eval} + 1$, Anzahl *aller* Eingabeknoten
$up(x)$	kleinste ganze Zahl $\geq x$
n_{mod}	$dim(mod(t)) = up(log_2 n_{conn})$, Anzahl der Modifizierknoten
n_{ana}	$dim(ana(t)) = up(log_2 n_{conn})$, Anzahl der Analysierknoten
n_y	$n_o + n_{ana} + n_{mod} + 1$, Anzahl der Nichteingabeknoten
n_z	$n_I + n_y$, Anzahl aller Knoten
n_h	Anzahl der 'versteckten' Knoten
w_{ij}	Verbindung vom Knoten j zum Knoten i
n_{conn}	$n_I(n_y + n_I)$, Anzahl aller Verbindungen
$adr(w_{ij})$	Adresse von w_{ij}
g	Funktion zur Definition der 'Nähe' von Adressen, mit enger Spitze um den Ursprung herum
$val(t+1)$	$\sum_{i,j} g[\|\|ana(t) - adr(w_{ij})\|\|^2] w_{ij}(t)$
$w_{ij}(t)$	Gewicht von w_{ij} zum Zeitpunkt t
$w_{ij}(t+1)$	$w_{ij}(t) + \triangle(t)\ g[\ \|\|adr(w_{ij}) - mod(t)\|\|^2\ ]$
n_r	konstante Blockgröße, $x(t)$ bleibt während Blöcken der Länge n_r invariant
n_s	Anzahl sukzessiver Blöcke in der Eingabesequenz
$n_{time} = n_r n_s$	Anzahl sukzessiver Zeitschritte in der Eingabesequenz
S	Menge der vom Netz ausführbaren Algorithmen

Tabelle 8.1: *Symboldefinitionen zur Beschreibung des 'selbstreferentiellen' Netzes.*

berechnen, die sich bei konstanter Maschinenzyklenanzahl n_{cyc} durch einen konventionellen digitalen (sequentiellen oder parallelen) Rechner mit begrenzten Zeit- und Speicherressourcen[6] berechnen läßt. Dies liegt daran, daß Informationsverarbeitung in konventionellen Rechnern durch wiederholte Anwendung einfacher Boolescher Funktionen ausdrückbar ist, welche sich ohne weiteres in rekurrenten Netzen implementieren lassen.

Wenigstens der $\triangle$ Ausgabeknoten und der val Eingabeknoten sollten nicht nur binäre, sondern reelle Werte annehmen können. Es bereitet jedoch keine größeren Schwierigkeiten, zu zeigen, daß die Bereiche $\{0,1\}$ in (8.7) für beliebige Knoten durch $\mathbf{R}$ ersetzt werden dürfen.

Dies erlaubt uns, ohne Rücksicht auf zusätzliche 'hardware'-spezifische Beschränkungen der Mächtigkeit des Netzes die Speicherbegrenzung n_h und die Zeitbegrenzung n_r durch zwei natürliche Parameter zu identifizieren[7].

Damit erhalten wir ein allgemeines 'selbstreferentielles' Netz mit im wesentlichen unbeschränkter Mächtigkeit (modulo unvermeidlicher Zeit- und Speicherbegrenzungen), welches im Prinzip seine eigene Gewichtsmatrix analysieren und ändern kann, einschließlich jener Teile der Gewichtsmatrix, die für die Analyse und Änderung der Gewichtsmatrix zuständig sind. Der Analyse- und Änderungsprozeß selbst kann beliebig komplex sein – es gibt keine wesentlichen theoretischen Begrenzungen für die Raffinesse der im Netz implementierbaren Gewichtsänderungsalgorithmen. Das gleiche gilt für die Gewichtsänderungsalgorithmen, die die Gewichtsänderungsalgorithmen ändern, und die Gewichtsänderungsalgorithmen, die die Gewichtsänderungsalgorithmen, die die Gewichtsänderungsalgorithmen ändern, ändern, etc.... Dies liegt im wesentlichen daran, daß jeder '*Meta-Level*' in dasselbe Netz eingebettet ist[8].

[6] Oder durch eine gegebene, für begrenzte Zeit (während der sie nur einen endlichen Teil ihres Bandes benutzen kann) operierende Turingmaschine – oder durch einen gegebenen endlichen Automaten.

[7] Es sollte erwähnt werden, daß n_{ana} und n_{mod} die Tendenz haben, mit n_h zu wachsen (falls beide nicht von vornherein groß genug gewählt worden sind), da n_{conn} mit n_h zunimmt. n_{ana} und n_{mod} wachsen für große Netzwerke jedoch viel langsamer als n_h.

[8] *Einfachere Alternative ohne Fähigkeit zu expliziter Gewichtsänderung.* Da rekurrente Netze Allzweckmaschinen mit im wesentlichen unbeschränkter Mächtigkeit sind, erlauben sie die Implementierung beliebiger Algorithmen, einschließlich *durch Aktivationen statt durch Gewichte repräsentierter Lern*algorithmen. Daher können wir im Prinzip ein einfacheres 'selbstreferentielles' System als dasjenige, auf welches sich vorliegendes Kapitel konzentriert, mit einem 'konventionellen' rekurrenten Netz *ohne* explizite Fähigkeit zur Eigengewichtsmodifikation (mittels $ana(t), val(t), mod(t), \triangle(t)$) realisieren. (So gesehen brauchen wir nur die erste der in 8.1.2 aufgelisteten Modifikationen konventioneller rekurrenter Netze zu behalten, nämlich die Evalknoten.) Dieses alternative Netz ist allerdings nicht imstande, all seine adaptiven Parameter selbst zu kontrollieren.

8.2 ALGORITHMUS FÜR ÜBERWACHTES LERNEN

Um nützliche 'selbst-modifizierende' Gewichtsänderungsalgorithmen zu finden, bedarf es eines *vorverdrahteten Gewichtsänderungsalgorithmus.* Wir müssen sicherstellen, daß gewisse Aspekte des Gewichtsänderungsalgorithmus unveränderbar bleiben. Wir können uns nicht erlauben, *alles* adaptiv zu machen: So muß man beispielsweise von Beginn an gewährleisten, daß zu 'wünschenswerten' Evaluationen führende Gewichtsmatrizen den Vorzug vor anderen Gewichtsmatrizen erhalten. Dem System darf nicht erlaubt werden, diese zentrale Spielregel zu ändern. Die *vorverdrahteten* Aspekte des Lernalgorithmus müssen 'introspektive' Gewichtsänderungsalgorithmen begünstigen, die sich selbst in 'wünschenswerter' Weise manipulieren. Der Begriff 'wünschenswert' wird im folgenden wie stets durch eine geeignete Zielfunktion spezifiziert werden.

[114] beschreibt u.a. einen nicht gradienten-basierten *vorverdrahteten Reinforcement-Lernalgorithmus.* Um den Rahmen dieser Schrift nicht zu sprengen, will ich mich hier jedoch auf durch Anwendung der Kettenregel herleitbare Lernalgorithmen beschränken. Als exemplarisches Beispiel soll der folgende *vorverdrahtete Algorithmus für 'selbstreferentielles' überwachtes Lernen* dienen.

