Diplomarbeit

Normalisierende Datenkompression mit symmetrischen Neuronalen Netzen

Informationsoptimale Kompression gestörter Daten

eingereicht bei
Dr. habil. R. Brause
AG Neuroinformatik und parallele Systeme

am
Fachbereich Informatik
der
Johann Wolfgang Goethe Universität
Frankfurt am Main

Michael Rippl
An der Leimenkaut 1
61352 Bad Homburg

Für meine Frau Kornelia

Ich möchte an dieser Stelle all denen meinen Dank aussprechen, die mir beim Erstellen meiner Diplomarbeit behilflich waren.

Mein besonderer Dank gilt Herrn Dr. habil. R. Brause, mit dem ich sehr viele interessante Gespräche führen konnte, die über das Thema dieser vorliegenden Diplomarbeit weit hinausgingen.

Ich möchte auch meinem Kommilitonen Armin Schmid danken, der mich das gesamte Studium lang als ein sehr guter Freund begleitet hat.

Weiterhin möchte ich mich bei Heike Precht, Petra Hilbert, Annette Forthmann, Gudrun Hauck, Wolfgang Bühler und vielen anderen Mitarbeitern der Microsoft Niederlassung Bad Homburg bedanken, die mir völlig uneigennützig sehr gute Hilfsmittel für meinen Computer zur Verfügung gestellt haben.

Auch meiner Mutter möchte ich herzlich danken, ohne deren Hilfe ich dieses Studium nie hätte anfangen können. Aufgrund ihrer klugen Ratschläge habe ich einst beschlossen auf ein Gymnasium zu gehen und anschließend ein Studium zu beginnen.

Aber vor allem möchte ich meiner Frau Kornelia danken, die mir während meiner Studienzeit sehr viel Kraft gegeben und mich immer tatkräftig unterstützt hat.

Ehrenwörtliche Erklärung

Ich versichere, daß ich die beiliegende Diplomarbeit selbständig und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel angefertigt und die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen.

Bad Homburg, den 1. Januar 2009

Inhaltsverzeichnis

1. Einleitung

Scientific theories deal with concepts, not with reality. All theoretical results are derived from certain axioms by deductive logic. In physical sciences the theories are so formulated as to correspond in some useful sense to the real world, whatever that may mean. However, this correspondence is approximate, and the physical justification of all theoretical conclusions is based on some form of inductive reasoning.

Athanasios Papoulis

Zu Beginn der achtziger Jahre hat es einen enormen Aufschwung im Wissenschaftsgebiet der Neuronalen Netze gegeben, der bis heute anhält. Ein wichtiger Grund für diesen Aufschwung ist, daß Neuronale Netze sich stark an der Funktionsweise des menschlichen Gehirns orientieren und daher eine hochgradig parallele Informationsverarbeitung durchführen können. Mit Hilfe von heute vorhandenen Mehrprozessorsystemen können Neuronale Netze simuliert werden, wobei durch deren parallele Natur enorme Verarbeitungskapazitäten möglich sind.

Neuronale Netze können aus völlig unterschiedlichen Blickwinkeln heraus untersucht und beschrieben werden. Der Biologe oder Neurophysiologe benutzt Modelle Neuronaler Netze vorwiegend zum Verständnis der biologischen Vorgänge im Gehirn. Der Mathematiker betrachtet sie als eine spezielle Form von abstrakten Automaten und der Informatiker sieht in ihnen hochgradig parallel arbeitende, lernfähige und teilweise selbstorganisiernde Systeme zur Informationsverarbeitung.

Auch in der vorliegenden Diplomarbeit werden Neuronale Netze aus der Perspektive der Informationsverarbeitung betrachtet. Insbesondere wird hier die Fähigkeit Neuronaler Netze genutzt, für eine Menge von Daten eine spezielle Transformation zu bestimmen, auf deren Basis eine Komprimierung der Daten durchgeführt werden kann.

Im heutigen Zeitalter von Multimedia und grafischen Benuteroberflächen entstehen durch immer leistungsfähigere Hardware zur digitalen Informationsverarbeitung wie

- Grafikkarten mit sehr hoher Auflösung und Farbtiefe
- Soundkarten mit hoher Wiedergabequalität
- Frame-Grabber zur digitalen Aufzeichnung von Videos

immer neue Einsatzgebiete, wo enorme Informationsmengen verarbeitet werden müssen. Um den benötigten Speicherplatz für sehr große Datenmengen zu reduzieren, werden häufig klassische Verfahren zur Datenkompression, wie die *Huffman-Kodierung* oder *LZW-Kodierung* eingestzt, die eine fehlerfreie Reduktion von Datenmengen ermöglichen.

In der Praxis hat sich aber gezeigt, daß die mit diesen Verfahren erzielten Kompressionsraten für die digitale Bild- und Tonverarbeitung bei weitem nicht ausreichen.

Für diese beiden Bereiche existieren eine Reihe von Verfahren zur Signalkodierung, die eine fehlerbehaftete Datenkompression mit sehr hohen Kompressionsraten durchführen können und unter dem Begriff *Transform Coding* zusammengefaßt werden. Da die dort angestrebte Komprimierung fehlerbehaftet ist, spielt auch die Qualität der dekomprimierten Daten eine wichtige Rolle.

Die in den nächsten Jahrzehnten wohl zur Realität gewordene Vision der totalen Kommunikation deutet bereits heute an, daß Bild- und Tondaten nicht nur komprimiert, sondern auch über große Entfernungen übertragen werden müssen. Da bei einer Übertragung von Daten sicherlich auch Störungen auftreten können, ist es sinnvoll Untersuchungen durchzuführen, die zeigen, wie sich Störungen auf den Informationsgehalt von Daten auswirken, die vor ihrer Übertragung kodiert und komprimiert wurden.

Eine wichtige Aufgabe meiner Diplomarbeit ist es zu zeigen, daß eine bestimmte Form von Kodierung, welche als *Datenorthonormalisierung* bezeichnet wird, eine möglichst störsichere Übertragung von Daten erlaubt, die vor ihrer Übertragung über ein gestörtes Medium im Sinne des Transform Coding kodiert und komprimiert werden. Damit wird in meiner Diplomarbeit die Datenorthonormalisierung nicht nur formal definiert und anschließend verwendet, sondern es wird mit mathematischen und informationstheoretischen Mitteln begründet, *warum* der Einsatz der Datenorthonormalisierung wesentliche Vorteile bietet.

Es ist das Ziel meiner Diplomarbeit ein von mir entwickeltes biologisch plausibles Modell eines Neuronalen Netzes vorzustellen, das für eine Menge von vorgegebenen Eingabedaten mit Hilfe eines speziellen Lernverfahrens eine Transformation berechnen kann, die eine Datenorthonormalisierung durchführt. Es wird in diesem Zusammenhang auch gezeigt, daß die Datenorthonormalisierung für eine fehlerbehaftete Datenkompression eingesetzt werden kann.

Auf eine saubere und mathematisch korrekte Beweisführung habe ich sehr viel Wert gelegt. So stützen sich alle informationstheoretischen und mathematischen Kapitel meiner Diplomarbeit auf grundlegende Definitionen der Literatur und liefern jeweils sauber nachvollziehbare Beweisketten von den Grundlagen bis hin zu den zentralen Aussagen.

Das 2. Kapitel der Diplomarbeit ist in drei Abschnitte unterteilt. Im ersten Abschnitt wird die Datenorthonormalisierung informationstheoretisch untersucht und gezeigt, daß sie bei der Kodierung und Übertragung von Informationen über ein gestörtes Medium wesentliche Vorteile bietet. Der zweite Abschnitt stellt eine Einführung in die Thematik des Transform Coding dar und liefert zusätzlich eine Vielzahl grundlegender und wichtiger Beweise. Bei dem dritten Abschnitt handelt es sich um eine Einführung in das Gebiet der Neuronalen Netze, wo auch gezeigt wird, wie Neuronale Netze für eine fehlerbehaftete Datenkompression eingesetzt werden können.

Nachdem im 2. Kapitel wichtige Grundlagen erarbeitet wurden, wird im 3. Kapitel ein von mir entwickeltes Modell eines Neuronalen Netzes vorgestellt, welches eine Datenorthonormalisierung durchführen kann. Es werden auch wichtige Beweise zur mathematischen Herleitung sowie Simulationen für die Praxistauglichkeit des Modells angegeben.

Im 4. Kapitel wird eine von mir entwickelte objektorientierte Software-Bibliothek beschrieben, die für alle durchgeführten Simulationen meiner Diplomarbeit verwendet wurde und einen beachtlichen Funktionsumfang besitzt. Mit Hilfe dieser Bibliothek läßt sich die Simulation komplexer Neuronaler Netze stark vereinfachen.

Kapitel 5. enthält ein Resümee und einen Ausblick.

Eine Aufstellung und Erklärung der verwendeten Definitionen und Begriffe befindet sich im Anhang der Diplomarbeit. Dort befinden sich auch die langwierigen und schwierigen Beweise zu den mathematischen Sätzen aus Kapitel 2. und 3. sowie die Schnittstellen der objektorientierten Software-Bibliothek. Zum Abschluß habe ich ein Literaturverzeichnis eingefügt, in dem alle Quellen aufgeführt sind, die zur Erstellung der Diplomarbeit verwendet wurden.

Und nun, viel Spaß beim Lesen.

2. Modelle zur Datenkompression

In diesem Kapitel werden die Grundlagen für den Einsatz Neuronaler Netze zur Datenkompression vorgestellt, wobei ein Schwerpunkt auf die Kompression von Bilddaten gelegt wird. Die dabei betrachteten Techniken zur Datenkompression sind approximativ und damit fehlerbehaftet. Sie bewirken über die Reduktion von Redundanz hinaus auch eine Reduktion von Information, so daß die Qualität der dekomprimierten Daten eine wichtige Rolle spielt.

Aber nicht nur die Datenkompression, sondern auch die Übertragung von komprimierten Daten über einen gestörten Kanal wird in diesem Kapitel behandelt. Hier liefert meine Diplomarbeit eine zentrale Aussage, nach der Daten auf eine bestimmte Art und Weise transformiert werden sollten, um bei der Übertragung dieser Daten über einen gestörten Kanal einen maximalen mittleren Informationsgehalt zu bekommen.

Es handelt sich aber bei diesem Kapitel *nicht* um eine Einführung der Diplomarbeit, die bereits bekannte Ergebnisse der Literatur zusammenfaßt, sondern um ein Kapitel, in dem neben selbst erarbeiteten Aussagen und Beweisen bekannte Ergebnisse aus der Literatur mit großer mathematischer Sorgfalt neu hergeleitet werden, da sie nicht mehr ohne weiteres nachvollziehbar waren. Eine Ausnahme bildet der Abschnitt 2.3., wo mit Hilfe einer Literaturzusammenfassung eine Einführung in das Wissenschaftsgebiet der Neuronalen Netze gegeben wird. Weiterhin werden in diesem Abschnitt bereits bekannte Netzmodelle vorgestellt, die für eine Kompression von Daten eingesetzt werden können.

Ich möchte an dieser Stelle hervorheben, daß alle wichtigen Beweise und damit die zentralen Aussagen in diesem Kapitel selbständig von mir erarbeitet wurden. Natürlich stützen sich die Aussagen auf grundlegende Definitionen der Literatur. Direkt aus der Literatur übernommene Sätze und Beweise sind durch entsprechende Hinweise besonders gekennzeichnet.

Inhaltlich ist das Kapitel ist in drei wichtige Abschnitte aufgeteilt, die sich mit

- Informationstheorie
- Transform Coding
- Neuronalen Netzen

beschäftigen. Jeder Abschnitt liefert aus seiner speziellen Sicht wichtige Ergebnisse, die eine Bewertung sowie ein besseres Verständnis bekannter Kompressionstechniken auf Basis Neuronaler Netze ermöglichen. Darüberhinaus können auf der Grundlage dieser Ergebnisse Methoden zur Datenkompression mit Neuronalen Netzen selbst entwickelt werden.

2.1. Informationstheorie und Datenkompression

Dieses Kapitel vermittelt die Grundlagen der Informationstheorie, die für das Verständnis der hier behandelten Bereiche aus der Kodierungstheorie und Datenkompression unerläßlich sind. Darüber hinaus liefert das Kapitel als wichtigstes Ergebnis zwei informationstheoretische Aussagen, welche zeigen, daß eine bestimmte Form von Kodierung bei der Kompression und Übertragung von Informationen wesentliche Vorteile bietet. Diese als *Datenorthonormalisierung* bezeichnete Kodierung wird in Kapitel 2.1.3. definiert und untersucht.

Ich habe in meiner Diplomarbeit sehr viel Wert auf eine mathematisch korrekte Beweisführung gelegt. So ergibt sich in diesem Kapitel eine saubere und einfach nachvollziehbare Beweiskette von den Grundlagen bis hin zu den zentralen Aussagen, da alle wichtigen Beweise von mir selbst erstellt wurden. Die in der Literatur angegebenen Beweise für einige der hier vorgestellten Sätze waren meiner Meinung nach nicht zufriedenstellend. Ich werde an den entsprechenden Stellen darauf eingehen.

2.1.1. Allgemeine Konzepte

Die Informationstheorie beschäftigt sich mit der Übertragung von Nachrichten und der Verarbeitung von Informationen. Einer der wichtigsten Beiträge zu diesem Wissenschaftsgebiet stammt von C. E. Shannon. Er veröffentlichte 1948 seine grundlegenden Untersuchungen über das Wesen von Nachrichten.

Eine weitere Hauptaufgabe der Informationstheorie ist die Beschreibung und die Bewertung von Übertragungskanälen sowie die Untersuchung geeigneter Kodierungsverfahren zur möglichst störsicheren Übertragung.

Grundmodelle der Kommunikation

Das hier verwendete Grundmodell einer Kommunikation entspricht voll und ganz dem Grundmodell nach Shannon [SHA76] und wird in der folgenden Abbildung dargestellt.

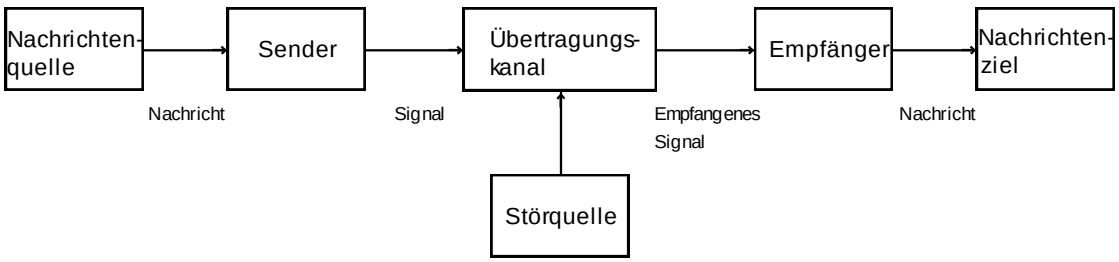


Abbildung 2-1 Grundmodell der Kommunikation nach Shannon

In diesem Modell wird in der Nachrichtenquelle in nicht vorhersehbarer Weise aus einer Menge von möglichen Nachrichten eine gewünschte Nachricht ausgewählt und diese zu einem Bestimmungsort gesendet, der als Nachrichtenziel bezeichnet wird. Der Sender übersetzt diese Nachricht in das Signal, welches dann vom Sender über den Übertragungskanal zum Empfänger übertragen wird. Der Empfänger wandelt das empfangene Signal wieder in eine Nachricht um und gibt diese an das Nachrichtenziel weiter. Ein Beispiel für das obige Kommunikationssystem ist ein Telefongespräch.

Während der Übertragung gehen manchmal bestimmte Teile des Signals verloren oder werden durch hinzugefügte Dinge verfälscht. Diese unerwünschten Veränderungen am übertragenen Signal werden als Störungen bezeichnet.