Bei nicht quantisierten Variablen wird im folgenden angenommen, daß sie über ihren gesamten Belegungsbereich rangieren. Für $o_k(t)$ mag zum Zeitpunkt t ein Zielwert $d_k(t)$ vorgegeben sein. Obwohl es keinerlei Bedeutung für die formale Herleitung des Lernalgorithmus hat, sei angenommen, daß Zielwerte $d_k(t)$ nur am Ende eines der n_s Blöcke mit n_r Zeitschritten auftreten können (um die temporalen Ressourcen des Netzes nicht unnötig zu beschränken). Wir setzen $n_{eval} = n_o$. Es gibt also ebensoviele Evalknoten wie 'normale' Ausgabeknoten. Es sei angenommen, daß Eingaben und Zielwerte nicht von früheren Ausgaben (via Rückkopplung durch die Umgebung) abhängen – siehe [114] für einen Lernalgorithmus für die allgemeinere Situation des *'Reinforcement'*-Lernens.

Zusammenfassend schreiben wir zunächst in kompakter Form die Systemdynamik nieder:

$$net_{y_k}(1) = 0, \quad \forall t \geq 1: \quad x_k(t) \leftarrow Umgebung, \quad y_k(t) = f_{y_k}(net_{y_k}(t)),$$
$$\forall t > 1: \quad net_{y_k}(t) = \sum_l w_{y_k l}(t-1) l(t-1), \tag{8.8}$$

$$\forall t \geq 1: \quad w_{ij}(t+1) = w_{ij}(t) + \triangle(t)\; g[\; \|adr(w_{ij}) - mod(t)\|^2\;], \quad (8.9)$$

$$val(1) = 0, \;\; \forall t \geq 1: \; val(t+1) = \sum_{i,j} g[\|ana(t) - adr(w_{ij})\|^2] w_{ij}(t), \; (8.10)$$

Der folgende Aspekt der Systemdynamik ist für überwachtes Lernen spezifisch und wurde daher in den vorangehenden Abschnitten noch nicht definiert. Wie stets beim überwachten Lernen vermitteln die $eval(t)$ Information über erwünschte Ausgaben zu bestimmten Zeitpunkten:

$$eval_k(1) = 0, \;\; \forall t \geq 1 : eval_k(t+1) = d_k(t) - o_k(t), \; falls\, d_k(t)\, existiert, \; und\, 0\, sonst. \quad (8.11)$$

8.2.1 PERFORMANZMASS

Wie stets beim überwachten Lernen wollen wir $E^{total}(n_r n_s)$ minimieren, wobei

$$E^{total}(t) = \sum_{\tau=1}^{t} E(\tau), \quad mit\; E(t) = \frac{1}{2} \sum_k (eval_k(t+1))^2.$$

Der folgende Algorithmus zur Minimierung von E^{total} ist teilweise inspiriert durch die 'konventionellen' Lernalgorithmen für rekurrente Netze aus Kapitel 2 sowie durch den Algorithmus für das die Gewichte eines zweiten Netzes manipulierende *nicht* 'selbstreferentielle' Netz aus Kapitel 3. Seine Komplexität übertrifft allerdings die Komplexität der bisher behandelten Algorithmen.

8.2.2 HERLEITUNG DES ALGORITHMUS

Mit der Kettenregel berechnen wir Gewichtsinkremente für die *Initial*gewichte $w_{ab}(1)$ gemäß

$$w_{ab}(1) \leftarrow w_{ab}(1) - \eta \frac{\partial E^{total}(n_r n_s)}{\partial w_{ab}(1)}, \quad (8.12)$$

wobei η eine konstante positive Lernrate bezeichnet. Damit erhalten wir einen *exakten* gradientenbasierten Algorithmus zur Minimierung von E^{total} unter der 'selbstreferentiellen' durch (8.8) bis (8.11) gegebenen Dynamik. Um den Schreibaufwand zu reduzieren, seien folgende (teilweise durch [148] inspirierte) Kurzschreibweisen eingeführt:

Für alle Knoten u und alle Gewichte w_{ab} schreiben wir

$$p^u_{ab}(t) = \frac{\partial u(t)}{\partial w_{ab}(1)}. \tag{8.13}$$

Für alle Verbindungspaare (w_{ij}, w_{ab}) schreiben wir

$$q^{ij}_{ab}(t) = \frac{\partial w_{ij}(t)}{\partial w_{ab}(1)}. \tag{8.14}$$

Man beachte zunächst, daß

$$\frac{\partial E^{total}(1)}{\partial w_{ab}(1)} = 0, \ \forall t > 1: \ \frac{\partial E^{total}(t)}{\partial w_{ab}(1)} = \frac{\partial E^{total}(t-1)}{\partial w_{ab}(1)} - \sum_k eval_k(t+1) p^{o_k}_{ab}(t). \tag{8.15}$$

Das verbleibende Problem besteht also in der Berechnung der $p^{o_k}_{ab}(t)$, welche sich durch inkrementelle Berechnung aller $p^{z_k}_{ab}(t)$ und $q^{ij}_{ab}(t)$ vollziehen läßt, wie wir gleich sehen werden.

Für den Zeitpunkt 1 gilt

$$p^{z_k}_{ab}(1) = 0. \tag{8.16}$$

Für $t \geq 1$ erhalten wir die Rekursion

$$p^{x_k}_{ab}(t+1) = 0, \tag{8.17}$$

$$p^{eval_k}_{ab}(t+1) = -p^{o_k}_{ab}(t), \ falls \ d_k(t) \ existiert, \ und \ 0 \ sonst, \tag{8.18}$$

$$p^{val}_{ab}(t+1) = \sum_{i,j} \frac{\partial}{\partial w_{ab}(1)} [\ g(\|ana(t) - adr(w_{ij})\|^2) w_{ij}(t)\ ] =$$

$$\sum_{i,j} \{\ \ q^{ij}_{ab}(t) g[\|ana(t) - adr(w_{ij})\|^2)] +$$

$$w_{ij}(t)\ [\ \ g'(\|ana(t) - adr(w_{ij})\|^2) 2 \sum_m (ana_m(t) - adr_m(w_{ij})) p^{ana_m}_{ab}(t)\ \ ]\ \ \} \tag{8.19}$$

(wobei $adr_m(w_{ij})$ das m-te Bit der Adresse von w_{ij} bezeichnet),

$$p^{y_k}_{ab}(t+1) = f'_{y_k}(net_{y_k}(t+1)) \sum_l \frac{\partial}{\partial w_{ab}(1)} [l(t) w_{y_k l}(t)] =$$

$$f'_{y_k}(net_{y_k}(t+1)) \sum_l w_{y_k l}(t) p^l_{ab}(t) + l(t) q^{y_k l}_{ab}(t), \tag{8.20}$$

wobei

$$q^{ij}_{ab}(1) = 1 \; falls \; w_{ab} = w_{ij}, \; und \; 0 \; sonst, \tag{8.21}$$

$$\forall t > 1 : \;\; q^{ij}_{ab}(t) = \frac{\partial}{\partial w_{ab}(1)} \left[w_{ij}(1) + \sum_{\tau < t} \triangle(\tau) g(\|mod(\tau) - adr(w_{ij})\|^2) \right] =$$

$$q^{ij}_{ab}(t-1) + \frac{\partial}{\partial w_{ab}(1)} \triangle(t-1) g(\|mod(t-1) - adr(w_{ij})\|^2) =$$

$$q^{ij}_{ab}(t-1) + p^{\triangle}_{ab}(t-1) g(\|mod(t-1) - adr(w_{ij})\|^2) +$$

$$2\triangle(t-1)\, g'(\|mod(t-1) - adr(w_{ij})\|^2) \sum_m [mod_m(t-1) - adr_m(w_{ij})] p^{mod_m}_{ab}(t-1). \tag{8.22}$$