Das von Shannon vorgeschlagene Grundmodell der Kommunikation ist für meine Diplomarbeit nicht ausreichend. Die in der folgenden Abbildung dargestellte Erweiterung des Modells verdeutlicht, daß es sich bei den zu übertragenden Nachrichten nicht unbedingt um einzelne Zeichen handeln muß, welche sequentiell nacheinander gesendet werden.

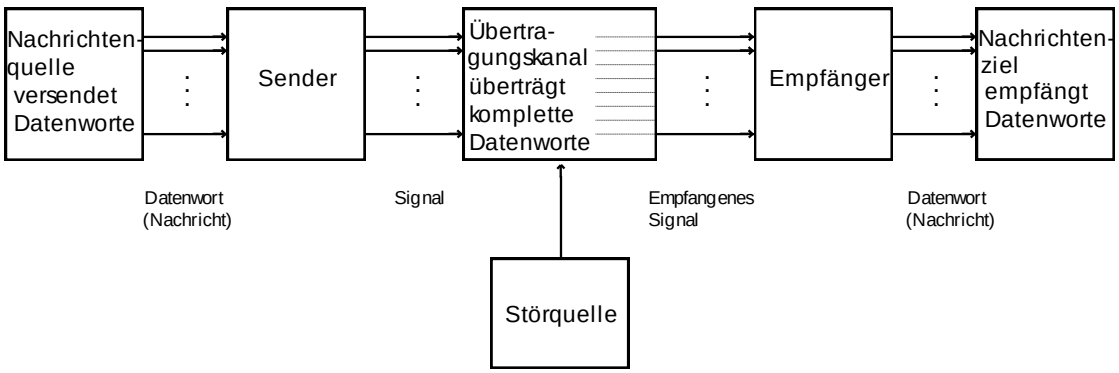


Abbildung 2-2 Erweitertes Grundmodell der Kommunikation

Eine einzelne Nachricht kann im obigen Modell aus einer Menge von Zeichen bestehen und wird in diesem Fall als Datenwort bezeichnet. Ist eine Nachrichtenquelle in der Lage komplette Datenworte auszuwählen und zu versenden, dann wird sie zu einer Verbundquelle. Diese Form von Informationsquellen wird in Kapitel 2.1.2. ausführlich untersucht.

Der Übertragungskanal setzt sich im erweiterten Grundmodell aus mehreren einzelnen Kanälen zusammen, die unabhängig voneinander Signale übertragen können. Dadurch ist es möglich, die einzelnen Zeichen eines Datenwortes nach ihrer Übersetzung durch den Sender parallel über die einzelnen Kanäle des Übertragungskanals weiterzuleiten.

Das erweiterte Grundmodell der Kommunikation wird im folgenden für alle informationstheoretischen Aussagen und Beweise verwendet.

Information

Das Wort Information wird in der Informationstheorie in einem besonderen Sinn verwendet, der nicht mit dem gewöhnlichen Gebrauch verwechselt werden darf. Information ist ein Maß für die Freiheit der Wahl, wenn eine Nachricht aus einer Nachrichtenquelle ausgesucht wird [SHA76].

Der Zeichenvorrat einer diskreten Nachrichtenquelle wird hier als endlich angenommen und ist gegeben durch $X = \{x_1, x_2, \#\#\#, x_n\}$. Die Auswahl der Zeichen wird durch einen Zufallsprozeß beschrieben, wobei $P(x_i)$ die Wahrscheinlichkeit angibt, daß gerade das Zeichen x_i ausgewählt und gesendet wurde. Dabei gilt

$$\sum_{i=1}^{n} P(x_i) = 1 \quad \wedge \quad P(x_i) \neq 0 \quad \forall x_i \in X, \tag{2-1}$$

so daß mit Wahrscheinlichkeit 1 irgendein Zeichen der diskreten Nachrichtenquelle gesendet wird. Die Zeichenauswahl wird in meiner Diplomarbeit stets durch einen stationären Zufallsprozeß beschrieben, so daß die im folgenden eingeführten Zufallsgrößen X nicht abhängig von einem Zeit-Parameter t sind.

Die oben definierte Nachrichtenquelle wird hier als eine diskrete Zufallsgröße X mit Wertemenge $W(X) = \{x_1, x_2, \#\#\#, x_n\}$ und Wahrscheinlichkeitsfunktion $P(x_i) = P(X = x_i)$ betrachtet.

Ist der Zeichenvorrat einer Nachrichtenquelle durch ein Intervall oder durch alle reellen Zahlen ### gegeben, dann wird diese kontinuierliche Nachrichtenquelle durch eine stetige Zufallsgröße X mit Wertebereich W(X) ### ### und Wahrscheinlichkeitsdichte p(x) beschrieben.

Zur Vereinfachung wird im folgenden stets einer diskreten Darstellung Vorrang gegeben, wenn Beweise nicht von einer bestimmten Wahrscheinlichkeitsdichte abhängig sind.

Definition 2-1: Es sei eine Nachrichtenquelle X gegeben. Einem einzelnen Zeichen x_i, das mit der Wahrscheinlichkeit $P(x_i)$ auftritt, wird ein Informationsgehalt

$$I(x_i) = \ln \frac{1}{P(x_i)} = -\ln P(x_i) \qquad i = 1, 2, \ldots, n \tag{2-2}$$

zugeordnet. Demnach besitzen selten auftretende Zeichen einen großen Informationsgehalt und häufig auftretende Zeichen einen geringen Informationsgehalt. Durch diese Definition wird die Information zu einem Maß der Wahlfreiheit einer Nachrichtenquelle.

In meiner Diplomarbeit wird die Information in *natürlichen Einheiten* gemessen. Der somit verwendete natürliche Logarithmus[1] ln: ###$^{*}_{+}$ ### ### vereinfacht als Umkehrfunktion der Exponentialfunktion exp: ### ### ### häufig mathematische Ausdrücke in den Beweisen dieses Kapitels, da sich beide Funktionen gegenseitig aufheben.

Entropie

Die im letzten Abschnitt eingeführte Größe $I(x_i)$ beschreibt den Informationsgehalt eines einzelnen Zeichens x_i ### X. Damit charakterisiert diese Größe den Informationsgehalt einer kompletten Nachrichtenquelle X nicht ausreichend. Es erscheint also sinnvoll, den Erwartungswert von $I(x_i)$ zu verwenden.

Definition 2-2: Der mittlere Informationsgehalt einer Nachrichtenquelle X ist gegeben durch

$$H(X) := \sum_{i=1}^{n} P(x_i)\, I(x_i) = -\sum_{i=1}^{n} P(x_i)\, \ln P(x_i). \tag{2-3}$$

Die Größe H(X) wird auch als Entropie einer Nachrichtenquelle X bezeichnet. Mit Hilfe des Erwartungswertoperators ###.### läßt sich die Entropie in Kurzschreibweise auch darstellen als

$$H(X) := \langle -\ln P(x_i) \rangle. \tag{2-4}$$

Gilt $P(x_i) = (1/n)$ für i = 1, 2, …, n, so ergibt sich

$$H(X) = \sum_{i=1}^{n} \frac{1}{n} \ln n = \ln n =: H_0. \tag{2-5}$$

Den mittleren Informationsgehalt H_0 einer Nachrichtenquelle X bei gleichwahrscheinlichen Zeichen x_i bezeichnet man als Entscheidungsgehalt. Nach [MIL90] gilt $H(X) \le H_0$.

Definition 2-3: Die Differenz von Entscheidungsgehalt und Entropie

$$R := H_0 - H \tag{2-6}$$

wird als Redundanz bezeichnet. Wird die Redundanz auf den Entscheidungsgehalt bezogen, so ergibt sich die relative Redundanz

[1] Die verschiednen Logarithmen ln, lg und ld sind proportional zueinander. So gilt ln x = (ln 2) (ld x) und lg x = (lg 2) (ld x). Die Einheit für den dualen Logarithmus ld ist das Bit.

$$R_{rel} := \frac{R}{H_0} = 1 - \frac{H}{H_0}. \tag{2-7}$$

Aus den obigen Definitionen läßt sich schließen, daß eine Information nur in einer Nachricht enthalten ist, wenn diese unterschiedlich ausfallen kann, so daß nach dem Erhalt der Nachricht eine zunächst noch vorhandene Ungewißheit beseitigt wird. Die Entropie kann als ein Maß für die Ungewißheit einer Nachrichtenquelle angesehen werden. Das Auftreten von H_0 bei gleichwahrscheinlichen Zeichen ist dann verständlich, da in diesem Fall die größte Ungewißheit über das von der Nachrichtenquelle ausgewählte Zeichen besteht [MIL90].

Bisher wurde die Entropie nur für diskrete Nachrichtenquellen definiert. Auch wenn ich die folgende Definition als nicht ganz unproblematisch ansehe, wird sie im folgenden stets bei kontinuierlichen Nachrichtenquellen verwendet.

Definition 2-4: Die Entropie einer kontinuierlichen Nachrichtenquelle X mit Wahrscheinlicheitsdichte p(x) ist gegeben durch

$$H(X) := \int_{-\infty}^{\infty} p(x) \cdot \log p(x)\, dx. \tag{2-8}$$

Da für eine kontinuierliche Nachrichtenquelle mit einem Intervall oder $\Re$ als Wertebereich unendlich viele verschiedene Zeichen existieren, die zu jedem beliebigen Zeitpunkt ausgewählt und gesendet werden können, sollte auch deren Informationsgehalt unendlich groß sein.

Die mathematische Herleitung der Entropie in [MIL90] berücksichtigt diesen Sachverhalt, so daß dort die oben aufgeführte Entropie aus Gleichung 2-8 als *differentielle Entropie* bezeichnet wird. Da aber die Transinformation (siehe Seite 15) nur von der differentiellen Entropie abhängt [MIL90], wird im folgenden nicht zwischen *differentieller Entropie* und *Gesamtentropie* kontinuierlicher Nachrichtenquellen unterschieden.

2.1.2. Verbundquellen und lineare Transformationen

Im Abschnitt Grundmodelle der Kommunikation wurden zwei formale Modelle vorgestellt, welche die Grundprinzipien einer Kommunikation beschreiben. Damit eine Kommunikation stattfinden kann, werden Daten über einen Übertragungskanal von einer Nachrichtenquelle zu einem Nachrichtenziel übertragen.

In diesem Kapitel wird gezeigt, wie eine Kodierung beschaffen sein muß, damit die zu übertragenden Daten *vor* ihrer Übertragung einen möglichst großen Informationsgehalt besitzen. Dazu wurde das Kapitel in drei Abschnitte unterteilt, wobei die ersten beiden Abschnitte Zwischenergebnisse liefern, die im letzten Abschnitt für den Beweis der Hauptaussage benötigt werden. Bei dieser Hauptaussage handelt es sich um die erste von zwei zentralen informationstheoretischen Aussagen, die in Kapitel 2.1. formuliert und bewiesen werden.

Die Kodierung der Daten soll hier unter Berücksichtigung der Kapitel 2.2. und 2.3. durch eine lineare Transformation erfolgen, welche eine Menge von Eingabedaten auf eine Menge von Ausgabedaten abbildet. Die einzelnen Elemente der Eingabe- und Ausgabedaten seien durch Vektoren gegeben.

Um die lineare Transformation informationstheoretisch untersuchen zu können, wird im folgenden angenommen, daß sowohl die Vektoren der Eingabedaten als auch die Vektoren der Ausgabedaten von Nachrichtenquellen erzeugt werden. Jede Komponente eines Vektors entspricht einem einzelnen Zeichen einer Nachricht. Da Vektoren im allgemeinen mehrere Komponenten besitzen, muß eine Nachrichtenquelle in der Lage sein Nachrichten zu erzeugen, die eine Menge von Zeichen enthalten. In diesem Fall wird die Nachrichtenquelle zu einer Verbundquelle und bei den Nachrichten handelt es sich um Datenworte. Das Schema einer solchen Modellierung wird in der folgenden Abbildung dargestellt.

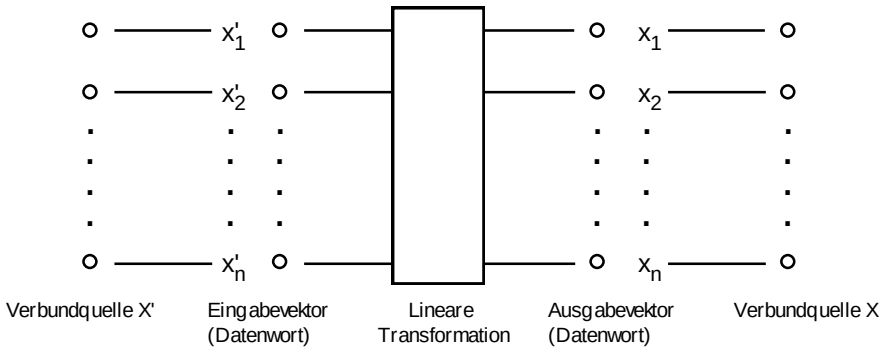


Abbildung 2-3 Modellierung der Ein- und Ausgabemenge einer linearen Transformation durch Verbundquellen

Die Ausgabedaten der linearen Transformation werden gemäß Abbildung 2-2 über einen Übertragungskanal übertragen, der sich aus mehreren einzelnen Kanälen zusammensetzt, wobei die einzelnen Zeichen eines Datenwortes (und damit die einzelnen Komponenten eines Vektors) parallel über die einzelnen Kanäle des Übertragungskanals weitergeleitet werden. Wie sich Störungen auf diese Form von Übertragung auswirken, wird in Kapitel 2.1.3. untersucht.

Im folgenden wird hergeleitet, wie die lineare Transformation beschaffen sein muß, damit die Vektoren ihrer Ausgabedaten einen möglichst großen Informationsgehalt besitzen. Für diese Herleitung werden einige Sätze und Korollare bewiesen, die auch in [SHA76] zu finden sind. Leider waren die Beweise dort nicht ohne weiteres für mich nachvollziehbar. Außerdem fand ich in der Literatur keine Untersuchungen über Verbundquellen, die den allgemeinen Fall von Datenworten mit *n* Zeichen explizit behandeln. Ich habe das Modell der Verbundquellen aus der Literatur [MIL90] entnommen, um lineare Transformationen informationstheoretisch untersuchen zu können. Dort beschränkte sich der Autor auf den Fall von Datenworten mit zwei Zeichen und stellte die unbewiesene Behauptung auf, daß dieser Fall verallgemeinert werden kann.

Damit war ich nicht einverstanden. Deshalb entnahm ich grundlegende Definitionen der Wahrscheinlichkeitsrechnung aus [PAP91] sowie [TOP74] und verallgemeinerte die Aussagen von [MIL90] über Verbundquellen für meine Diplomarbeit.

Die Menge der Ausgabedaten einer linearen Transformation wird aus informationstheoretischer Sicht durch eine Verbundquelle und jeder Vektor durch ein Datenwort beschrieben. Um den mittleren Informationsgehalt der Datenworte bestimmen zu können, wird im folgenden die Entropie von Verbundquellen untersucht. Dabei werden nicht nur Wahrscheinlichkeiten berücksichtigt, mit denen bestimmte Datenworte auftreten, sondern auch stochastische Abhängigkeiten zwischen den einzelnen Zeichen der Datenworte.

In meiner Diplomarbeit soll eine Verbundquelle als eine Nachrichtenquelle aufgefaßt werden, die aus mehreren Teilquellen besteht. Die folgende Abbildung zeigt eine Verbundquelle X, die sich aus n Teilquellen X^1, X^2, ###, X^n zusammensetzt.