Die $p^j_{ab}(t)$ und $q^{ij}_{ab}(t)$ lassen sich gemäß den Gleichungen (8.16)-(8.22) inkrementell zu jedem Zeitschritt aktualisieren, was bedeutet, daß auch (8.15) und (8.12) inkrementell berechnet werden können. Die Speicherkomplexität beträgt unabhängig von der Sequenzlänge $O(n^2_{conn})$. Der Berechnungsaufwand pro Zeitschritt beträgt unabhängig von der Sequenzlänge $O(n^2_{conn} log n_{conn})$. Dies übertrifft bedauerlicherweise sogar die Berechnungskomplexität von RTRL (siehe Kapitel 2), welche gleich $O(n^2_{conn})$ ist.

Ein weiterer Nachteil (neben der hohen Komplexität) ist die hohe Anzahl lokaler Minima der ungewöhnlich komplexen Zielfunktion. Der Zweck dieses letzten Kapitels der Arbeit besteht jedoch nicht in der Auffindung des effizientesten oder praktikabelsten gradientenbasierten 'selbstreferentiellen' Lernalgorithmus, sondern in der Demonstration, daß solche Algorithmen überhaupt theoretisch möglich sind.

Offenbar gerät der Algorithmus nicht in eine endlose Rekursion. Er verwendet Gradientenabstieg, um Gewichtsänderungsalgorithmen zu finden, die zwar nicht notwendigerweise Gradientenabstieg betreiben (sondern möglicherweise etwas Raffinierteres), aber dennoch dazu beitragen, E^{total} zu minimieren. Damit läßt sich das Gesamtsystem als eine 'selbstreferentielle' *Erweiterung* der 'konventionelleren' Algorithmen aus Kapitel 2 ansehen. Beschleunigungsverfahren *à la* Kapitel 2 sind möglich, für die Zwecke des vorliegenden Kapitels jedoch nicht von Belang.

8.3 ABSCHLIESSENDE BEMERKUNGEN ZU KAPITEL 8

In diesem Kapitel habe ich im wesentlichen ein gradientenbasiertes Netzwerk konstruiert, welches nicht nur lernen kann, von der Umgebung gestellte Probleme zu lösen, sondern auch, seine eigenen Gewichte als Eingabedaten zu verwenden und seinen eigenen Gewichtsänderungsalgorithmus in sinnvoller Weise beliebig zu manipulieren. Der Effekt dieses Vorgehens besteht im wesentlichen im Kollaps einer Kette von *'Meta*-Netzwerken' und *'Meta-Meta-*...-Netzwerken' in das Originalnetzwerk selbst [115].

Das System will sich als Repräsentant einer Vielzahl ähnlicher Systeme verstanden wissen, denen zwei Dinge gemein sind: Expliziter Zugang zu allen modifizierbaren Algorithmuskomponenten sowie prinzipiell unbeschränkte Mächtigkeit (in praktischen Anwendungen allerdings limitiert durch unvermeidliche vorgegebene Zeit- und Speicherbegrenzungen).

Biologische Plausibilität. Es scheint, als ob ich meiner 10^{15}-ten Synapse nicht explizit vorschreiben kann, welchen Wert sie annehmen soll. Statt dessen scheinen sich meine eigenen Synapsen nur indirekt durch Willensakte beeinflussen zu lassen – beispielsweise durch Kreierung von 'Schlüsseln' und 'Einträgen', die miteinander assoziiert werden sollen (wie z.B. korrespondierende Wörter aus zwei verschiedenen Sprachen). Auf den ersten Blick scheint das neuronale Netz in meinem Kopf demzufolge eine andere Art von 'Selbstreferenz' zu pflegen als das KNN aus Sektion 8.2. Mir ist allerdings keine wirkliche Evidenz für diese Vermutung bekannt: Auch unser 'selbstreferentielles' KNN verfügt ursprünglich über kein Konzept seines n-ten Gewichtes – alles, was es tun kann, ist, herauszufinden, 'in der Sprache der Aktivationen über Gewichte zu reden', ohne wirklich zu 'wissen', was ein Gewicht ist (auch Menschen besaßen für lange Zeit keinerlei Vorstellung von ihren Synapsen). Mir sind andererseits allerdings auch keine biologischen Fakten bekannt, die die Theorie stützen würden, daß organische Gehirne in der Tat Mittel besitzen, all ihre Synapsen einzeln[9] zu adressieren.

Zukünftige Forschungen. Der Lernalgorithmus aus Abschnitt 8.2 basiert auf dem Konzept separater Trainingsintervalle. Nach jeder Trainingssequenz wird die am Ende des Intervalls durch Selbstmodifikation gewonnene Ge-

[9] Wie bereits verschiedentlich erwähnt, besteht dieses Kapitel *nicht* auf Schemata zur Adressierung *einzelner* Gewichte. Es existieren viele alternative Möglichkeiten, g zu wählen und die Gleichungen (8.3) und (8.5) entsprechend umzudefinieren.

wichtsmatrix einfach weggeworfen. Eine Alternative bestünde darin, die *finale* Gewichtsmatrix gegen die bisher beste zu testen und sie zu behalten, falls sie besser ist[10]. Ich würde jedoch ein hypothetisches Lernsystem vorziehen, das nicht von Anfang an auf das Konzept von Trainingssequenzen festgelegt ist. Statt dessen sollte dieses hypothetische System in der Lage sein, zu lernen, einen kontinuierlichen Eingabestrom dergestalt zu segmentieren und zu manipulieren, daß es sich selbst so etwas wie *maßgeschneiderte* Trainingssequenzen zurechtlegen kann. Im Rahmen zukünftiger Forschungen soll versucht werden, geeignete Lernalgorithmen (mit formal beweisbarer Wirksamkeit) für derartige Systeme zu entwerfen.

Wie bereits durch den Titel nahegelegt, stellt dieses abschließende Kapitel[11] einen Ausblick dar. Es möchte sich selbst im Rahmen eines Gedankenexperimentes auf das Aufweisen der theoretischen Möglichkeit gewisser 'selbstreferentieller' Gewichtsänderungsalgorithmen beschränken – das konstruktive Beispiel aus den vorangegangenen Abschnitten stellt gewiß nicht den effizientesten oder praktikabelsten gradientenbasierten 'selbstreferentiellen' Lernalgorithmus dar. Der Entwurf alternativer, effizienterer Adressierschemata sowie experimentelle Auswertungen spezifischer Details sollen zukünftigen Forschungen überlassen bleiben (ein Schritt in diese Richtung wurde in [116] getan).

[10] Bei dem alternativen Netzwerk aus Fußnote[8] (Abschnitt 8.1.3) würde dies darauf hinauslaufen, die Aktivationen des Netzwerks am Ende eines Trainingsintervalls *nicht* von neuem zu initialisieren. Denn es könnte ja sein, daß die *Aktivationen* einen nützlichen 'selbstreferentiellen' Lernalgorithmus darstellen.