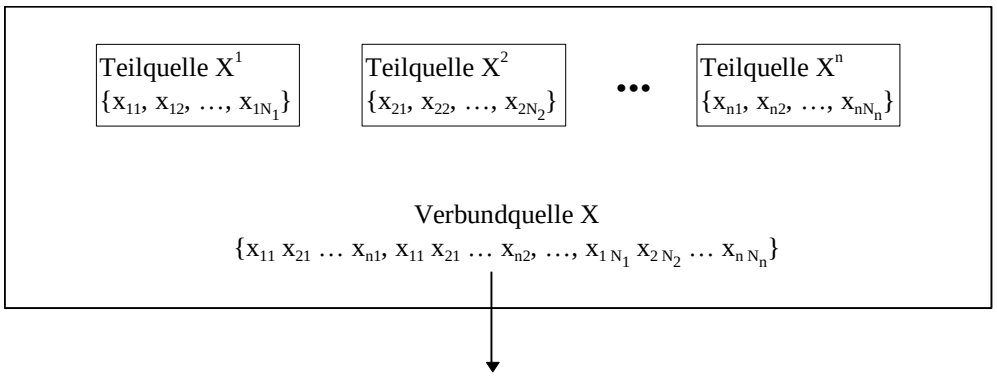


Abbildung 2-4 Modell einer aus n Teilquellen bestehenden Verbundquelle

Der Zeichenvorrat der Verbundquelle X besteht besteht aus allen N_1 ### N_2 ######### N_n Zeichenkombinationen der Teilquellen X^1, X^2, ###, X^n. Dabei produziert die Verbundquelle X immer ein komplettes Wort der Breite n zu einem Zeitpunkt. Jedes einzelne Zeichen x_{ij} eines Wortes wird von einer einzelnen Teilquelle X^i erzeugt, wobei alle Teilquellen unterschiedliche Zeichenvorräte und Wahrscheinlichkeitsfunktionen besitzen können. Somit handelt es sich bei jeder Teilquelle um eine vollwertige Nachrichtenquelle, für die auch die Bedingungen aus Gleichung 2-1 gelten. Die Entropie der Verbundquelle X ergibt sich mit Hilfe von Gleichung 2-3 zu

$$H(X) = H(X^1, X^2, \ldots, X^n)$$

$$= -\sum_{i=1}^{N_1}\sum_{j=1}^{N_2}\cdots\sum_{k=1}^{N_n} P(x_{1i}, x_{2j}, \ldots, x_{nk}) \cdot \log P(x_{1i}, x_{2j}, \ldots, x_{nk}) \qquad (2\text{-}9)$$

wobei $P(x_{1i}, x_{2j}, \ldots, x_{nk})$ die Wahrscheinlichkeit angibt, daß das Wort $x_{1i}\ x_{2j}$ ### x_{nk} ausgewählt und gesendet wurde.

Mit

$$P(x_{1i}, x_{2j}, \ldots, x_{nk}) = P(X^1 = x_{1i} \wedge X^2 = x_{2j} \wedge \ldots \wedge X^n = x_{nk}) \qquad (2\text{-}10)$$

läßt sich die Funktion P(.) als gemeinsame Wahrscheinlichkeitsfunktion der Teilquellen X^1, X^2, ###, X^n darstellen.

In den folgenden beiden Abschnitten wird untersucht, wie sich stochastische Abhängigkeiten zwischen den Teilquellen auf die Entropie der Verbundquelle auswirken. Damit soll gezeigt werden, ob bei stochastischer Unabhängigkeit der Informationsgehalt der Ausgabedaten einer linearen Transformation größer ist als bei stochastischer Abhängigkeit.

Unabhängige Teilquellen

Dieser Abschnitt beschäftigt sich mit einer Verbundquelle X, welche sich aus n stochastisch unabhängigen Teilquellen X^1, X^2, ###, X^n zusammensetzt. Dies entspricht nach dem oben eingeführten Modell einer linearen Transformation, deren Ausgabemenge nur aus Vektoren mit stochastisch unabhängigen Komponenten besteht. Wie groß ist in diesem Fall der mittlere Informationsgehalt dieser Ausgabemenge ?

Auf diese Frage gibt das folgende Korollar eine Antwort, indem die Entropie der Verbundquelle X berechnet wird. Für den Beweis des Korollars wird die gemeinsame Wahrscheinlichkeitsfunktion der Teilquellen X^1, X^2, ###, X^n aus Gleichung 2-10 im Falle stochastischer Unabhängigkeit benötigt. Nach [PAP91] läßt sich diese schreiben als

$$P(x_{1i}, x_{2j}, \ldots, x_{nk}) = P(x_{1i}) \cdot P(x_{2j}) \cdot \ldots \cdot P(x_{nk}) \quad . \tag{2-11}$$

Ich habe dieses Korollar in meine Diplomarbeit aufgenommen, um explizit den Fall von Verbundquellen mit *n* Teilquellen zu untersuchen und damit die Aussagen von [MIL90] zu verallgemeinern.

Korollar 2-1: Es sei X eine Verbundquelle, welche sich aus n stochastisch unabhängigen Teilquellen X^1, X^2, ###, X^n zusammensetzt. Dann gilt für die Entropie von X

$$H(X^1, X^2, \ldots, X^n) = H(X^1) + H(X^2) + \ldots + H(X^n)$$

Beweis 2-1: (siehe Anhang 1)

Die Entropie einer Verbundquelle mit stochastisch unabhängigen Teilquellen läßt sich also als Summe der Entropien der einzelnen Teilquellen darstellen.

Abhängige Teilquellen

Nachdem im letzten Abschnitt gezeigt wurde, welche Beziehungen für die Entropie einer Verbundquelle X mit stochastisch unabhängigen Teilquellen X^1, X^2, ###, X^n gelten, beschäftigt sich dieser Abschnitt mit stochastisch abhängigen Teilquellen. Dies entspricht dem Fall, wo eine lineare Transformation eine Ausgabemenge besitzt, die Vektoren mit stochastisch abhängigen Komponenten enthält. Es stellt sich auch hier die Frage, wie groß der mittlere Informationsgehalt dieser Ausgabemenge ist. Wie bereits im letzten Abschnitt wird diese Frage durch ein Korollar beantwortet.

Für den Beweis des Korollars wird die gemeinsame Wahrscheinlichkeitsfunktion der Teilquellen X^1, X^2, ###, X^n aus Gleichung 2-10 im Falle stochastischer Abhängigkeit benötigt. Diese Funktion ist nach [PAP91] gegeben durch

$$P(x_{1i}, x_{2j}, \ldots, x_{nk}) = P(x_{1i}) \cdot P_{x_{1i}}(x_{2j}) \cdot \ldots \cdot P_{x_{1i}, x_{2j}, \ldots, x_{(n-1)l}}(x_{nk}) \quad , \tag{2-12}$$

wobei es sich bei $P_{x_{1i}, x_{2j}, \#\#\#, x_{(n-1)l}}(x_{nk})$ um eine bedingte Wahrscheinlichkeit handelt. Sie gibt die Wahrscheinlichkeit für das Auftreten des Zeichens x_{nk} an, unter der Bedingung, daß zuvor die Zeichen x_{1i}, x_{2j}, ###, $x_{(n-1)l}$ auftraten.

Zusätzlich zur Wahrscheinlichkeitsfunktion wird für den Beweis auch die bedingte Entropie $H_{X^1, X^2, \#\#\#, X^{n-1}}(X^n)$ benötigt. In diesem Zusammenhang beschreibt $H_{x_{1i}, x_{2j}, \#\#\#, x_{(n-1)l}}(X^n)$ den mittleren Informationsgehalt der Teilquelle X^n unter der Bedingung, daß die Teilquellen X^1, X^2, ###, X^{n-1} die Zeichen x_{1i}, x_{2j}, ###, $x_{(n-1)l}$ ausgewählt haben.

Nach [PAP91] und gemäß Gleichung 2-3 wird $H_{x_{1i}, x_{2j}, \#\#\#, x_{(n-1)l}}(X^n)$ definiert als

$$H_{x_{1i},x_{2j},\ldots,x_{nl}}(X^n) = -\sum_{k=1}^{N_n} P_{x_{1i},x_{2j},\ldots,x_{nl}}(x_{ik}) \log_2 P_{x_{1i},x_{2j},\ldots,x_{nl}}(x_{ik}). \qquad (2\text{-}13)$$

Die bedingte Entropie $H_{X^1, X^2, \#\#\#, X^{n-1}}(X^n)$ erhält man dann durch den Erwartungswert

$$H_{X^1,X^2,\ldots,X^{n-1}}(X^n) = \left\langle H_{x_{1i},x_{2j},\ldots,x_{nl}}(X^n) \right\rangle_{i,j,\ldots,l}$$

$$= \sum_{i=1}^{N_1}\sum_{j=1}^{N_2}\cdots\sum_{l=1}^{N_{n-1}} P_{x_{1i},x_{2j},\ldots,x_{nl}}(x_{1i},x_{2j},\ldots,x_{nl}) \cdot H_{x_{1i},x_{2j},\ldots,x_{nl}}(X^n)$$

$$= \sum_{i=1}^{N_1}\cdots\sum_{l=1}^{N_{n-1}} P_{x_{1i},\ldots,x_{nl}}(x_{1i},\ldots,x_{nl}) \cdot \sum_{k=1}^{N_n} -P_{x_{1i},\ldots,x_{nl}}(x_{ik}) \log_2 P_{x_{1i},\ldots,x_{nl}}(x_{ik}). \qquad (2\text{-}14)$$

Diese durchaus nicht trivialen Definitionen dienen nur als Grundlage für den Beweis des Korollars, welches wiederum die Aussagen von [MIL90] verallgemeinert und Verbundquellen mit *n* Teilquellen im Falle stochastischer Abhängigkeit untersucht.

Korollar 2-2: Es sei X eine Verbundquelle, die aus n stochastisch abhängigen Teilquellen X^1, X^2, ###, X^n besteht. Dann gilt für die Entropie von X

$$H(X) = H(X^1,X^2,\ldots,X^n) = H(X^1) + H_{X^1}(X^2) + \ldots + H_{X^1,X^2,\ldots X^{n-1}}(X^n).$$

Beweis 2-2: (siehe Anhang 1)

In welcher Beziehung stehen die Entropien von Verbundquellen mit stochastisch unabhängigen oder abhängigen Teilquellen ? Durch eine Beantwortung dieser Frage könnte gezeigt werden, ob es vorteilhaft ist, wenn die Vektoren der Ausgabemenge einer linearen Transformation stochastisch unabhängige Komponenten besitzen. Daher beschäftigt sich der folgende Abschnitt mit dieser Thematik.

Lineare Transformationen und unabhängige Teilquellen

In den letzten beiden Abschnitten wurde berechnet, wie groß die Entropie bei Verbundquellen mit stochastisch unabhängigen und abhängigen Teilquellen ist. In diesem Abschnitt wird mit Hilfe eines mathematischen Satzes gezeigt, daß die Entropie bei stochastischer Unabhängigkeit größer ist als bei stochastischer Abhängigkeit. Was bedeutet diese informationstheoretische Aussage für die linearen Transformationen ?

Wie in diesem Kapitel bereits angedeutet, werden die Vektoren der Ausgabemenge einer linearen Transformation als Datenworte betrachtet, die von einer Verbundquelle gemäß einer Wahrscheinlichkeitsfunktion ausgewählt und gesendet werden. Jede Komponente der Vektoren wird damit von einer einzelnen Teilquelle ausgewählt und gesendet.

Eine lineare Transformation, deren Ausgabemenge so beschaffen ist, daß die Komponenten der Vektoren stochastisch unabhängig sind, entspricht damit informationstheoretisch einer Verbundquelle, deren Entropie durch die Summe der Entropien der einzelnen Teilquellen gegeben ist (siehe Korollar 2-1). Nach [SHA76] und [MIL90] nimmt dann und nur dann die Entropie ihren Maximalwert an.

Für den Beweis des mathematischen Satzes wird auch der Begriff der *Transinformation* benötigt, welche aus diesem Grund mit Hilfe des folgenden Modells vorgestellt wird.

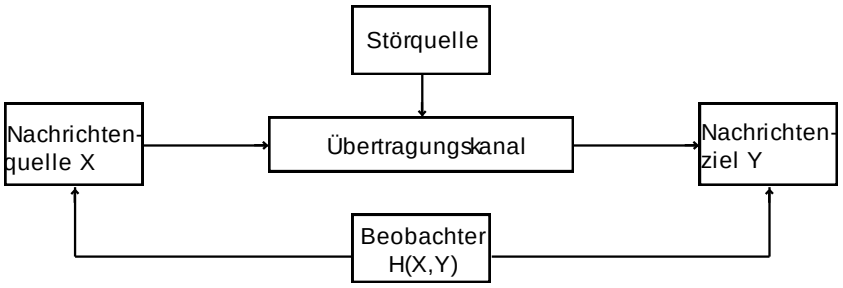


Abbildung 2-5 Modell der Informationsübertragung mit einem fiktiven Beobachter

In diesem Modell überträgt ein Übertragungskanal diskrete Nachrichten von einer Nachrichtenquelle X zu einem Nachrichtenziel Y. Zusätzlich ist ein fiktiver Beobachter vorhanden, für den sowohl Nachrichtenquelle als auch Nachrichtenziel Informationsquellen darstellen. Damit handelt es sich für den Beobachter bei diesem Modell um eine Verbundquelle mit den Teilquellen X und Y sowie der Entropie H(X,Y).

Die Transinformation H(X;Y) charakterisiert nun den Teil der Quellenentropie H(X), der beim Nachrichtenziel Y angekommen ist und nicht durch Störungen verloren ging. Bei völlig störungsfreier Übertragung gilt also H(X;Y) = H(X) = H(Y).

Der andere Grenzfall tritt bei total gestörter Übertragung auf, dann gilt H(X;Y) = 0. In diesem Fall liegen für den Beobachter zwei voneinander unabhängige Teilquellen vor und es gilt H(X,Y) = H(X) + H(Y). Nach [MIL90] ist die Transinformation gegeben durch

$$H(X;Y) = H(X) + H(Y) - H(X,Y) \quad . \tag{2-15}$$

Diese Form der Transinformation ist für den Beweis des folgenden Satzes noch ungeeignet, weil sie eine Nachrichtenquelle und ein Nachrichtenziel voraussetzt. Da in diesem Kapitel Verbundquellen mit n Teilquellen betrachtet werden, wird im folgenden die verallgemeinerte Form der Transinformation

$$H(X^1;X^2;\ldots;X^n) = H(X^1) + H(X^2) + \ldots + H(X^n) - H(X^1,X^2,\ldots,X^n) \tag{2-16}$$

verwendet, wobei die Transinformation dann als jene Information angesehen wird, die alle Teilquellen X^1, X^2, …, X^n gemeinsam besitzen. Nach [PAP91] wird diese Information als *mutual information* (Austauschinformation) bezeichnet.

Der Beweis des folgenden Satzes beschäftigte mich längere Zeit. So gab es mehrere Versionen des Beweises, die mich aber alle nicht zufriedenstellen konnten. Daraufhin entschloß ich mich, stärkere informationstheoretische Aussagen aus der Literatur zu verwenden und damit den Beweis argumentativ zu führen.

Satz 2-1: Es sei X eine Verbundquelle, die aus n Teilquellen X^1, X^2, …, X^n besteht. Dann gilt die folgende Beziehung

$$H(X^1) + H_{X^1}(X^2) + \ldots + H_{X^1,X^2,\ldots X^{n-1}}(X^n) \le H(X^1) + H(X^2) + \ldots + H(X^n),$$

wobei Gleichheit dann und nur dann gilt, wenn die Teilquellen X^1, X^2, …, X^n stochastisch unabhängig sind.

Beweis 2-1: Es genügt zu zeigen

$$H(X^1) + H(X^2) + \ldots + H(X^n) - H(X^1) - H_{X^1}(X^2) - \ldots - H_{X^1,X^2,\ldots X^{n-1}}(X^n) \ge 0.$$

Nach [SHA76] und [PAP91] gilt die Beziehung

$$H(X^1, X^2, \ldots, X^n) \le H(X^1) + H(X^2) + \ldots + H(X^n)$$

mit Gleichheit genau dann, wenn die Teilquellen stochastisch unabhängig sind. Daraus ergibt sich für die Transinformation aus Gleichung 2-16

$$H(X^1; X^2; \ldots; X^n) \ge 0,$$

wobei Gleichheit auch hier dann und nur dann gilt, wenn die zugrundeliegenden Teilquellen stochastisch unabhängig sind. Für den Fall stochastisch abhängiger Teilquellen X^1, X^2, …, X^n erhält man nach Gleichung 2-16 und Korollar 2-2 für die Transinformation

$$0 \le H(X^1; X^2; \ldots; X^n)$$

$$= H(X^1) + H(X^2) + \ldots + H(X^n) - H(X^1) - H_{X^1}(X^2) - \ldots - H_{X^1,X^2,\ldots X^{n-1}}(X^n)$$

Daraus folgt bereits die Behauptung, denn die zu beweisende Ungleichung entspricht genau der Transinfomation $H(X^1;X^2;\ldots;X^n)$ im Falle stochastisch abhängiger Teilquellen.

q.e.d.