[11] Das Kapitel ist teilweise inspiriert durch ältere Ideen zum Thema 'Lernen lernen' – [88] beschreibt einen 'selbstreferentiellen' genetischen Algorithmus sowie einge weitere 'introspektive' Systeme.

8.4 SCHLUSSBEMERKUNGEN ZUR ARBEIT

Die Idee, Gradientenabstiegsverfahren in der Informatik zur Auffindung einfacher Rechenvorschriften einzusetzen, ist im Prinzip alt. Ihre Ursprünge lassen sich mindestens bis zu Rosenblatts einfachem linearen assoziativen Speicher zurückverfolgen [83]. Daß dieser Idee jedoch große Allgemeinheit zukommt und daß sie zahlreiche weitergehende Anwendungsmöglichkeiten gestattet, fand allerdings erst in jüngster Zeit Aufmerksamkeit. Ich habe mich im Rahmen dieser Arbeit bemüht, an zumeist neuartigen Architekturen einige wichtige und häufig wiederkehrende Aspekte des Entwurfs gradientenbasierter KNN darzulegen sowie klarzumachen, daß die formale Spezifikation gewünschter *'Software'*-Eigenschaften durch Zielfunktionen, die bezüglich der *'Software'* differenzierbar sind, eine sehr vielseitige Alternative zu herkömmlichen Lösungsspezifikationsansätzen darstellt.

In Kapitel 2 haben wir gesehen, wie sich Gradientenabstiegsverfahren zur zielgerichteten Suche in Räumen beliebig komplexer sequentiell/paralleler Algorithmen für rückgekoppelte sequenzverarbeitende Netzwerke herleiten lassen.

In Kapitel 3 wurde beispielhaft demonstriert, daß rückgekoppelte Netzwerke nicht die einzige Architektur darstellen, die die gradientenbasierte Herleitung sequenzverarbeitender Algorithmen erlaubt.

Die Kapitel 3, 4, und 6 zeigten in den höchst unterschiedlichen Kontexten des überwachten Lernens, des *'Reinforcement'*-Lernens und des unüberwachten Lernens eine ganze Reihe von Möglichkeiten auf, durch *Hintereinanderschaltung* mehrerer differenzierbarer adaptiver Module (mit oft sehr unterschiedlichen Aufgaben) geeignete Lernalgorithmen für die jeweilige Problemstellung herzuleiten.

Kapitel 8 trieb die Grundidee ins Extreme: Dort wurde der Gewichtsänderungsalgorithmus eines Netzwerkes dergestalt in differenzierbarer Form repräsentiert, daß sich eine zielgerichtete gradientenbasierte Suche im Raum der Gewichtsänderungsalgorithmen selbst veranstalten ließ.

Die in dieser Arbeit abgehandelten Methoden gehen zwar in vieler Hinsicht über das bekannteste bisher mit Erfolg angewendete gradientenbasierte Verfahren (nämlich einfaches BP) hinaus, schöpfen jedoch das Potential der Idee gradientenbasierter Suche nach formal durch geeignete Zielfunktionen

spezifizierten Algorithmen nicht aus. Vielmehr stellen sie lediglich bedeutende Repräsentanten einer umfangreichen Klasse gradientenbasierter Algorithmen dar, die erst in Ansätzen erforscht ist. Ich glaube, daß sich viele altbekannte Konzepte der Informatik so modifizieren lassen, daß sie sich in gradientenbasierte Systeme einbetten lassen. Als Beispiel diene der aus der Informatik nicht wegzudenkende 'Keller': [20] und [133] beschreiben im Kontext des Erlernens kontextfreier Grammatiken 'differenzierbare' Keller, die nicht nur diskrete Standardoperationen wie *'push'* und *'pop'* erlauben, sondern auch beliebige Zwischenabstufungen wie *'push ein bißchen'* oder *'pop ein bißchen mehr'*: Kellerinhalte werden dabei als kontinuierliche Aktivationsmuster repräsentiert, und Kellerinhalte sind bezüglich der Funktionen, welche Kellerinhalte verändern, differenzierbar. Dies erlaubt die Herleitung gradientenbasierter Algorithmen zum Erlernen der Ausführung geeigneter Kellermodifikationen zu geeigneten Zeitpunkten, wobei der Begriff 'geeignet' wie stets indirekt durch eine angemessene Zielfunktion spezifiziert wird. Eine interessante und vielschichtige Fragestellung ist, welche der vielbenützten diskreten Datenstrukturen und Standardalgorithmen der Informatik (wie Keller und Kelleroperationen, oder aber beispielsweise auch Logikklauseln und Unifikation) sich in ähnlicher Weise dergestalt sinnvoll in ein Kontinuum von Datenstrukturen und zugehörigen Algorithmen einbetten lassen, daß sie geeignete Differenzierbarkeitskriterien erfüllen und dadurch mit lernenden KNN zusammenschaltbar werden. Hier öffnet sich ein vielversprechendes, noch weitgehend unbeackertes Forschungsfeld.

Das Konzept der gradientenbasierten Suchverfahren in Räumen von Rechenvorschriften stellt einen der wichtigsten Beiträge der KNN-Forschung zur Informatik dar, obwohl die umfassende Bedeutung und Anwendbarkeit dieses Konzeptes in früheren Arbeiten meines Wissens nach so gut wie gar nicht zum Ausdruck kam, sondern statt dessen durch die Betonung anderer typischer KNN-Eigenschaften wie beispielsweise der Eignung zur 'subsymbolischen' und massiv parallelen Informationsverarbeitung (die für gradientenbasierte Lernalgorithmen in keiner Weise erforderlich ist) in den Hintergrund trat.

Aus den Überlegungen in Kapitel 7 ist allerdings ersichtlich, daß auch das Konzept des Gradientenabstiegs im Algorithmenraum für sich genommen kein Allheilmittel sein kann – gewisse einfache, *nicht* kettenregelspezifische Einsichten gestatten zumindest gelegentlich, die Performanz gradientenbasierter Lernalgorithmen gewaltig zu verbessern.

Literaturverzeichnis

[1] A.M.H.J. Aertsen, G.L. Gerstein, M.K. Habib, and G. Palm. Dynamics of neuronal firing correlation: Modulation of "effective connectivity". *Journal of Neurophysiology*, 61:900–917, 1989.

[2] L. B. Almeida. A learning rule for asynchronous perceptrons with feedback in a combinatorial environment. In *IEEE 1st International Conference on Neural Networks, San Diego*, volume 2, pages 609–618, 1987.

[3] J. Bachrach. Learning to represent state, 1988. Unpublished master's thesis, University of Massachusetts, Amherst.

[4] H. B. Barlow. Unsupervised learning. *Neural Computation*, 1(3):295–311, 1989.

[5] H. B. Barlow, T. P. Kaushal, and G. J. Mitchison. Finding minimum entropy codes. *Neural Computation*, 1(3):412–423, 1989.

[6] A. G. Barto. Connectionist approaches for control. Technical Report COINS Technical Report 89-89, University of Massachusetts, Amherst MA 01003, 1989.

[7] A. G. Barto and P. Anandan. Pattern recognizing stochastic learning automata. *IEEE Transactions on Systems, Man, and Cybernetics*, 15:360–375, 1985.