Bereits intuitiv ist der obige Sachverhalt klar, denn auch die Entropie einer Verbundquelle ist ein Maß für die Ungewißheit einer Nachrichtenquelle. Abhängigkeiten schränken die Wahlfreiheit ein und verringern die Ungewißheit (und damit die Entropie) der Verbundquelle.

Als ein wichtiges Ergebnis dieses Abschnitts kann man festhalten, daß die Ausgabedaten einer linearen Transformation durch eine Menge von Vektoren mit stochastisch unabhängigen Komponenten gegeben sein sollten. Ist eine lineare Transformation so beschaffen, daß sie diese Bedingung erfüllt, dann besitzen ihre Ausgabedaten einen maximalen mittleren Informationsgehalt.

2.1.3. Informationsoptimale Übertragung gestörter Daten

Bisher beschäftigte sich das Kapitel 2.1. mit der Transformation von Daten *bevor* diese über einen Übertragungskanal gesendet werden. Im Abschnitt Lineare Transformationen und unabhängige Teilquellen wurde mit Hilfe von Satz 2-1 gezeigt, daß die Ausgabedaten der Transformation durch Vektoren mit stochastisch unabhängigen Komponenten gegeben sein sollten.

Bezieht man diese Aussage auf das erweiterte Grundmodell der Kommunikation in Abbildung 2-2, dann sollte die dort dargestellte Nachrichtenquelle Datenworte auswählen und versenden, deren einzelne Zeichen stochastisch unababhängig voneinander sind. Die bisher durchgeführten Untersuchungen behandeln also nur die zugrundeliegende Nachrichtenquelle einer Kommunikation.

In diesem Kapitel wird die Übertragung von transformierten Daten über einen gestörten Kanal untersucht. Es ist das Ziel dieses Kapitels eine lineare Transformation anzugeben, welche die zu übertragenden Daten derart an den Übertragungskanal und dessen Störungen anpaßt, daß ein Maximum an Information von der Nachrichtenquelle zum Nachrichtenziel übertragen wird. Das Grundprinzip Daten an einen Übertragungskanal und dessen Störungen anzupassen, um ein Maximum an Information zu übertragen, stammt aus der Informationstheorie. In [MIL90] wird diese Vorgehensweise als "Anpassung der Quelle an den Kanal" bezeichnet.

Bereits 1949 beschäftigte sich C. E. Shannon in [SHA49] mit der Übertragung von Signalen über einen gestörten Kanal. Dort gab er ein informationstheoretisches Modell für die Signalübertragung an und untersuchte es mit Mitteln der Signaltheorie. Unter Berücksichtigung gewisser physikalischer Gegebenheiten leitete er eine einfache Formel für die Kanalkapazität (siehe folgenden Abschnitt) her, welche nur noch von einem Signal-Rauschabstand abhängig ist. Diese Herleitung war auch für mich sehr interessant, denn es wurde ein Modell angegeben, mit dem die Übertragung von Signalen über einen gestörten Kanal untersucht werden kann. Jedoch war es für mich nicht möglich mit Hilfe von Shannon's Formel herzuleiten, welche Bedingungen die zu übertragenden Signale erfüllen müssen, um ein Maximum an Information über den gestörten Kanal weiterzuleiten. Somit konnte ich auch keine lineare Transformation angeben, welche die Daten an den gestörten Kanal anpaßt.

Meine eigenen Untersuchungen beruhen auf der Vermutung, daß sich die zu übertragenden Signale sehr deutlich von den Störungen abheben müssen. Für das erweiterte Grundmodell der Kommunikation aus Abbildung 2-2 bedeutet dies, daß die Aktivitäten der Signale in jedem einzelnen Kanal des Übertragungskanals so weit verstärkt werden müssen, bis jeder einzelne Kanal voll ausgelastet wird. Neben einer vollen Auslastung der einzelnen Kanäle bewirkt die Verstärkung auch, daß der Informationsgehalt von Signalen mit geringen Aktivitäten nicht gänzlich verloren geht. Das Signal in jedem Kanal des Übertragungskanals hebt sich dann soweit wie möglich von den Störungen ab, so daß dessen Informationsgehalt nur minimal verfälscht wird. Setzt man voraus, daß es sich um gleichartige Kanäle handelt, dann werden die Aktivitäten der einzelnen Signale auf dieselbe Größe normiert.

Für die im folgenden durchgeführten Untersuchungen mußte ich einen sehr langen Weg durch die Informationstheorie und Signaltheorie zurücklegen.

Meinem Ziel eine lineare Transformation anzugeben, welche die zu übertragenden Daten unter Berücksichtigung der oben beschriebenen Überlegungen derart an einen gestörten Kanal anpaßt, daß ein Maximum an Information übertragen wird, kam ich durch den Bericht [PLU93] ein ganzes Stück näher.

In diesem Bericht beschäftigte sich Plumbley ebenfalls mit der Übertragung von Signalen über einen gestörten Übertragungskanal, welcher sich aus mehreren unabhängig arbeitenden einzelnen Kanälen zusammensetzt. Aus der Bedingung ein Maximum an Information zu übertragen, leitet Plumbley mehrere Lernverfahren für Neuronale Netze her, die nach ihrer Lernphase eine lineare Transformation durchführen können. Diese Transformation paßt eine Menge von Daten so an den gestörten Kanal an, daß ein Maximum an Information weitergeleitet werden kann.

Leider waren die mathematischen Betrachtungen in [PLU93] sehr unvollständig und teilweise schwer nachvollziehbar. Daher entschloß ich mich in eigenständiger Arbeit ein informationstheoretisches Modell für die Übertragung von Daten über einen gestörten Kanal auf Basis der Verbundquellen aus Kapitel 2.1.2. zu erstellen und für dieses Modell eine mathematische Herleitung anzugeben, welche zeigt, daß eine bestimmte Form von Transformation eine Menge von Daten derart an den gestörten Übertragungskanal anpassen kann, daß ein Maximum an Information von einer Nachrichtenquelle zu einem Nachrichtenziel übertragen wird.

Als Ergebnis liefert diese Herleitung mehrere Korollare und Sätze, durch welche die zweite zentrale informationstheoretische Aussage des Kapitels 2.1. formuliert wird. Bevor ein konkretes Modell und eine Transformation im Abschnitt Die maximal erreichbare Transinformation angegeben und untersucht werden, müssen noch einige informationstheoretische Grundbegriffe eingeführt werden, die für das Verständnis der Thematik wichtig sind.

Kanalkapazität

In diesem Kapitel wurde bisher mehrmals erwähnt, daß ein Maximum an Information von einer Nachrichtenquelle zu einem Nachrichtenziel übertragen werden soll. Es wurde aber bisher keine formale Definition dieses Maximums angegeben. Daher wird in diesem Abschnitt am Beispiel des folgenden Modells eine solche Definition hergeleitet.

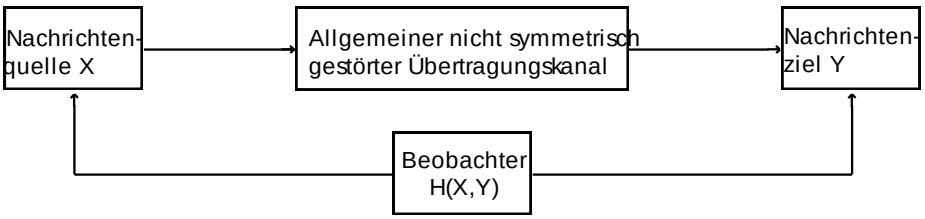


Abbildung 2-6 Modell eines nicht symmetrisch gestörten Kanals mit einem fiktiven Beobachter

In diesem Modell wird die Übertragung von Nachrichten über einen gestörten Kanal beschrieben, wobei ein fiktiver Beobachter die Nachrichtenquelle und das Nachrichtenziel als Informationsquellen betrachtet. Diese beiden Informationsquellen behandelt der Beobachter als eine Verbundquelle mit den Teilquellen X und Y sowie der Entropie H(X,Y).

Bei dem oben dargestellten allgemeinen nicht symmetrisch gestörten Übertragungskanal werden Störungen bei der Übertragung von Informationen durch eine Rauschmatrix beschrieben. Diese Matrix gibt für jedes Zeichen der Nachrichtenquelle X eine Wahrscheinlichkeit für dessen Störung an, wobei diese Wahrscheinlichkeiten bei einem *nicht* symmetrisch gestörten Kanal unterschiedlich sind.

Welche Größe muß nun im obigen Modell maximiert werden, um ein Maximum an Information von der Nachrichtenquelle zum Nachrichtenziel zu übertragen ?

Eine Antwort auf diese Frage wurde indirekt schon im Abschnitt Lineare Transformationen und unabhängige Teilquellen gegeben. Dort wurde die *Transinformation* als eine Größe beschrieben, die den Teil der Quellenentropie H(X) charakterisiert, der beim Nachrichtenziel Y ankommt und nicht durch Störungen verloren geht. Nach [MIL90] muß die Nachrichtenquelle X derart an den gestörten Übertragungskanal angepaßt werden, damit die Transinformation, also die tatsächlich übertragene Information, maximal wird.

Die Anpassung erfolgt am Beispiel des obigen Modells in der Tendenz so, daß weniger stark gestörte Zeichen mit größeren Wahrscheinlichkeiten auftreten und häufig gestörte Zeichen mit kleineren Wahrscheinlichkeiten.

Es zeigt sich also, daß durch die Anpassung der Nachrichtenquelle ein gestörter Übertragungskanal so betrieben werden kann, daß ein Maximum an Informationen von der Nachrichtenquelle zum Nachrichtenziel übertragen wird. Dieser Maximalwert $H(X;Y)_{max}$ wird als Kanalkapazität bezeichnet

$$C := H(X;Y)_{max} \tag{2-17}$$

Es ist nun das Ziel im Abschnitt Die maximal erreichbare Transinformation für ein informationstheoretisches Modell eines gestörten Kanals eine Transformation anzugeben, so daß bei einer Übertragung der transformierten Daten die Kanalkapazität erreicht wird.

Kontinuierliche Nachrichtenquellen

Die bisher behandelten Nachrichtenquellen verfügten über einen diskreten Zeichenvorrat aus dem gemäß einer Wahrscheinlichkeitsfunktion ein bestimmtes Zeichen ausgewählt und gesendet wurde.

Da die Betrachtungen im nachfolgenden Abschnitt von Wahrscheinlichkeitsdichten abhängig sind, werden in diesem Abschnitt kontinuierliche Nachrichtenquellen kurz vorgestellt.

Die Nachrichten einer kontinuierlichen Informationsquelle werden nicht durch diskrete Zeichenmengen, sondern durch eine zeit- und wertekontinuierliche Funktion beschrieben. Um kontinuierliche Nachrichtenquellen informationstheoretisch untersuchen zu können, werden sie in diesem Kapitel als stetige Zufallsgrößen betrachtet. Dabei repräsentieren diese Zufallsgrößen stationäre Zufallsprozesse, die zur Modellierung von Zufallssignalen benötigt werden.

Definition 2-5: Es sei X eine stetige Zufallsgröße mit einer Dichtefunktion p(x). Dann ist der zweite Moment der Zufallsgröße gegeben durch

$$\left\langle X^2 \right\rangle = \int_{-\infty}^{\infty} x^2 \cdot p(x)\, dx = P_X . \tag{2-18}$$

Die Größe P_X wird hier als *mittlere Leistung* eines Zufallssignals bezeichnet. Falls X mittelwertfrei ist, entspricht P_X genau der Varianz σ_X^2. Dieses Definition von P_X stellt eine Vereinfachung dar, denn bei real existierenden physikalischen Signalen ist die mittlere Leistung P_X lediglich proportional zum zweiten Moment $\langle X^2 \rangle$.

Mit Hilfe dieser Definition können jetzt die Aktivitäten von Signalen (siehe Seite 18) formal beschrieben werden, denn sie entsprechen exakt der mittleren Leistung. In der Realität können Übertragungskanäle nur Signale mit einer begrenzten mittleren Leistung übertragen. Daher wird im folgenden für die zu übertragenden Signale eine mittlere Leistung so vorgegeben, daß der Übertragungskanal voll ausgelastet wird.

Die maximal erreichbare Transinformation

In diesem Abschnitt wird ein konkretes Modell und eine lineare Transformation für die Übertragung von transformierten Daten über einen gestörten Übertragungskanal angegeben. Anschließend wird mit Hilfe einiger Sätze und Korollare bewiesen, daß mit diesem informationstheoretischen Modell bei der Übertragung der transformierten Daten die Kanalkapazität erreicht wird.

Bei dem von mir entworfenen Modell handelt es sich um ein zweistufiges informationstheoretisches Modell, welches auf dem erweiterten Grundmodell der Kommunikation in Abbildung 2-2 basiert.

Alle Informationsquellen in diesem Modell werden durch Verbundquellen beschrieben und die Übertragung der Daten in der ersten und zweiten Stufe erfolgt in Form von Datenworten, deren einzelne Komponenten parallel über die einzelnen Kanäle des Übertragungskanals weitergeleitet werden.

Das zweistufige informationstheoretische Modell wird in der folgenden Abbildung schematisch dargestellt.

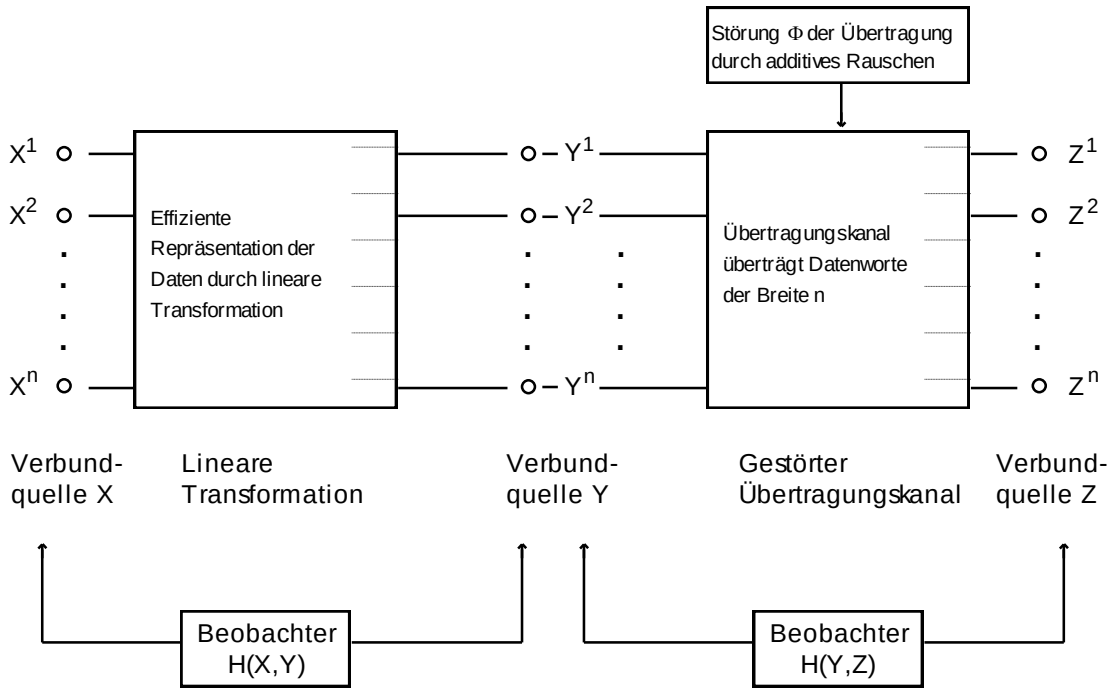


Abbildung 2-7 Modell der Informationsübertragung mit zwei fiktiven Beobachtern

In der ersten Stufe des Modells werden die Daten von einer Verbundquelle X zu einer Verbundquelle Y übertragen und dabei linear transformiert. Bei dieser Übertragung treten keinerlei Störungen auf. Die lineare Transformation ermöglicht eine effizientere Repräsentation der Daten durch eine Darstellung bezüglich einer neuen Basis. Der erste Beobachter betrachtet X und Y als eine einzige Verbundquelle mit den Teilquellen X und Y sowie der Entropie H(X,Y).

In der zweiten Stufe erfolgt eine Übertragung der linear transformierten Daten von der Verbundquelle Y zur Verbundquelle Z über einen gestörten Übertragungskanal. Für den zweiten Beobachter handelt es sich bei Y und Z um eine einzige Verbundquelle mit den Teilquellen Y und Z sowie der Entropie H(Y,Z).

Um das oben beschriebene Modell mit Mitteln der Informationstheorie und Stochastik untersuchen zu können, muß es stärker formalisiert werden. Daher werden an dieser Stelle alle Voraussetzungen genau beschrieben, welche für die nachfolgenden Untersuchungen erfüllt sein müssen.

- Bei den Größen X und Y handelt es sich, um kontinuierliche Verbundquellen, die Datenworte der Breite n gemäß einer mittelwertfreien n-dimensionalen normalen Dichtefunktion auswählen. Man kann daher X und Y auch als mittelwertfreie normalverteilte Zufallsgrößen betrachten, die zur Modellierung von Zufallssignalen benötigt werden.

- Wie schon in [SHA76] werden Störungen im Übertragungskanal durch ein mittelwertfreies normalverteiltes Zufallssignal Φ beschrieben, welches in der Literatur häufig auch als *Rauschen* bezeichnet wird. Weiterhin wird angenommen, daß Störungen das eigentliche zu übertragende Signal Y additiv überlagern. Somit ist das empfangene Signal Z durch $Z = Y + \Phi$ gegeben. Aus der Sicht der Stochastik ist die Zufallsgröße Z als Summe zweier normalverteilter Zufallsgrößen Y und Φ wiederum normalverteilt.

- Die Zufallsgrößen Y und Φ sind stochastisch unabhängig. Daraus ergibt sich für die aus der Stochastik bekannten Korrelationsmatrix von Z

$$C_{ZZ} = \left\langle z z^T \right\rangle = \left\langle \quad \right\rangle \ominus \left\langle y y^T + y \phi^T + \phi y^T + \phi \phi^T \right\rangle$$

$$= \left\langle y y^T \right\rangle + \underbrace{\left\langle y \phi^T \right\rangle}_{=0} + \underbrace{\left\langle \phi y^T \right\rangle}_{=0} + \left\langle \phi \phi^T \right\rangle = \left\langle y y^T \right\rangle + \left\langle \phi \phi^T \right\rangle$$

$$= C_{YY} + C_{\Phi\Phi}.$$

  Da Y und Φ stochastisch unabhängig sind, ist nach [PAP91] die Kovarianz der beiden Zufallsgrößen immer 0. Da Y und Φ zusätzlich auch mittelwertfrei sind, ergibt sich für deren Korrelationsmatrix

$$C_{Y\Phi}\ i, j = 0 \quad \wedge \quad C_{\Phi Y}\ i, j = 0 \qquad\qquad i, j = 1, 2, \ldots, n.$$

- Das Rauschen Φ besitzt die gleiche mittlere Leistung P_Φ in jeder der n Komponenten. Die Komponenten sind zusätzlich unkorreliert, so daß Störungen von einzelnen Zeichen aus einem Wort unabhängig voneinander erfolgen. Dadurch ist die Korrelationsmatrix von Φ gegeben durch

$$C_{\Phi\Phi} = P_\Phi I.$$

- Als Zufallssignal besitzt die Zufallsgröße Y eine vorgegebene mittlere Gesamtleistung. Die mittlere Leistung in einer Komponente i ist gegeben durch

$$P_{Y^i} = \left\langle y_i^2 \right\rangle \qquad\qquad i = 1, 2, \ldots, n$$

  und die mittlere Gesamtleistung ergibt sich dann zu

$$P_Y = \sum_{i=1}^{n} P_{Y^i} = \mathrm{Tr}\left(\quad \right).$$

Es ist durchaus sinnvoll die Zufallsgrößen Y und Φ (und damit auch Z) als normalverteilt anzunehmen.

Geht man nämlich von Zufallssignalen mit einer vorgegebenen mittleren Leistung aus, dann wird ein Maximum der Entropie für die Zufallssignale erreicht, wenn sie normalverteilt sind. Nach [SHA76] gilt das auch für Zufallssignale mit einer n-dimensionalen Dichtefunktion. Bei der Bestimmung der Transinformation H(Y;Z) für das obige informationstheoretische Modell erhält man mit dieser Annahme dann die Kanalkapazität.

An dieser Stelle wird nun die folgende Behauptung aufgestellt:

Werden die Daten der Verbundquelle X bei der Übertragung zur Verbundquelle Y in der ersten Stufe des Modells derart transformiert, daß sich die Korrelationsmatrix C_{YY} von Y darstellen läßt als

$$C_{YY} = \left\langle y\, y^T \right\rangle = \frac{P_Y}{n} E,$$

dann wird im oben beschriebenen informationstheoretischen Modell in der zweiten Stufe ein Maximum an Information über den gestörten Übertragungskanal von Y nach Z übertragen. In diesem Fall gilt für jede einzelne Komponente der Korrelationsmatrix C_{YY}

$$C_{YY}(i,j) = \left\langle y_i\, y_j \right\rangle = \begin{cases} 0 & i \neq j \quad \text{(Dekorrelation)} \\ \frac{P_Y}{n} & i = j \quad \text{(Normierung)} \end{cases} \qquad i, j = 1, 2, \ldots, n.$$

Die Daten der Verbundquelle Y sind dekorreliert und alle Komponenten der Datenworte von Y besitzen die gleiche normierte Varianz[2]. Diese Form von Transformation wird in der Literatur als *Datenorthonormalisierung*[3] bezeichnet.

Der Rest dieses Abschnitts beschäftigt sich mit dem Beweis der obigen Behauptung. Dazu wird gezeigt, wie die Menge der Ausgabedaten Y der linearen Transformation beschaffen sein muß, damit bei ihrer Übertragung von Y nach Z die Transinformation H(Y;Z), also die tatsächlich übertragene Information, maximal wird.

Im ersten Teil des Beweises werden die für die Transinformation

H(Y;Z) = H(Y) + H(Z) − H(Y,Z)

---

[2] Da Y als mittelwertfrei ($\langle y \rangle = 0$) vorausgesetzt wird, vereinfacht sich die Varianz σ_Y^2 von Y zu $\sigma_Y^2 = \langle (y - \langle y \rangle)^2 \rangle = \langle y^2 \rangle$. Für die Varianz einer Komponente i eines Datenwortes $y \in Y$ folgt dann $\sigma_{y_i}^2 = \langle y_i^2 \rangle = P_{Yi}$. Sie entspricht damit der mittleren Leistung von Y in der i-ten Komponente. Die mittlere Gesamtleistung von Y ist dann $P_Y = \sum_i^n P_{Yi}$.

[3] In der klassischen Sichtweise der Datenorthonormalisierung werden die Varianzen der einzelnen Komponenten auf 1 normiert. Die hier angegebene Definition stellt eine Verallgemeinerung dar, die für den Spezialfall $P_Y = n$ der klassischen Sichtweise entspricht.

notwendigen Terme H(Y), H(Z) und H(Y,Z) berechnet.

Da es sich bei X, Y und Z um n-dimensionale normalverteilte Zufallsgrößen handelt, wird die Entropie H(X) stellvertretend für die Größen H(Y) und H(Z) ermittelt.

Im zweiten Teil des Beweises wird dann gezeigt, wie die Daten der Verbundquelle Y beschaffen sein müssen, damit bei einer vorgegebenen mittleren Leistung P_Y die Transinformation H(Y;Z) einen globalen Maximalwert annimmt.

Korollar 2-3: Es sei X eine n-dimensional normalverteilte Zufallsgröße mit einer Dichtefunktion

$$p(x_1, x_2, \ldots x_n) = \frac{1}{\sqrt{(2\pi)^n \det C_{XX}}} e^{\underbrace{-\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} C_{XX}^{-1}(i,j)\, x_i x_j}_{=x^T C_{XX}^{-1} x}},$$

wobei die Korrelationsmatrix C_{XX} berechnet wird durch

$$C_{XX}(i,j) = \langle x_i x_j \rangle = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} x_i x_j\, p(x_1, x_2, \ldots, x_n)\, dx_1 dx_2 \ldots dx_n .$$

Dann ist die Entropie von X gegeben durch

$$H(X) = \ld \sqrt{(2\pi e)^n \det C_{XX}} .$$

Beweis 2-3: (siehe Anhang 1)

Zur Bestimmung der Transinformation H(Y;Z) wird auch die Entropie H(Y,Z) benötigt. Für den Beobachter auf der rechten Seite in Abbildung 2-7 scheint es egal zu sein, ob er die Verbundquellen (Y, Z) oder (Y, Φ) beobachtet, um vollständig informiert zu sein. Er bekommt in beiden Fällen Informationen über das Zufallssignal der Nachrichtenquelle Y und über das Rauschen der Störquelle Φ. Es liegt nun nahe zu behaupten, daß die Beziehung H(Y,Z) = H(Y,Φ) gilt. Diese Behauptung beweist das folgende Korollar.

Korollar 2-4: Es seien die Verbundquellen Y, Z und Φ wie im Modell in Abbildung 2-7 gegeben. Weiterhin seien alle geforderten Voraussetzungen für das Modell erfüllt. Dann gilt die Beziehung

$$H(Y,Z) = H(Y,\Phi) .$$

Beweis 2-4: (siehe Anhang 1)

Nachdem jetzt alle Größen zur Ermittlung der Transinformation H(Y;Z) bekannt sind, kann nun ein Satz formuliert werden, der als Ergebnis die Transinformation H(Y;Z) in einer sehr einfachen Form liefert, auf deren Basis anschließend die Hauptaussage dieses Abschnitts bewiesen werden kann.

Satz 2-2: Es seien die Verbundquellen Y, Z und Φ wie im Modell in Abbildung 2-7 gegeben und alle geforderten Voraussetzungen für dieses Modell seien erfüllt. Dann ist die Transinformation H(Y;Z) gegeben durch

$$H(Y;Z) = \frac{1}{2}\ln\det\left(\frac{1}{P_\Phi}C_{YY} + I\right).$$

Beweis 2-2: (siehe Anhang 1)

Da die Transinformation H(Y;Z) jetzt in einer sehr einfachen Form vorliegt, stellt sich die Frage, wie die Daten der Verbundquelle Y beschaffen sein müssen, damit bei einer vorgegebenen mittleren Leistung $P_Y = Tr(C_{YY})$ die Transinformation einen globalen Maximalwert annimmt.

An dieser Stelle erscheint es sinnvoll, die Funktion H(Y;Z) mit Hilfe der Methode der Lagrange-Multiplikatoren zu untersuchen. Diese Lösung wurde zunächst von mir in Betracht gezogen. Bereits nach kurzer Zeit stellte sich allerdings heraus, daß es sich bei diesem Lösungsweg um eine Fehlentwicklung handelte, denn mit Hilfe der Methode der Lagrange-Multiplikatoren lassen sich nur lokale Extrema einer Funktion unter Berücksichtigung von Nebenbedingungen finden. Es ist aus mathematischer Sicht sehr aufwendig (siehe [FIH73]) von einem lokalen Extremum unter Berücksichtigung von Nebenbedingungen zu zeigen, daß es sich hierbei um ein globales Maximum einer Funktion handelt.

Da auch argumentativ nicht geklärt werden konnte, ob es sich bei einem konkreten lokalen Extremum der Funktion H(Y;Z) um ein globales Maximum handelt, wurde dieser Lösungsweg verworfen. Ich fand schließlich mit Hilfe der Literatur [PAP91] den entscheidenden Hinweis, durch den das globale Maximum der Transinformation H(Y;Z) mit einer vorgegebenen mittleren Leistung P_Y gefunden werden konnte.

Dazu wird H(Y;Z) als eine Kostenfunktion

$$R(c_{11}, c_{12}, \ldots, c_{nn}) = \frac{1}{2}\ln\det\left(\frac{1}{P_\Phi}C_{YY} + I\right) \qquad (2\text{-}19)$$

mit

$$c_{ij} = C_{YY}\ i, j \qquad i, j = 1, 2, \ldots, n$$

definiert, für die ein globales Maximum unter der Nebenbedingung $Tr(C_{YY}) = P_Y$ gesucht wird.

Da der natürliche Logarithmus für positive Werte streng monoton wachsend ist, genügt es das Maximum der Kostenfunktion

$$R'(c_{11}, c_{12}, \ldots, c_{nn}) = \det\left(\frac{1}{P_\Phi} C_{YY} + I\right) \tag{2-20}$$

zu suchen. Bei der Größe C_{YY} handelt es sich um eine Korrelationsmatrix und die Größen $1/P_\Phi$ und I verändern die typischen Eigenschaften dieser Korrelationsmatrix nicht. Nach [PAP91] gilt für die Determinante einer $n \times n$ Korrelationsmatrix C die Beziehung

$$\det C \le C_{1,1} \cdot C_{2,2} \cdots C_{n,n} ,$$

mit Gleichheit genau dann, wenn C eine Diagonalmatrix darstellt. Die Kostenfunktion R'(.) besitzt also genau dann ein globales Maximum, wenn es sich bei

$$\frac{1}{P_\Phi} C_{YY} + I$$

um eine Diagonalmatrix handelt. Dies gilt dann und nur dann, wenn die Matrix C_{YY} eine Diagonalgestalt besitzt.

Unter diesen Voraussetzungen kann die zu untersuchende Kostenfunktion vereinfacht werden zu

$$R''(c_{11}, c_{22}, \ldots, c_{nn}) = \prod_{i=1}^{n} \left(\frac{1}{P_\Phi} C_{YY}(i,i) + 1\right) \tag{2-21}$$

mit

$$c_{ii} = C_{YY}(i,i) \qquad i = 1, 2, \ldots, n.$$

Satz 2-3: Es sei durch C_{YY} eine $n \times n$ Diagonalmatrix gegeben, deren Diagonalelemente in der Form

$$\frac{1}{P_\Phi} C_{YY}(i,i) + 1 \qquad i = 1, 2, \ldots, n$$

darstellbar sind. Dann besitzt die Funktion

$$f(c_{11}, c_{22}, \ldots, c_{nn}) = \prod_{i=1}^{n} \left(\frac{1}{P_\Phi} c_{ii} + 1\right) \qquad c_{ii} = C_{YY}(i,i)$$

unter der Nebenbedingung

$$\sum_{i=1}^{n} c_{ii} = P_Y$$

genau dann ein relatives Extremum, wenn die Matrix C_{YY} gegeben ist durch

$$C_{YY} = \frac{P_Y}{n} E.$$

Beweis 2-3: (siehe Anhang 1)

Das es sich bei diesem Extremum um ein globales Maximum handelt wird ersichtlich, wenn man berücksichtigt, daß es sich hier um das Problem der Maximierung eines Produkts handelt, bei dem die Gesamtsumme der einzelnen Produktterme vorgegeben ist. Dies entspricht einer Verallgemeinerung der Extremalwertaufgabe unter allen Rechtecken gleichen Umfangs U (vorgegebene Summe) jenes mit maximalem Flächeninhalt (maximales Produkt) zu bestimmen. Als einzige Lösung für diese Aufgabe erhält man ein Quadrat, bei dem alle Seiten gleich lang sind (die einzelnen Produktterme besitzen identische Werte).