[8] A. G. Barto and M. I. Jordan. Gradient following without back propagation in layered networks. In *IEEE 1st International Conference on Neural Networks, San Diego*, volume 2, pages 629–636, 1987.

[9] A. G. Barto, R. S. Sutton, and C. W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE*

Transactions on Systems, Man, and Cybernetics, SMC-13:834–846, 1983.

[10] E. B. Baum and D. Haussler. What size net gives valid generalization? *Neural Computation*, 1(1):151–160, 1989.

[11] S. Becker and G. E. Hinton. Spatial coherence as an internal teacher for a neural network. Technical Report CRG-TR-89-7, Department of Computer Science, University of Toronto, Ontario, 1989.

[12] K. Bergner. Diploma thesis, 1991. Institut für Informatik, Technische Universität München.

[13] G. L. Bilbro and D. E. Van den Bout. Maximum entropy and learning theory. *Neural Computation*, 4(6):839–852, 1992.

[14] U. Bodenhausen and A. Waibel. The tempo 2 algorithm: Adjusting time-delays by supervised learning. In D. S. Lippman, J. E. Moody, and D. S. Touretzky, editors, *Advances in Neural Information Processing Systems 3*, pages 155–161. San Mateo, CA: Morgan Kaufmann, 1991.

[15] H. Bourlard, N. Morgan, and C. Wooters. Connectionist approaches to the use of Markov models for speech recognition. In D. S. Lippman, J. E. Moody, and D. S. Touretzky, editors, *Advances in Neural Information Processing Systems 3*, pages 213–219. San Mateo, CA: Morgan Kaufmann, 1991.

[16] J. S. Bridle and D. J. C. MacKay. Unsupervised classifiers, mutual information and 'phantom' targets. In D. S. Lippman, J. E. Moody, and D. S. Touretzky, editors, *Advances in Neural Information Processing Systems 4,*. San Mateo, CA: Morgan Kaufmann, 1992.

[17] G.J. Chaitin. A theory of program size formally identical to information theory. *Journal of the ACM*, 22:329–340, 1965.

[18] Y. Chauvin. Generalization performance of overtrained networks. In L. B. Almeida and C. J. Wellekens, editors, *Proc. of the EURASIP'90 Workshop, Portugal*, page 46. Springer, 1990.

[19] P. Cheeseman, M. Self, J. Kelly, J. Stutz, W. Taylor, and D. Freeman. AutoClass: a Bayesian classification system. In *Machine Learning: Proceedings of the Fifth International Workshop*. Morgan Kaufmann, San Mateo, CA, 1988.

[20] S. Das, C.L. Giles, and G.Z. Sun. Learning context-free grammars: Capabilities and limitations of a neural network with an external stack memory. In *Proceedings of the The Fourteenth Annual Conference of the Cognitive Science Society, Bloomington*, 1992.

[21] P. Dayan and G. Hinton. Feudal reinforcement learning. In D. S. Lippman, J. E. Moody, and D. S. Touretzky, editors, *Advances in Neural Information Processing Systems 5,*. San Mateo, CA: Morgan Kaufmann, 1993. In preparation.

[22] M. Eldracher and B. Baginski. Neural subgoal generation using backpropagation. In *Submitted to WCNN'93*, July 1993.

[23] J. L. Elman. Finding structure in time. Technical Report CRL Technical Report 8801, Center for Research in Language, University of California, San Diego, 1988.

[24] S. E. Fahlman. An empirical study of learning speed in backpropagation networks. Technical Report CMU-CS-88-162, Carnegie-Mellon Univ., 1988.

[25] R. Farber, A. Lapedes, and K. Sirotkin. Determination of eukaryotic protein coding regions using neural networks and information theory. *Journal of Molecular Biology*, 226:471–479, 1992.

[26] W. Finnoff. Diffusion approximations for the constant learning rate backpropagation algorithm and resistance to local minima. In D. S. Lippman, J. E. Moody, and D. S. Touretzky, editors, *Advances in Neural Information Processing Systems 5,*. San Mateo, CA: Morgan Kaufmann, 1993. In preparation.

[27] P. Földiák. Forming sparse representations by local anti-hebbian learning. *Biological Cybernetics*, 64:165–170, 1990.

[28] M. Gherrity. A learning algorithm for analog fully recurrent neural networks. In *IEEE/INNS International Joint Conference on Neural Networks, San Diego*, volume 1, pages 643–644, 1989.

[29] C. L. Giles, C. B. Miller, D. Chen, H. H. Chen, G. Z. Sun, and Y. C. Lee. Learning and extracting finite state automata with second-order recurrent neural networks. *Neural Computation*, 4:393–405, 1992.

[30] B. Glavina. Planung kollisionsfreier Bewegungen für Manipulatoren durch Kombination von zielgerichteter Suche und zufallsgesteuerter Zwischenzielerzeugung. Dissertation, 1991.

[31] S. Grossberg. Adaptive pattern classification and universal recoding, 1: Parallel development and coding of neural feature detectors. *Biological Cybernetics*, 23:187–202, 1976.

[32] D. Haussler, M. Kearns, M. Opper, and R. Schapire. Estimating average-case learning curves using Bayesian, statistical physics and VC dimension methods. In D. S. Lippman, J. E. Moody, and D. S. Touretzky, editors, *Advances in Neural Information Processing Systems 4,*, pages 855–862. San Mateo, CA: Morgan Kaufmann, 1992.

[33] D. O. Hebb. *The Organization of Behavior.* Wiley, New York, 1949.

[34] G. Held. *Data Compression.* Wiley and Sons LTD, New York, 1991.

[35] G. E. Hinton and S. Becker. An unsupervised learning procedure that discovers surfaces in random-dot stereograms. In *Proc. IEEE/INNS International Joint Conference on Neural Networks*, volume 1, pages 218–222. Hillsdale, NJ. Erlbaum, 1990.

[36] G. E. Hinton and T. E. Sejnowski. Learning and relearning in Boltzmann machines. In *Parallel Distributed Processing*, volume 1, pages 282–317. MIT Press, 1986.

[37] J. Hochreiter. Diploma thesis, 1991. Institut für Informatik, Technische Universität München.

[38] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proc. of the National Academy of Sciences*, 79:2554–2558, 1982.

[39] C. Ji and D. Psaltis. The VC dimension versus the statistical capacity of multilayer networks. In D. S. Lippman, J. E. Moody, and D. S. Touretzky, editors, *Advances in Neural Information Processing Systems 4,*, pages 928–935. San Mateo, CA: Morgan Kaufmann, 1992.

[40] M. I. Jordan. Serial order: A parallel distributed processing approach. Technical Report ICS Report 8604, Institute for Cognitive Science, University of California, San Diego, 1986.

[41] M. I. Jordan and D. E. Rumelhart. Supervised learning with a distal teacher. Technical Report Occasional Paper #40, Center for Cog. Sci., Massachusetts Institute of Technology, 1990.

[42] T. Kohonen. *Self-Organization and Associative Memory.* Springer, second edition, 1988.

[43] A.N. Kolmogorov. Three approaches to the quantitative definition of information. *Problems of Information Transmission*, 1:1–11, 1965.

[44] S. Kullback. *Statistics and Information Theory.* J. Wiley and Sons, New York, 1959.

[45] A. Lapedes and R. Faber. How neural nets work. In D. Z. Anderson, editor, *'Neural Information Processing Systems: Natural and Synthetic' (NIPS).* NY, American Institute of Physics, 1987.