Als Ergebnis dieses Kapitels kann man festhalten, daß bei einer Übertragung von Datenworten einer Verbundquelle Y über einen gestörten Kanal gemäß dem informationstheoretischen Modell aus Abbildung 2-7 genau dann ein Maximum an Information übertragen wird, wenn die Daten der Verbundquelle Y bereits orthonormalisiert sind. Denn nur dann besitzt die Korrelationsmatrix C_{YY} die oben hergeleitete Form.

Dieses Ergebnis bestätigt auch die ursprüngliche Vermutung auf Seite 18, daß sich die zu übertragenden Signale sehr deutlich von den Störungen abheben sollten. Die mittlere Leistung des Signals in jedem einzelnen Kanal des Übertragungskanals sollte soweit verstärkt werden, daß jeder Kanal voll ausgelastet wird. In diesem Fall wirken sich die additiven Störungen am wenigsten auf den Informationsgehalt des zu übertragenden Signals aus. Für *n* gleichartige Kanäle bedeutet dann die Vorgabe einer mittleren Gesamtleistung P_Y, daß jedes Signal in den einzelnen Kanälen des Übertragungskanals auf den gleichen Wert $(1 / n)\ P_Y$ verstärkt werden sollte.

Im Abschnitt Lineare Transformationen und unabhängige Teilquellen wurde gezeigt, daß die Ausgabemenge einer linearen Transformation aus Vektoren mit stochastisch unabhängigen Komponenten bestehen sollte. Dann besitzen die Ausgabedaten einen maximalen mittleren Informationsgehalt. In diesem Abschnitt wurde gezeigt, daß die Ausgabemenge der linearen Transformation aus Vektoren bestehen sollte, deren Komponenten dekorreliert und auf die gleiche Varianz normiert sind. In diesem Fall ist eine möglichst störsichere Übertragung über einen gestörten Übertragungskanal gegeben.

Nach [LAU79] sind stochastisch unabhängige Zufallsgrößen stets auch dekorreliert. Für den Fall der stochastischen Unabhängigkeit können Ausgabedaten mit einem maximalen mittleren Informationsgehalt auch möglichst störsicher übertragen werden.

Im allgemeinen sind dekorrelierte Zufallsgrößen aber *nicht* stochastisch unabhängig. Eine Äquivalenz von stochastischer Unabhängigkeit und Dekorrelation gilt nach [PAP91] jedoch für die sehr wichtige Klasse der normalverteilten Zufallsgrößen. Daher ist die Datenorthonormalisierung für die Klasse der normalverteilten Signale ausreichend und bietet in diesem Fall neben einer dekorrelierten (und stochastisch unabhängigen) Ausgabemenge mit einem maximalen mittleren Informationsgehalt auch eine möglichst störsichere Übertragung dieser Ausgabemenge über einen gestörten Übertragungskanal.

2.2. Transform Coding

Als Transform Coding bezeichnet man Verfahren der Signalkodierung, die eine zweistufige approximative (und damit fehlerbehaftete) Datenkompression durchführen können. Diese Verfahren sind notwendig für die Übertragung und Speicherung von sehr großen Informationsmengen, wie zum Beispiel bei

- hochauflösenden Fernsehbildern HDTV,
- digitaler Musik,
- Multi-Media Anwendungen,
- Telekommunikation,
- Satellitendaten,
- Bilddatenbanken

und einige andere mehr. In diesem Kapitel wird speziell das Problem untersucht, ein Bild in binäre Zahlen und diese binären Zahlen wieder zurück in eine Kopie des ursprünglichen Bildes zu transformieren. Dabei werden ausschließlich quadratische diskrete zweidimensionale Bilder mit einer bestimmten Anzahl von Graustufen betrachtet.

Als ich begann mich mit der Thematik des Transform Coding zu beschäftigen, machte ich die Erfahrung, daß Berichte und Bücher über diesen Bereich der Datenkompression viele grundlegende Fragen unbeantwortet ließen. So wurde beispielsweise in der Literatur oft kommentarlos für eine Eingabemenge X und Vektoren $x \in X$ der Fall $\langle x \rangle = 0$ angenommen, um die durchzuführenden Beweise einfacher zu halten. In diesem Fall sind die aus der Stochastik bekannten Kovarianzmatrix C_X und Korrelationsmatrix C_{XX} identisch. Was geschieht nun, wenn die Bedingung $\langle x \rangle = 0$ *nicht* gegeben ist und welche Rolle spielen dann die Matrizen C_X und C_{XX} ? In der Literatur fand ich auf diese Fragen keine Antwort. Außerdem blieben auch viele grundlegende Aussagen unbewiesen.

Aus diesem Grund stellt dieses Kapitel viel mehr als eine Einführung in die Thematik des Transform Coding dar. Es liefert Beweise für eine Vielzahl von grundlegenden Aussagen und setzt dabei nicht unbedingt den Fall $\langle x \rangle = 0$ voraus. Zusätzlich beschäftigt es sich mit der Bedeutung der Kovarianzmatrix C_X und der Korrelationsmatrix C_{XX} und liefert einen Beweis, mit dessen Hilfe geklärt werden kann, wie sich der Einsatz der Matrizen C_X oder C_{XX} auswirkt.

Insgesamt handelt es sich bei diesem Kapitel also um eine durch Beweise fundierte Einführung in die Thematik des Transform Coding, wobei ich alle wichtigen Beweise selbständig erarbeitet und großen Wert auf mathematische Korrektheit gelegt habe. Die grundlegenden Definitionen wurden aus [HAB71] und [WIN72] entnommen. Auch bei der Strukturierung des Kapitels habe ich mich von diesen Berichten inspirieren lassen, da sie einen sehr guten Überblick über die Thematik des Transform Coding geben.

2.2.1. Strukturen in Bildern

Bilddaten enthalten signifikante Strukturen. Eine effiziente Kodierung kann erreicht werden, indem zuerst die Strukturen der Bilddaten bestimmt werden und dann ein Kodierungsverfahren entwickelt wird, welches speziell auf Daten mit diesen Strukturen seine volle Leistungsfähigkeit entfaltet. Um so mehr Strukturen in den Bilddaten enthalten sind, um so größer ist die Leistungsfähigkeit eines Kodierers, der speziell auf diese Strukturen abgestimmt ist.

Wird nun ein Kodierer, der speziell auf bestimmte Strukturen in Bilddaten abgestimmt ist, auf Bilddaten mit einer anderen Struktur angewendet, so verringert sich die Leistungsfähigkeit des Kodierers proportional zur Unterschiedlichkeit der Strukturen in den beiden Bilddatenmengen [WIN72].

Bilddaten sind nicht homogen. Unterschiedliche Bereiche in Bildern enthalten unterschiedliche Strukturen. Nicht-adaptive Kodierer werden auf ein Mittel aller Strukturen der unterschiedlichen Bildbereiche abgestimmt.

Adaptive Kodierer hingegen werden auf die lokalen Strukturen jedes Bildbereichs abgestimmt, indem zuerst die lokalen Strukturen eines Bildbereichs ermittelt werden und anschließend ein Kodierer angewendet wird, der effizient auf diesen lokalen Strukturen arbeitet.

Um den lokalen Strukturen von unterschiedlichen Bildbereichen gerecht zu werden, müssen zu kodierende Bilder in kleinere Teilbilder partitioniert werden. Diese Teilbilder werden isoliert betrachtet und kodiert.

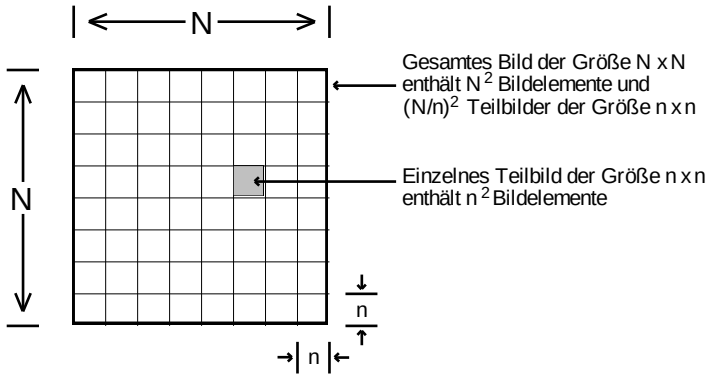


Abbildung 2-8 Partitionierung eines N × N Bildes in $(N / n)^2$ Teilbilder der Größe n × n

Im folgenden werden daher digitalisierte Bilder betrachtet, die aus einem Feld von N × N Bildelementen bestehen, wobei jedem Bildelement eine der 2^k Graustufen 1, 2, …, 2^k zugeordnet wird.

Ein solches Bild wird in eine Anzahl von Teilbildern partitioniert, die jeweils aus einem Feld von n × n Bildelementen bestehen, wobei n ≤ N gilt[4]. In Abbildung 2-8 wird das Schema einer solchen Partitionierung dargestellt.

Das *Transform Coding* stellt nun ein Verfahren dar, mit dem Bilder in einer Sequenz von zwei Operationen kodiert werden können. Die erste Operation besteht aus einer linearen Transformation, die im Idealfall eine Menge von stochastisch abhängigen Bildelementen in eine Menge von stochastisch unabhängigeren Koeffizienten transformiert. Die zweite Operation quantisiert jeden dieser Koeffizienten und bildet damit eine Menge benachbarter Punkte, welche hier als Klasse bezeichnet werden soll, auf einen einzigen Punkt ab, den sogenannten Klassenprototypen dieser Punktmenge.

Die Sequenz dieser beiden Operationen reduziert die Datenmenge, sie führt damit die eigentliche Datenkompression durch. Die folgende Abbildung stellt schematisch den Prozeß der Kodierung und Dekodierung dar.

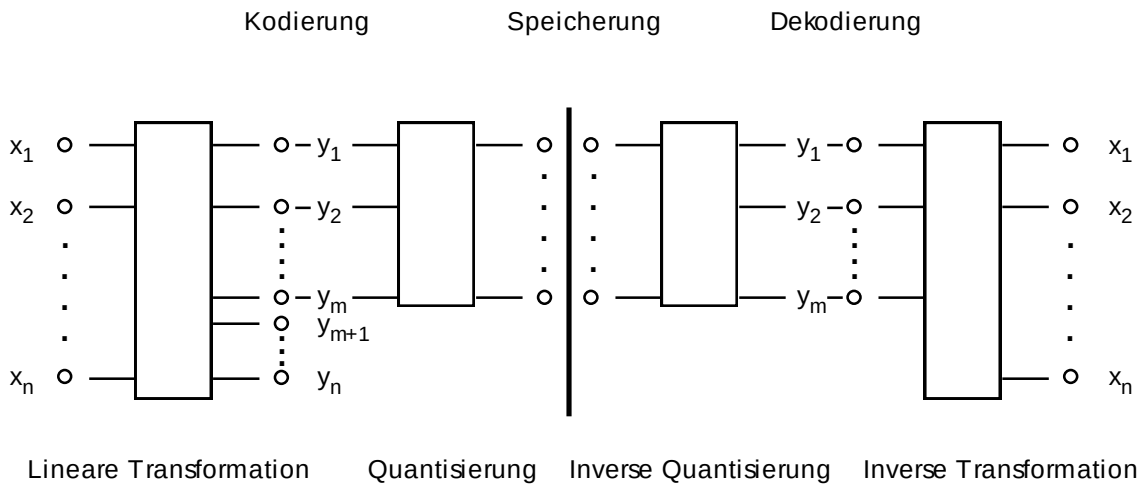


Abbildung 2-9 Transform Coding: Schema der Kodierung und Dekodierung

In den folgenden beiden Kapiteln wird nun auf die zwei Operationen *Lineare Transformation* und *Quantisierung* näher eingegangen.

2.2.2. Lineare Transformationen

Unter einer linearen Transformation soll hier eine Abbildung verstanden werden, die Punkte aus einem n-dimensionalen euklidischen Raum E^n in einen m-dimensionalen euklidischen Raum E^m abbildet, wobei m ≤ n gilt.

[4] Nach [WIN72] liegt die beste Größe für ein Teilbild bei n = 4 bis n = 8. Ist ein Teilbild zu groß, dann werden gerade bei adaptiven Kodierungssystemen lokale Bildstrukturen nicht ausreichend berücksichtigt. Ist ein Teilbild zu klein, so werden Abhängigkeiten zwischen den Bildelementen außer acht gelassen, die für lineare Transformationen wichtig sind.

Daher wird ein n × n Teilbild eines N × N Bildes mit n ≤ N (siehe Abbildung 2-8) als ein Punkt in einem n^2-dimensionalen Raum interpretiert, wobei jede der n^2 Koordinatenachsen des Raumes einem der n^2 Bildelemente zugeordnet wird und die Werte auf jeder Koordinatenachse den 2^k Graustufen jedes Bildelementes entsprechen.

Als Beispiel soll hier ein nicht-quadratisches Teilbild der Größe 1 × 2 dienen, das einem Teilbild bestehend aus zwei benachbarten Bildelementen entspricht. Beiden Bildelementen werden jeweils $2^3 = 8$ Graustufen zugeordnet. Dann können alle 64 möglichen Teilbilder in einem zweidimensionalen Koordinatensystem dargestellt werden.

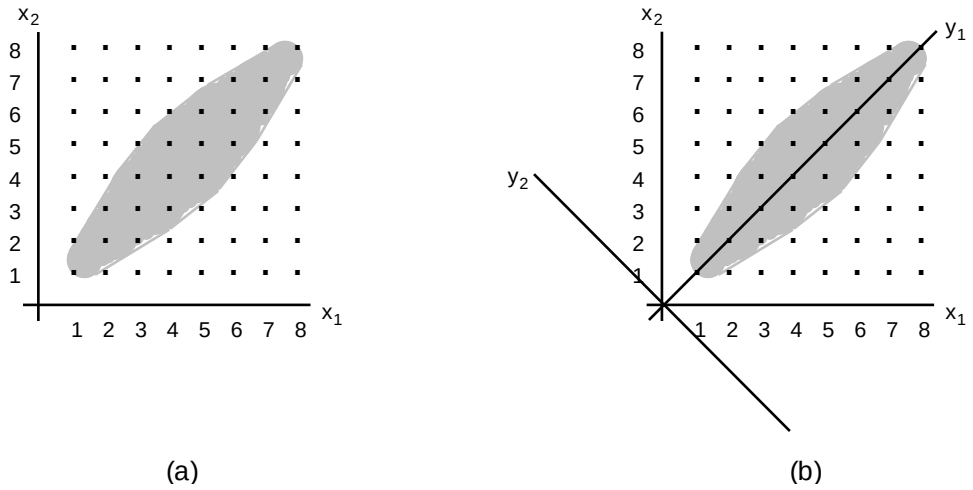


Abbildung 2-10 Koordinatensysteme für zwei benachbarte Bildelemente x_1 und x_2

Da benachbarte Bildelemente in natürlichen Bilddaten mit hoher Wahrscheinlichkeit eine ähnliche Graustufe besitzen [HAB71], liegen die wahrscheinlichsten Teilbilder dieses Beispiels im grauen Bereich des Koordinatensystems aus Abbildung 2-10 (a).

Wird nun eine Rotation des ursprünglichen Koordinatensystems durchgeführt, so liegen die wahrscheinlichsten Teilbilder nicht mehr auf der Diagonalen bei $x_1 = x_2$, sondern auf der y_1-Achse, wie in Abbildung 2-10 (b) dargestellt.

Daher sind die Variablen y_1 und y_2 unabhängiger, als die Variablen x_1 und x_2. Durch die Rotation des Koordinatensystems verändern sich auch die Varianzen der einzelnen Variablen, denn galt vor einer Rotation $\sigma_{x_1}^2 = \sigma_{x_2}^2$, so gilt danach $\sigma_{y_1}^2 > \sigma_{y_2}^2$.

Eine Verallgemeinerung des obigen Beispiels kann erreicht werden, indem Teilbilder der Größe n × n mit 2^k Graustufen betrachtet werden. In diesem Fall wird ein n^2-dimensionales Koordinatensystem mit den Achsenbeschriftungen 1, 2, …, 2^k benötigt. Jeder Punkt in dem Raum, der durch dieses Koordinatensystem aufgespannt wird, entspricht einem der $(2^k)^{n^2} = 2^{n^2 k}$ möglichen Teilbilder.

Eindimensionale Transformationen

Durch eine lineare eindimensionale[5] Transformation der Art

$$y = A\,x \tag{2-22}$$

kann eine Rotation des Koordinatensystems wie in Abbildung 2-10 durchgeführt werden, wobei es sich bei der Variablen x um einen Vektor mit n^2 Komponenten handelt, der die n^2 Graustufen der Bildelemente eines Teilbildes der Größe $n \times n$ enthält (siehe Abbildung 2-8). Die Zeilen des Teilbildes werden dabei hintereinander im Vektor x abgelegt, so daß sich für n Zeilen mit je n Bildelementen ein Vektor mit n^2 Komponenten ergibt. Bei der Variablen A handelt es sich um eine orthogonale[6] $n^2 \times n^2$ Matrix und bei der Variablen y um einen Vektor mit n^2 Komponenten. Was kann über die Komponenten des Vektors y bereits jetzt ausgesagt werden ?

Da es sich bei der Variablen A um eine orhogonale Transformationsmatrix handelt, gilt nach [STA83] der folgende Satz.

Satz 2-4: Es sei A eine reelle $n^2 \times n^2$ Matrix. Dann sind folgende Aussagen äquivalent:

(i) A ist orthogonal;
(ii) $A^T A = I$;
(iii) $AA^T = I$;
(iv) die Zeilen (wie auch die Spalten von A) bilden eine orthonormierte Basis des Vektorraums $\Re^{n^2}$, versehen mit dem Standard-Skalarprodukt.

Mit dem Beweis des folgenden Korollars läßt sich dann eine genaue Aussage über die Komponenten des Vektors y machen. Dieses Korollar und sein Beweis wurden ohne Änderungen aus [STA83] entnommen.

Korollar 2-5: Es sei V ein endlich-dimensionaler Vektorraum mit Skalarprodukt. Weiterhin sei $\{a^1, a^2, ..., a^{n^2}\}$ eine orthonormierte Basis. Dann gilt für jedes $x \in V$

$$x = \sum_{i=1}^{n^2} \langle x, a^i \rangle a^i.$$

---

5 In [HAB71] und [WIN72] wird zwischen den sogenannten eindimensionalen und zweidimensionalen Transformationen unterschieden. Bei einer eindimensionalen Transformation werden die Bildelemente eines Teilbildes in einem *Vektor* abgelegt und anschließend mit Hilfe einer Matrix transformiert. Bei den zweidimensionalen Transformationen werden die Bildelemente eines Teilbildes in einer *Matrix* abgelegt und mit Hilfe eines Punkttensors vierter Ordnung transformiert (siehe auch Abschnitt Zweidimensionale Transformationen).

6 Die hier betrachteten eindimensionalen Transformationen können im reellen Fall durch eine orthogonale und im komplexen Fall durch eine unitäre Matrix beschrieben werden. Hier wird ausschließlich der reelle Fall betrachtet, da der komplexe Fall analog behandelt werden kann.

Beweis 2-5: Es sei jedes $x \in V$ durch eine Linearkombination

$$x = \sum_{i=1}^{n^2} y_i \, a^i$$

gegeben. Dann folgt

$$\langle x, a^i \rangle = \left\langle \sum_{j=1}^{n^2} y_j \, a^j, a^i \right\rangle = \left\langle \sum_{j=1}^{n^2} y_j \, a^j, a^i \right\rangle = y_i \langle a^i, a^i \rangle$$

$$= y_i \left| a^i \right|^2 = y_i \qquad i = 1, 2, \ldots, n^2. \qquad \text{q.e.d.}$$

Da es sich bei A eine orthogonale Matrix handelt, bilden die Zeilen von A eine orthonormierte Basis des Vektorraums $\Re^{n^2}$ (siehe Satz 2-4). Wird nun eine Transformation wie in Gleichung 2-22 durchgeführt, so enthält der Vektor y nach Korollar 2-5 die Koordinaten des Vektors x bezüglich der orthonormierten Basis in den Zeilen von A, wobei diese Koordinaten im folgenden als Koeffizienten bezeichnet werden.

Mit Hilfe von Satz 2-4 kann die inverse Transformation zu Gleichung 2-22 bestimmt werden, indem die Eigenschaft $A^T = A^{-1}$ einer orthogonalen Matrix verwendet wird

$$x = A^T y. \tag{2-23}$$

Damit ergibt sich für die Transformation, daß jeder Koeffizient y_k eine Linearkombination aller Bildelemente seines Teilbildes darstellt

$$y_k = \sum_{i=1}^{n^2} a_{ki} \, x_i \qquad k = 1, 2, \ldots, n^2, \tag{2-24}$$

während für die inverse Transformation jedes Bildelement x_k eine Linearkombination aller Koeffizienten seines Teilbildes ist

$$x_k = \sum_{i=1}^{n^2} a_{ik} \, y_i \qquad k = 1, 2, \ldots, n^2. \tag{2-25}$$

Am Beispiel auf Seite 33 wurde bereits gezeigt, daß sich die Varianzen $\sigma^2_{x_k}$ der einzelnen Bildelemente x_k von den Varianzen $\sigma^2_{y_k}$ der durch die Rotation des Koordinatensystems resultierenden Koeffizienten y_k unterscheiden. Was läßt sich nun über die Summe der Varianzen

$$\sum_{i=1}^{n^2} \sigma^2_{y_i} \tag{2-26}$$

aussagen ? Diese Frage soll mit dem folgenden Korollar beantwortet werden.

Korollar 2-6: Es sei A eine orthogonale $n^2 \times n^2$ Matrix und $y = A\,x$ eine lineare Transformation. Dann gilt

$$\sum_{i=1}^{n^2} \sigma_{y_i}^2 = \sum_{i=1}^{n^2} \sigma_{x_i}^2 \,.$$

Beweis 2-6:

$$\sum_{i=1}^{n^2} \sigma_{y_i}^2 = \sum_{i=1}^{n^2} \left\langle \left(y_i - \langle y_i \rangle \right)^2 \right\rangle = \left\langle \sum_{i=1}^{n^2} \left(y_i - \langle y_i \rangle \right)^2 \right\rangle = \left\langle \left(y - \langle y \rangle \right)^T \left(y - \langle y \rangle \right) \right\rangle$$

$$= \left\langle y^T y - y^T \langle y \rangle - \langle y^T \rangle y + \langle y^T \rangle \langle y \rangle \right\rangle = \left\langle y^T y \right\rangle - \langle y^T \rangle \langle y \rangle \underbrace{- \langle y^T \rangle \langle y \rangle + \langle y^T \rangle \langle y \rangle}_{0}$$

$$= \left\langle y^T y \right\rangle - \langle y^T \rangle \langle y \rangle = \left\langle x^T A^T A\, x \right\rangle - \langle x^T A^T \rangle \langle A\, x \rangle$$

$$= \left\langle x^T A^T A\, x \right\rangle - \langle x^T \rangle A^T A \langle x \rangle = \left\langle x^T x \right\rangle - \langle x^T \rangle \langle x \rangle$$

$$= \left\langle \left(x - \langle x \rangle \right)^T \left(x - \langle x \rangle \right) \right\rangle = \left\langle \sum_{i=1}^{n^2} \left(x_i - \langle x_i \rangle \right)^2 \right\rangle = \sum_{i=1}^{n^2} \left\langle \left(x_i - \langle x_i \rangle \right)^2 \right\rangle = \sum_{i=1}^{n^2} \sigma_{x_i}^2 \qquad \text{q.e.d.}$$

Das obige Korollar zeigt, daß lineare Transformationen mit einer orthogonalen Matrix A varianz-erhaltend sind, so daß für das Beispiel auf Seite 33 gilt: $\sigma_{y_1}^2 + \sigma_{y_2}^2 = \sigma_{x_1}^2 + \sigma_{x_2}^2$.

Hotelling Transformation

Nach Kapitel 2.1.2. wäre es informationstheoretisch von großem Vorteil eine lineare Transformation zu verwenden, die stochastisch unabhängige Koeffizienten liefert. Betrachtet man die Ausgabemenge aller Vektoren einer linearen Transformation als Verbundquelle Y mit n^2 Teilquellen, dann vermindern nach Satz 2-1 stochastische Abhängigkeiten zwischen den einzelnen Teilquellen die Entropie der Verbundquelle Y und damit auch den mittleren Informationsgehalt der Datenworte $y \in Y$. Nach [WIN72] kann eine umkehrbare lineare Transformation, die stochastisch unabhängige Koeffizienten liefert, nicht ohne weiteres bestimmt werden.

Es ist aber durchaus möglich eine umkehrbare Transformation zu bestimmen, die unkorrelierte[7] Koeffizienten liefert. Betrachtet man die Dekorrelation als eine wichtige Vorstufe der stochastischen Unabhängigkeit, dann ist es informationstheoretisch bereits vorteilhaft eine Dekorrelation durchzuführen.

---

[7] Nach [LAU79] sind zwei Zufallsgrößen Y^i und Y^j unkorreliert, falls die Kovarianz von Y^i und Y^j den Wert 0 ergibt, also $\langle (y_i - \langle y_i \rangle)\,(y_j - \langle y_j \rangle) \rangle = 0$ für $y_i \in Y^i$ und $y_j \in Y^j$.

Wird Y durch eine normalverteilte Zufallsgröße beschrieben, dann sind Dekorrelation und stochastische Unabhängigkeit zwischen den einzelnen Teilquellen von Y äquivalent (siehe Abschnitt Die maximal erreichbare Transinformation). In diesem Fall wäre die Dekorrelation bereits ausreichend (und informationstheoretisch optimal), um einen maximalen mittleren Informationsgehalt in den Datenworten $y \in Y$ zu bekommen.

Nach [WIN72] liegt eine Transformation mit unkorrelierten Koeffizienten am dichtesten an der Transformation, die unabhängige Koeffizienten liefern würde. Es wird nun dargestellt, wie eine orthogonale $n^2 \times n^2$ Transformationsmatrix A berechnet werden kann, die mit Gleichung 2-22 unkorrelierte Koeffizienten y_k produziert.

Nachdem alle $n \times n$ Teilbilder eines $N \times N$ Bildes (siehe Abbildung 2-8) in Vektoren x mit jeweils n^2 Komponenten abgelegt wurden, kann deren Kovarianzmatrix C_X berechnet werden durch

$$C_X = \left\langle \left(x - \langle x \rangle\right)\left(x - \langle x \rangle\right)^T \right\rangle = \left\langle x\,x^T \right\rangle - \langle x \rangle \left\langle x^T \right\rangle \tag{2-27}$$

wobei C_X die Statistik eines Bildes beschreibt. Gilt die Bedingung $\langle x \rangle = 0$, so vereinfacht sich die obige Gleichung zu

$$C_{XX} = \left\langle x\,x^T \right\rangle, \tag{2-28}$$

wobei C_{XX} als Korrelationsmatrix bezeichnet wird.

Die normierten Eigenvektoren $\{e^1, e^2, ..., e^{n^2}\}$ von C_X werden nun in den Zeilen der Matrix A abgelegt, wie in der folgenden Gleichung 2-29 dargestellt.

$$A = \begin{pmatrix} e^{1T} \\ \vdots \\ e^{n^2 T} \end{pmatrix} \tag{2-29}$$

Im folgenden wird bewiesen, daß die so konstruierte Matrix A eine orthogonale Matrix darstellt und mit Gleichung 2-22 unkorrelierte Koeffizienten liefert.

Die Frage, ob die obige Matrix A eine orthogonale Matrix darstellt, läßt sich sehr einfach beantworten, wenn man berücksichtigt, daß es sich bei der Matrix C_X um eine reelle symmetrische Matrix handelt. Denn nach [STA83] beschreibt jede symmetrische Matrix eine lineare selbstadjungierte Selbstabbildung[8] bezüglich einer orthonormierten Basis. Das folgende Korollar zeigt, daß Eigenvektoren einer linearen selbstadjungierten Selbstabbildung $f: V \rightarrow V$ (und damit einer reellen symmetrischen Matrix C_X) orthogonal zueinander sind.

[8] Die lineare Selbstabbildung $f: V \rightarrow V$ heißt selbstadjungiert, falls $\forall\, a, b \in V$ gilt: $(f(a), b) = (a, f(b))$.

Korollar 2-7: Es sei V ein Vektorraum mit Skalarprodukt und f: V → V eine lineare selbstadjungierte Selbstabbildung. Dann sind die Eigenvektoren zu verschiedenen Eigenwerten von f orthogonal.

Beweis 2-7: Da f selbstadjungiert ist, gilt für ∀ a, b ∈ V

$$(f(a), b) = (a, f(b)) \ .$$

Falls a und b Eigenvektoren zu verschiedenen Eigenwerten λ_a und λ_b von f sind, folgt:

$$(f(a), b) = (\lambda_a a, b) = \lambda_a (a, b)$$

$$(a, f(b)) = (a, \lambda_b b) = \lambda_b (a, b)$$

$$\Rightarrow \lambda_a (a, b) = \lambda_b (a, b)$$

$$\Rightarrow \lambda_a (a, b) - \lambda_b (a, b) = (\lambda_a - \lambda_b)(a, b) = 0$$

Da nach Voraussetzung $\lambda_a \neq \lambda_b$ gilt, muß (a, b) = 0 sein und damit a ⊥ b. q.e.d.

Die Zeilenvektoren der Matrix A sind durch normierte Eigenvektoren der Matrix C_X gegeben, die nach Korollar 2-7 orthogonal zueinander sind. Damit enthält A eine orthonormierte Eigenvektorbasis von $\Re^{n^2}$ in den Zeilen und stellt nach Satz 2-4 eine orthogonale Matrix dar. Es bleibt noch zu zeigen, daß mit Gleichung 2-22 und 2-29 berechnete Koeffizienten unkorreliert sind.

Satz 2-5: Es sei y = A x eine lineare Transformation mit einer orthogonalen $n^2 \times n^2$ Matrix A, die als Zeilenvektoren die orthonormierten Eigenvektoren $\{e^1, e^2, ..., e^{n^2}\}$ der Kovarianzmatrix C_X enthält. Dann gilt

$$\left\langle (y_i - \langle y_i \rangle)(y_j - \langle y_j \rangle) \right\rangle = 0 \qquad i, j = 1, 2, \ldots, n^2;\ i \neq j,$$

$$\left\langle (y_i - \langle y_i \rangle)^2 \right\rangle = \lambda_i \qquad i = 1, 2, \ldots, n^2.$$

Beweis 2-5: Sei λ_i der Eigenwert zum normierten Eigenvektor e^i der Matrix C_X.

$$\left\langle (y_i - \langle y_i \rangle)(y_j - \langle y_j \rangle) \right\rangle = \left\langle y_i\, y_j \right\rangle - \langle y_i \rangle \langle y_j \rangle$$

$$= \left\langle e^{iT} x\, x^T e^j \right\rangle - \left\langle e^{iT} x \right\rangle \left\langle x^T e^j \right\rangle = e^{iT} \left\langle x\, x^T \right\rangle e^j - e^{iT} \langle x \rangle \left\langle x^T \right\rangle e^j$$

$$= e^{iT} \left(\left\langle x x^T \right\rangle - \langle x \rangle \left\langle x^T \right\rangle \right) e^j = e^{iT} C_X\, e^j$$

$$= e^{iT} \lambda_j e^j = \lambda_j e^{iT} e^j = 0 \qquad i, j = 1, 2, \ldots, n^2;\ i \neq j$$

und

$$\left\langle \left(y_i - \langle y_i \rangle \right)^2 \right\rangle = \left\langle y_i^2 \right\rangle - \langle y_i \rangle^2 = \langle y_i\, y_i \rangle - \langle y_i \rangle \langle y_i \rangle$$

$$= \left\langle e^{iT}\, x\, x^T\, e^i \right\rangle - \left\langle e^{iT}\, x \right\rangle \left\langle x^T\, e^i \right\rangle = e^{iT} \left\langle x\, x^T \right\rangle e^i - e^{iT} \langle x \rangle \left\langle x^T \right\rangle e^i$$

$$= e^{iT} \left(\left\langle x\, x^T \right\rangle - \langle x \rangle \left\langle x^T \right\rangle \right) e^i = e^{iT}\, C_X\, e^i$$

$$= e^{iT}\, \lambda_i\, e^i = \lambda_i\, e^{iT}\, e^i = \lambda_i \left| e^i \right|^2 = \lambda_i \qquad i = 1, 2, \ldots, n^2 \qquad \text{q.e.d.}$$

Der obige Satz zeigt auch, daß die Orthogonalität der Eigenvektoren $\{e^1, e^2, ..., e^{n^2}\}$ eine notwendige Bedingung ist, um die Dekomposition einer Eingabemenge in eine unkorrelierte Ausgabemenge durchzuführen. Die Orthonormalität ist eine strengere Bedingung, aus der sich ergibt, daß eine lineare Transformation varianz-erhaltend ist (siehe Korollar 2-6) und damit keine Skalierung der Eingabemenge durchführt.