[46] Y. LeCun. Une procédure d'apprentissage pour réseau à seuil asymétrique. *Proceedings of Cognitiva 85, Paris*, pages 599–604, 1985.

[47] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Back-propagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989.

[48] D. Lenat. Theory formation by heuristic search. *Machine Learning*, 21, 1983.

[49] R. Linsker. Self-organization in a perceptual network. *IEEE Computer*, 21:105–117, 1988.

[50] R. Linsker. How to generate ordered maps by maximizing the mutual information between input and output. *Neural Computation*, 1(3):402–411, 1989.

[51] G. Lukes. Review of Schmidhuber's paper 'Recurrent networks adjusted by adaptive critics'. *Neural Network Reviews*, 4(1):41–42, 1990.

[52] Y. Lyuu and I. Rivin. Tight bounds on transition to perfect generalization. *Neural Computation*, 4(6):854–862, 1992.

[53] D. J. C. MacKay. Bayesian interpolation. *Neural Computation*, 4:415–447, 1992.

[54] D. J. C. MacKay. A practical Bayesian framework for backprop networks. *Neural Computation*, 4:448–472, 1992.

[55] M. Minsky. Steps toward artificial intelligence. In E. Feigenbaum and J. Feldman, editors, *Computers and Thought*, pages 406–450. McGraw-Hill, New York, 1963.

[56] M. Minsky and S. Papert. *Perceptrons.* Cambridge, MA: MIT Press, 1969.

[57] K. Möller and S. Thrun. Task modularization by network modulation. In J. Rault, editor, *Proceedings of Neuro-Nimes '90*, pages 419–432, November 1990.

[58] J. E. Moody. The effective number of parameters: An analysis of generalization and regularization in nonlinear learning systems. In D. S. Lippman, J. E. Moody, and D. S. Touretzky, editors, *Advances in Neural Information Processing Systems 4,*, pages 847–854. San Mateo, CA: Morgan Kaufmann, 1992.

[59] M. C. Mozer. A focused back-propagation algorithm for temporal sequence recognition. *Complex Systems*, 3:349–381, 1989.

[60] M. C. Mozer. Connectionist music composition based on melodic, stylistic, and psychophysical constraints. Technical Report CU-CS-495-90, University of Colorado at Boulder, 1990.

[61] M. C. Mozer. Induction of multiscale temporal structure. In D. S. Lippman, J. E. Moody, and D. S. Touretzky, editors, *Advances in Neural Information Processing Systems 4,*, pages 275–282. San Mateo, CA: Morgan Kaufmann, 1992.

[62] C. Myers. Learning with delayed reinforcement through attention-driven buffering. Technical report, Neural Systems Engineering Group, Dept. of Electrical Engineering, Imperial College of Science, Technology and Medicine, 1990.

[63] Nguyen and B. Widrow. The truck backer-upper: An example of self learning in neural networks. In *IEEE/INNS International Joint Conference on Neural Networks, Washington, D.C.*, volume 1, pages 357–364, 1989.

[64] S. J. Nowlan. Auto-encoding with entropy constraints. In *Proceedings of INNS First Annual Meeting, Boston, MA.*, 1988. Also published in special supplement to Neural Networks.

[65] E. Oja. Neural networks, principal components, and subspaces. *International Journal of Neural Systems*, 1(1):61–68, 1989.

[66] D. B. Parker. Learning-logic. Technical Report TR-47, Center for Comp. Research in Economics and Management Sci., MIT, 1985.

[67] D. B. Parker. Optimal algorithms for adaptive networks: Second order back propagation, second order direct propagation, and second order hebbian learning. In *IEEE 1st International Conference on Neural Networks, San Diego*, volume 2, pages 593–600, 1987.

[68] B. A. Pearlmutter. Learning state space trajectories in recurrent neural networks. *Neural Computation*, 1(2):263–269, 1989.

[69] B. A. Pearlmutter and G. E. Hinton. G-maximization: An unsupervised learning procedure for discovering regularities. In J. S. Denker, editor, *Neural Networks for Computing: American Institute of Physics Conference Proceedings 151*, volume 2, pages 333–338, 1986.

[70] F. J. Pineda. Dynamics and architecture for neural computation. *Journal of Complexity*, 4:216–245, 1988.

[71] F. J. Pineda. Recurrent backpropagation and the dynamical approach to adaptive neural computation. *Neural Computation*, 1(2):161–172, 1989.

[72] F. J. Pineda. Time dependent adaptive neural networks. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, pages 710–718. San Mateo, CA: Morgan Kaufmann, 1990.

[73] M. D. Plumbley. On information theory and unsupervised neural networks. Dissertation, published as technical report CUED/F-INFENG/TR.78, Engineering Department, Cambridge University, 1991.

[74] J. B. Pollack. Recursive distributed representation. *Artificial Intelligence*, 46:77–105, 1990.

[75] D. Prelinger. Diploma thesis, 1992. Institut für Informatik, Technische Universität München.

[76] M. B. Ring. PhD Proposal: Autonomous construction of sensorimotor hierarchies in neural networks. Technical report, University of Texas at Austin, 1990.

[77] M. B. Ring. Incremental development of complex behaviors through automatic construction of sensory-motor hierarchies. In L. Birnbaum and G. Collins, editors, *Machine Learning: Proceedings of the Eighth International Workshop*, pages 343–347. Morgan Kaufmann, 1991.

[78] A. J. Robinson. *Dynamic Error Propagation Networks.* PhD thesis, Trinity Hall and Cambridge University Engineering Department, 1989.

[79] A. J. Robinson and F. Fallside. The utility driven dynamic error propagation network. Technical Report CUED/F-INFENG/TR.1, Cambridge University Engineering Department, 1987.

[80] R. Rohwer. The 'moving targets' training method. In J. Kindermann and A. Linden, editors, *Proceedings of 'Distributed Adaptive Neural Information Processing', St.Augustin, 24.-25.5,*. Oldenbourg, 1989.

[81] R. Rohwer and B. Forrest. Training time-dependence in neural networks. In *IEEE 1st International Conference on Neural Networks, San Diego*, volume 2, pages 701–708, 1987.

[82] M. Röscheisen, R. Hofmann, and V. Tresp. Neural control for rolling mills: Incorporating domain theories to overcome data deficiencies. In D. S. Lippman, J. E. Moody, and D. S. Touretzky, editors, *Advances in Neural Information Processing Systems 4,*, pages 659–666. San Mateo, CA: Morgan Kaufmann, 1992.

[83] F. Rosenblatt. *Principles of Neurodynamics.* Spartan, New York, 1962.

[84] J. Rubner and K. Schulten. Development of feature detectors by self-organization: A network model. *Biological Cybernetics*, 62:193–199, 1990.

[85] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In *Parallel Distributed Processing*, volume 1, pages 318–362. MIT Press, 1986.

[86] D. E. Rumelhart and D. Zipser. Feature discovery by competitive learning. In *Parallel Distributed Processing*, pages 151–193. MIT Press, 1986.

[87] T. D. Sanger. An optimality principle for unsupervised learning. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 1*, pages 11–19. San Mateo, CA: Morgan Kaufmann, 1989.

[88] J. H. Schmidhuber. Evolutionary principles in self-referential learning, or on learning how to learn: The meta-meta-... hook, 1987. Institut für Informatik, Technische Universität München.

[89] J. H. Schmidhuber. Accelerated learning in back-propagation nets. In R. Pfeifer, Z. Schreter, Z. Fogelman, and L. Steels, editors, *Connectionism in Perspective*, pages 429 – 438. Amsterdam: Elsevier, North-Holland, 1989.

[90] J. H. Schmidhuber. A local learning algorithm for dynamic feedforward and recurrent networks. *Connection Science*, 1(4):403–412, 1989.

[91] J. H. Schmidhuber. The neural bucket brigade. In R. Pfeifer, Z. Schreter, Z. Fogelman, and L. Steels, editors, *Connectionism in Perspective*, pages 439–446. Amsterdam: Elsevier, North-Holland, 1989.

[92] J. H. Schmidhuber. Additional remarks on G. Lukes' review of Schmidhuber's paper 'Recurrent networks adjusted by adaptive critics'. *Neural Network Reviews*, 4(1):43, 1990.

[93] J. H. Schmidhuber. Dynamische neuronale Netze und das fundamentale raumzeitliche Lernproblem. Dissertation, Institut für Informatik, Technische Universität München, 1990.

[94] J. H. Schmidhuber. Learning algorithms for networks with internal and external feedback. In D. S. Touretzky, J. L. Elman, T. J. Sejnowski, and G. E. Hinton, editors, *Proc. of the 1990 Connectionist Models Summer School*, pages 52–61. San Mateo, CA: Morgan Kaufmann, 1990.

[95] J. H. Schmidhuber. Networks adjusting networks. In J. Kindermann and A. Linden, editors, *Proceedings of 'Distributed Adaptive Neural Information Processing', St.Augustin, 24.-25.5. 1989*, pages 197–208. Oldenbourg, 1990. In November 1990 a revised and extended version appeared as FKI-Report FKI-125-90 (revised) at the Institut für Informatik, Technische Universität München.

[96] J. H. Schmidhuber. An on-line algorithm for dynamic reinforcement learning and planning in reactive environments. In *Proc. IEEE/INNS International Joint Conference on Neural Networks, San Diego*, volume 2, pages 253–258, 1990.

[97] J. H. Schmidhuber. Recurrent networks adjusted by adaptive critics. In *Proc. IEEE/INNS International Joint Conference on Neural Networks, Washington, D. C.*, volume 1, pages 719–722, 1990.

[98] J. H. Schmidhuber. Reinforcement learning with interacting continually running fully recurrent networks. In *Proc. INNC International Neural Network Conference, Paris*, volume 2, pages 817–820, 1990.

[99] J. H. Schmidhuber. Reinforcement-Lernen und adaptive Steuerung. *Nachrichten Neuronale Netze*, 2:1–3, 1990.

[100] J. H. Schmidhuber. Temporal-difference-driven learning in recurrent networks. In R. Eckmiller, G. Hartmann, and G. Hauske, editors, *Parallel Processing in Neural Systems and Computers*, pages 209–212. North-Holland, 1990.

[101] J. H. Schmidhuber. Adaptive decomposition of time. In T. Kohonen, K. Mäkisara, O. Simula, and J. Kangas, editors, *Artificial Neural Networks*, pages 909–914. Elsevier Science Publishers B.V., North-Holland, 1991.

[102] J. H. Schmidhuber. Adaptive history compression for learning to divide and conquer. In *Proc. International Joint Conference on Neural Networks, Singapore*, volume 2, pages 1130–1135. IEEE, 1991.

[103] J. H. Schmidhuber. Curious model-building control systems. In *Proc. International Joint Conference on Neural Networks, Singapore*, volume 2, pages 1458–1463. IEEE, 1991.

[104] J. H. Schmidhuber. Learning factorial codes by predictability minimization. Technical Report CU-CS-565-91, Dept. of Comp. Sci., University of Colorado at Boulder, December 1991.

[105] J. H. Schmidhuber. Learning temporary variable binding with dynamic links. In *Proc. International Joint Conference on Neural Networks, Singapore*, volume 3, pages 2075–2079. IEEE, 1991.

[106] J. H. Schmidhuber. Learning to generate sub-goals for action sequences. In T. Kohonen, K. Mäkisara, O. Simula, and J. Kangas, editors, *Artificial Neural Networks*, pages 967–972. Elsevier Science Publishers B.V., North-Holland, 1991.

[107] J. H. Schmidhuber. A possibility for implementing curiosity and boredom in model-building neural controllers. In J. A. Meyer and S. W. Wilson, editors, *Proc. of the International Conference on Simulation of Adaptive Behavior: From Animals to Animats*, pages 222–227. MIT Press/Bradford Books, 1991.

[108] J. H. Schmidhuber. Reinforcement learning in markovian and non-markovian environments. In D. S. Lippman, J. E. Moody, and D. S. Touretzky, editors, *Advances in Neural Information Processing Systems 3*, pages 500–506. San Mateo, CA: Morgan Kaufmann, 1991.

[109] J. H. Schmidhuber. A fixed size storage $O(n^3)$ time complexity learning algorithm for fully recurrent continually running networks. *Neural Computation*, 4(2):243–248, 1992.

[110] J. H. Schmidhuber. Learning complex, extended sequences using the principle of history compression. *Neural Computation*, 4(2):234–242, 1992.

[111] J. H. Schmidhuber. Learning factorial codes by predictability minimization. *Neural Computation*, 4(6):863–879, 1992.

[112] J. H. Schmidhuber. Learning to control fast-weight memories: An alternative to recurrent nets. *Neural Computation*, 4(1):131–139, 1992.

[113] J. H. Schmidhuber. Learning unambiguous reduced sequence descriptions. In J. E. Moody, S. J. Hanson, and R. P. Lippman, editors, *Advances in Neural Information Processing Systems 4*, pages 291–298. San Mateo, CA: Morgan Kaufmann, 1992.

[114] J. H. Schmidhuber. Steps towards 'self-referential' learning. Technical Report CU-CS-627-92, Dept. of Comp. Sci., University of Colorado at Boulder, November 1992.

[115] J. H. Schmidhuber. A neural network that embeds its own meta-levels. In *Proc. of the International Conference on Neural Networks '93, San Francisco*. IEEE, 1993. Accepted for publication.

[116] J. H. Schmidhuber. On decreasing the ratio between learning complexity and number of time varying variables in fully recurrent nets. Technical report, Institut für Informatik, Technische Universität München, 1993. In preparation.

[117] J. H. Schmidhuber and R. Huber. Learning to generate artificial fovea trajectories for target detection. *International Journal of Neural Systems*, 2(1 & 2):135–141, 1991.

[118] J. H. Schmidhuber and R. Huber. Using sequential adaptive neurocontrol for efficient learning of rotation and translation invariance. In T. Kohonen, K. Mäkisara, O. Simula, and J. Kangas, editors, *Artificial Neural Networks*, pages 315–320. Elsevier Science Publishers B.V., North-Holland, 1991.

[119] J. H. Schmidhuber, M. C. Mozer, and D. Prelinger. Continuous history compression. Technical report, Dept. of Comp. Sci., University of Colorado at Boulder, 1993. In preparation.

[120] J. H. Schmidhuber and D. Prelinger. Discovering predictable classifications. Technical Report CU-CS-626-92, Dept. of Comp. Sci., University of Colorado at Boulder, November 1992.

[121] J. H. Schmidhuber and D. Prelinger. Discovering predictable classifications. *Accepted by Neural Computation*, 1993.

[122] J. H. Schmidhuber and R. Wahnsiedler. Planning simple trajectories using neural subgoal generators. In J. A. Meyer, H. Roitblat, and S. Wilson, editors, *Proc. of the 2nd International Conference on Simulation of Adaptive Behavior*. MIT Press, 1992. In press.

[123] B. Schürmann. Stability and adaptation in artificial neural systems. *Physical Review*, A 40(50):2681–2688, 1989.

[124] C. E. Shannon. A mathematical theory of communication (part III). *Bell System Technical Journal*, XXVII:623–656, 1948.

[125] C. E. Shannon. A mathematical theory of communication (parts I and II). *Bell System Technical Journal*, XXVII:379–423, 1948.

[126] F. M. Silva and L. B. Almeida. A distributed decorrelation algorithm. In Erol Gelenbe, editor, *Neural Networks, Advances and Applications*. North-Holland, 1991.

[127] P.Y. Simard and Y. LeCun. Local computation of the second derivative information in a multi-layer network. In D. S. Lippman, J. E. Moody, and D. S. Touretzky, editors, *Advances in Neural Information Processing Systems 5,*. San Mateo, CA: Morgan Kaufmann, 1993. In preparation.

[128] S.P. Singh. The efficient learning of multiple task sequences. In J.E. Moody, S.J. Hanson, and R.P. Lippman, editors, *Advances in Neural Information Processing Systems 4*, San Mateo, CA, 1992. Morgan Kaufmann.

[129] A. W. Smith and D. Zipser. Learning sequential structures with the real-time recurrent learning algorithm. *International Journal of Neural Systems*, 1:125–131, 1990.

[130] S. Solla. The emergence of generalization ability in learning. In D. S. Lippman, J. E. Moody, and D. S. Touretzky, editors, *Advances in Neural Information Processing Systems 5,*. San Mateo, CA: Morgan Kaufmann, 1993. In preparation.

[131] S. A. Solla. Accelerated learning in layered neural networks. *Complex Systems*, 2:625–640, 1988.

[132] P. Stolorz, A. Lapedes, and X. Xia. Predicting protein secondary structure using neural net and statistical methods. *Journal of Molecular Biology*, 225:363–377, 1992.

[133] G.Z. Sun, H.H. Chen, C.L. Giles, Y.C. Lee, and D. Chen. Connectionist pushdown automata that learn context-free grammars. In *Proceedings of the International Joint Conference on Neural Networks IJCNN-90*, volume 1, page 577. Lawrence Erlbaum, Hillsdale, N.J., 1990.

[134] G. Tesauro. Practical issues in temporal difference learning. In D. S. Lippman, J. E. Moody, and D. S. Touretzky, editors, *Advances in Neural Information Processing Systems 4*,, pages 259–266. San Mateo, CA: Morgan Kaufmann, 1992.

[135] N. Tishby, E. Levin, and S. Solla. Consistence inference of probabilities in layered networks: predictions and generalizations. In *Proceedings of the International Joint Conference on Neural Networks IJCNN-90*, volume 2, pages 403–409. Lawrence Erlbaum, Hillsdale, N.J., 1990.

[136] A. C. Tsoi and R. A. Pearson. Comparison of three classification techniques, CART, C4.5 and multi-layer perceptrons. In D. S. Lippman, J. E. Moody, and D. S. Touretzky, editors, *Advances in Neural Information Processing Systems 3*, pages 963–969. San Mateo, CA: Morgan Kaufmann, 1991.

[137] V. Vapnik. Principles of risk minimization for learning theory. In D. S. Lippman, J. E. Moody, and D. S. Touretzky, editors, *Advances in Neural Information Processing Systems 4*,, pages 831–838. San Mateo, CA: Morgan Kaufmann, 1992.

[138] C. v.d. Malsburg. Technical Report 81-2, Abteilung für Neurobiologie, Max-Planck Institut für Biophysik und Chemie, Göttingen, 1981.

[139] R. Wahnsiedler. Diploma thesis, 1992. Institut für Informatik, Technische Universität München.

[140] C. Watkins. *Learning from Delayed Rewards*. PhD thesis, King's College, 1989.

[141] R. L. Watrous and G. M. Kuhn. Induction of finite-state languages using second-order recurrent networks. *Neural Computation*, 4:406–414, 1992.

[142] A. S. Weigend, B. A. Huberman, and D. E. Rumelhart. Predicting the future: A connectionist approach. *International Journal of Neural Systems*, 1:193–209, 1990.

[143] P. J. Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences.* PhD thesis, Harvard University, 1974.

[144] P. J. Werbos. Generalization of backpropagation with application to a recurrent gas market model. *Neural Networks*, 1, 1988.

[145] H. White. Learning in artificial neural networks: A statistical perspective. *Neural Computation*, 1(4):425–464, 1989.

[146] R. J. Williams. Reinforcement-learning in connectionist networks: A mathematical analysis. Technical Report 8605, Institute for Cognitive Science, University of California, San Diego, 1986.

[147] R. J. Williams. Toward a theory of reinforcement-learning connectionist systems. Technical Report NU-CCS-88-3, College of Comp. Sci., Northeastern University, Boston, MA, 1988.

[148] R. J. Williams. Complexity of exact gradient computation algorithms for recurrent neural networks. Technical Report Technical Report NU-CCS-89-27, Boston: Northeastern University, College of Computer Science, 1989.

[149] R. J. Williams and J. Peng. An efficient gradient-based algorithm for on-line training of recurrent network trajectories. *Neural Computation*, 4:491–501, 1990.

[150] R. J. Williams and D. Zipser. Experimental analysis of the real-time recurrent learning algorithm. *Connection Science*, 1(1):87–111, 1989.

[151] R. J. Williams and D. Zipser. A learning algorithm for continually running fully recurrent networks. *Neural Computation*, 1(2):270–280, 1989.

[152] R. J. Williams and D. Zipser. Gradient-based learning algorithms for recurrent networks and their computational complexity. In *Backpropagation: Theory, Architectures and Applications.* Hillsdale, NJ: Erlbaum, 1992.

[153] D.J. Willshaw and C. von der Malsburg. How patterned neural connections can be set up by self-organization. *Proc. R. Soc. London B*, 194:431–445, 1976.

[154] A. Wyner and J. Ziv. Fixed data base version of the Lempel-Ziv data compression algorithm. *IEEE Transactions Information Theory*, 37:878–880, 1991.

[155] R. S. Zemel and G. E. Hinton. Discovering viewpoint-invariant relationships that characterize objects. In D. S. Lippman, J. E. Moody, and D. S. Touretzky, editors, *Advances in Neural Information Processing Systems 3*, pages 299–305. San Mateo, CA: Morgan Kaufmann, 1991.

[156] D. Zipser. A subgrouping strategy that reduces learning complexity and speeds up learning in recurrent networks. *Neural Computation*, 1(4):552–558, 1989.

[157] D. Zipser. Recurrent network model of short-term active memory. *Neural Computation*, 3(2):179–193, 1991.

[158] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, IT-23(5):337–343, 1977.