Die Kovarianzmatrix C_Y der resultierenden Koeffizienten ist somit gegeben durch

$$C_Y = \begin{pmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \cdots & 0 & \lambda_{n^2} \end{pmatrix} \tag{2-30}$$

wobei es sich bei $\lambda_1, \lambda_2, \ldots, \lambda_{n^2}$ um die Eigenwerte zu den Eigenvektoren $\{e^1, e^2, ..., e^{n^2}\}$ handelt. Die hier beschriebene Dekomposition einer Eingabemenge in eine Menge von unkorrelierten Koeffizienten bezüglich einer orthonormierten Eigenvektorbasis wird als Hotelling Transformation, Eigenvektortransformation oder Hauptkomponentenanalyse bezeichnet.

Zweidimensionale Transformationen

In letzten Abschnitt wurden n × n Teilbilder eines N × N Bildes in Vektoren x mit jeweils n^2 Komponenten abgelegt und mit Hilfe von Gleichung 2-22 transformiert. Es besteht aber auch die Möglichkeit die n^2 Graustufen eines n × n Teilbildes in einer n × n Matrix X abzulegen

$$X = \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nn} \end{pmatrix} \tag{2-31}$$

und diese Matrix in eine andere n × n Matrix Y mit Koeffizienten y_{kl} zu transformieren

$$Y = \begin{pmatrix} y_{11} & y_{12} & \cdots & y_{1n} \\ y_{21} & y_{22} & \cdots & y_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ y_{n1} & y_{n2} & \cdots & y_{nn} \end{pmatrix} \tag{2-32}$$

indem ein Punkttensor vierter Ordnung, der im folgenden als Transformationskernel bezeichnet wird, mit der Matrix X multipliziert wird. Dieser Transformationskernel enthält n^4 Elemente $a_{kl_{ij}}$ k, l, i, j = 1, 2, …, n und entspricht der orthogonalen[9] $n^2 \times n^2$ Transformationsmatrix A (siehe Seite 34), die ebenfalls n^4 Elemente enthält. Eine Transformationsmatrix A kann zu einem Transformationskernel umstrukturiert werden, indem jede der n^2 Zeilen von A mit jeweils n^2 Elementen in die n × n Matrix A'_{kl} mit k, l = 1, 2, …, n umgestaltet wird. Die folgende Abbildung zeigt das Schema einer solchen Umstrukturierung.

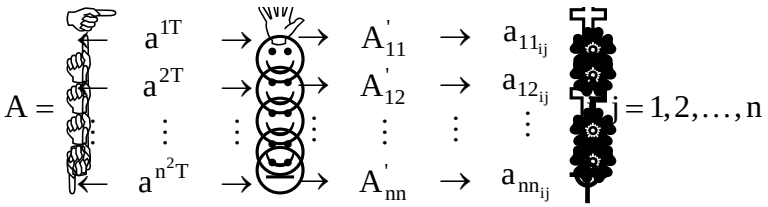


Abbildung 2-11 Umstrukturierung einer $n^2 \times n^2$ Transformationsmatrix A zu einem Transformationskernel mit n^4 Elementen

Durch eine zweidimensionale lineare Transformation der Art

$$y_{kl} = \sum_{i=1}^{n}\sum_{j=1}^{n} a_{kl_{ij}}\, x_{ij} \qquad k, l = 1, 2, \ldots, n \tag{2-33}$$

können die Koeffizienten y_{kl} der Matrix Y berechnet werden. Auf eine ähnliche Weise werden durch die inverse Transformation

$$x_{kl} = \sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij_{kl}}\, y_{ij} \qquad k, l = 1, 2, \ldots, n \tag{2-34}$$

[9] Die hier betrachteten zweidimensionalen Transformationen können im reellen Fall durch ein orthogonales und im komplexen Fall durch ein unitäres Transformationskernel beschrieben werden. Hier wird ausschließlich der reelle Fall betrachtet, da der komplexe Fall analog behandelt werden kann.

aus den Koeffizienten y_{ij} wieder die Bildelemente x_{kl} der Matrix X bestimmt. Es zeigt sich, daß die eindimensionale wie auch die zweidimensionale lineare Transformation eine Menge von n^2 Koeffizienten liefert, wobei jeder dieser Koeffizienten eine Linearkombination aller n^2 Bildelemente seines Teilbildes darstellt (siehe Gleichung 2-24 und 2-33). Daher unterscheiden sich diese beiden Transformationen nur in der Notation.

Separierbare zweidimensionale Transformationen

Für Berechnungen und Simulationen auf einem Computer ist es sinnvoll, Operationen auf Basis von Matrizen und Vektoren durchführen zu können (siehe Kapitel 4.). Mit einem Punkttensor vierter Ordnung als Transformationskernel für zweidimensionale lineare Transformationen ist dies nicht möglich. Falls aber der Transformationskernel separierbar ist, können auch zweidimensionale Transformationen ausschließlich mit Vektoren und Matrizen durchgeführt werden.

Nach [JAY84] ist ein Transformationskernel separierbar, falls sich dieser in getrennte horizontale und vertikale Operationen aufteilen läßt, so daß sich jedes Element $a_{kl_{ij}}$ des Kernels darstellen läßt als

$$a_{kl_{ij}} = A_v\ k,i\ A_h\ l,j\ , \qquad k, l, i, j = 1, 2, \ldots, n \qquad (2\text{-}35)$$

wobei es sich bei A_v und A_h um passend gewählte $n \times n$ Matrizen handelt.

Nun lassen sich die Gleichungen 2-33 und 2-34 umformen in

$$y_{kl} = \sum_{i=1}^{n}\sum_{j=1}^{n} a_{kl_{ij}}\, x_{ij} = \sum_{i=1}^{n}\sum_{j=1}^{n} A_v\ k,i\ A_h\ l,j\ x_{ij} = \sum_{i=1}^{n} A_v\ k,i\ \sum_{j=1}^{n} x_{ij}\, A_h\ l,j$$

$$\Rightarrow Y = A_v X A_h^T \qquad k, l = 1, 2, \ldots, n \qquad (2\text{-}36)$$

und

$$x_{kl} = \sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij_{kl}}\, y_{ij} = \sum_{i=1}^{n}\sum_{j=1}^{n} A_v\ i,k\ A_h\ j,l\ y_{ij}$$

$$\Rightarrow X = \sum_{i=1}^{n}\sum_{j=1}^{n} y_{ij}\, a_v^i\, a_h^{jT} = \sum_{i=1}^{n}\sum_{j=1}^{n} y_{ij}\, A_{ij}^{vh} \qquad k, l = 1, 2, \ldots, n \qquad (2\text{-}37)$$

mit $a_v^i = A_v\ i,k\ _{k=1,2,\ldots,n}$ sowie $a_h^j = A_h\ j,l\ _{l=1,2,\ldots,n}$.

Falls der Transformationskernel separierbar und symmetrisch ist, dann gilt $A_v = A_h = A$. In diesem Fall lassen sich die hier betrachteten orthogonalen Transformationen, sowie deren inverse Transformationen darstellen als

$$Y = A\,X\,A^T \tag{2-38}$$

und

$$X = A^T\,Y\,A. \tag{2-39}$$

Diskrete Fourier Transformation

In diesem und den beiden folgenden Abschnitten werden drei zweidimensionale lineare Transformationen vorgestellt, die im Bereich der Bildkodierung sehr populär sind. Jede der im folgenden beschriebenen Transformationen kann mit Hilfe der Gleichungen 2-33 und 2-34 modelliert werden.

Dabei enthält der Transformationskernel der Fourier Transformation und der Hadamard Transformation nur eingabe-unabhängige Elemente, während die Karhunen-Loève Transformation eingabe-abhängige Kernel-Elemente liefert.

Die zuerst betrachtete Fourier Transformation ist gegeben durch

$$a_{kl_{ij}} = \frac{1}{n} e^{\frac{-2\pi\sqrt{-1}}{n}(k-l)(i-j)} \qquad k, l, i, j = 1, 2, \ldots, n. \tag{2-40}$$

Der durch diese Transformation definierte Kernel bewirkt nach [JAY84] keine Diagonalisierung der Kovarianzmatrix C_Y, woraus sich ergibt, daß die resultierenden Koeffizienten y_{kl} dieser Transformation nicht unkorreliert sind.

Der Transformationskernel der Fourier Transformation ist jedoch separierbar

$$a_{kl_{ij}} = \frac{1}{n} e^{\frac{-2\pi\sqrt{-1}}{n}(k-l)(i-j)} = \frac{1}{n} e^{\frac{-2\pi\sqrt{-1}}{n}ki} e^{\frac{2\pi\sqrt{-1}}{n}lj}$$

$$A_v(k,i) = \frac{1}{\sqrt{n}} e^{\frac{-2\pi\sqrt{-1}}{n}ki}$$

$$A_h(l,j) = \frac{1}{\sqrt{n}} e^{\frac{2\pi\sqrt{-1}}{n}lj}$$

$$\Rightarrow a_{kl_{ij}} = A_v(k,i)\,A_h(l,j) \qquad k, l, i, j = 1, 2, \ldots, n$$

und auch symmetrisch

$$a_{kl_{ij}} = \frac{1}{n} e^{\frac{-2\pi\sqrt{-1}}{n}(k-l)(i-j)} = \frac{1}{n} e^{\frac{-2\pi\sqrt{-1}}{n}(k-l)(j-i)}$$

$$\Rightarrow a_{kl_{ij}} = a_{ij_{kl}} \qquad k, l, i, j = 1, 2, \ldots, n.$$

Nach [JAY84] handelt es sich bei dem obigen Transformationskernel um einen unitären Kernel mit komplexen Elementen.

Daher ist der Kernel der inversen Transformation durch den konjugiert-transponierten Transformationskernel gegeben, so daß Gleichung 2-34 modifiziert wird zu

$$x_{kl} = \sum_{i=1}^{n}\sum_{j=1}^{n} \bar{a}_{ij_{kl}}\, y_{ij} \qquad k, l = 1, 2, \ldots, n.$$

Hadamard Transformation

Die Hadamard Transformation ist nach [JAY84] durch ein orthogonales Transformationskernel charakterisiert und gegeben durch

$$a_{kl_{ij}} = \frac{1}{n}(-1)^{b_{kl_{ij}}} \qquad k, l, i, j = 1, 2, \ldots, n \qquad (2\text{-}41)$$

$$b_{kl_{ij}} = \sum_{h=0}^{\log_2 n-1} \left(b_h(k)\, b_h(l) + b_h(i)\, b_h(j) \right) \qquad k, l, i, j = 1, 2, \ldots, n, \qquad (2\text{-}42)$$

wobei es sich bei $b_h(.)$ um das h-te Bit der binären Repräsentation von (.) handelt. Der so konstruierte Transformationskernel ist symmetrisch

$$b_{kl_{ij}} = \sum_{h=0}^{\log_2 n-1} \left(b_h(k)\, b_h(l) + b_h(i)\, b_h(j) \right) = \sum_{h=0}^{\log_2 n-1} \left(b_h(i)\, b_h(j) + b_h(k)\, b_h(l) \right)$$

$$\Rightarrow b_{kl_{ij}} = b_{ij_{kl}}$$

$$\Rightarrow a_{kl_{ij}} = a_{ij_{kl}} \qquad k, l, i, j = 1, 2, \ldots, n,$$

jedoch nicht separierbar [JAY84] und liefert, wie auch die Fourier Transformation, keine unkorrelierten Koeffizienten y_{kl}.

Karhunen-Loève Transformation

Die Karhunen-Loève Transformation wird in der Literatur auch als optimale Transformation bezeichnet, da sie bei einer Approximation von Teilbildern den dort entstehenden mittleren quadratischen Fehler minimiert (siehe Kapitel 2.2.4.). Gilt zusätzlich die Bedingung $\langle X \rangle = 0$ für die betrachteten $n \times n$ Teilbilder, dann liefert die Karhunen-Loève Transformation nach [HAB71] unkorrelierte Koeffizienten y_{kl}.

Da es sich bei der Karhunen-Loève Transformation um eine kontinuierliche Funktion handelt, ist der Transformationskernel durch n^2 Basisfunktionen $a_{kl}(.)$ definiert. Die Basisfunktionen, welche den mittleren quadratischen Fehler für eine gegebene Menge von $n \times n$ Teilbildern minimieren, sind nach [HAB71] durch die n^2 Eigenfunktionen zu den n^2 größten Eigenwerten der Integralgleichung

$$\lambda_{kl}\, a_{kl}(i,j) = \int_0^1 \int_0^1 c(i,j,g,h)\, a_{kl}(g,h)\, dg\, dh \qquad k, l = 1, 2, \ldots, n \qquad (2\text{-}43)$$

gegeben, wobei es sich bei c(.) um die Korrelationsfunktion der gegebenen Teilbilder handelt. Sie berechnet die Korrelation zwischen zwei Bildelementen x_{ij} und x_{gh}.

Ist diese Korrelationsfunktion separierbar, dann sind nach [HAB71] auch die Eigenfunktionen und Eigenwerte separierbar, so daß sich mit

$$c(i,j,g,h) = c_v(i,g)\, c_h(j,l)$$

und

$$\lambda_{kl} = \lambda_v\, k\, \lambda_h\, l \qquad k, l = 1, 2, \ldots, n,$$

$$a_{kl}(i,j) = A_v\, k\, A_h\, l\, (i,j) \qquad k, l = 1, 2, \ldots, n$$

das Doppelintegral in Gleichung 2-43 in zwei Einzelintegrale

$$\lambda_v\, k\, A_v\, k\, (i) = \int_0^1 c_v(i,g)\, A_v k\, (g)\, dg \qquad k = 1, 2, \ldots, n$$

$$\lambda_h\, l\, A_h\, l\, (j) = \int_0^1 c_h(j,h)\, A_h l\, (h)\, dh \qquad l = 1, 2, \ldots, n$$

aufteilen läßt.

Für die Modellierung einer separierbaren Korrelationsfunktion, mit deren Hilfe sich Statistiken von natürlichen Bildern[10] beschreiben lassen, gibt es in [HAB71] einen sehr überzeugenden Lösungsvorschlag. Experimente mit unterschiedlichen Bilddaten zeigten, daß eine Korrelationsfunktion für eine breite Palette von Bildmaterialien gegeben ist durch

$$c(i,j,g,h) = e^{-(\alpha|i-g|-\beta|j-h|)}, \qquad (2\text{-}44)$$

[10] Bilddaten, welche durch die Natur gegeben sind, wie zum Beispiel Bäume, Gesichter, ein Horizont oder eine Menschenmenge.

welche die Korrelation zwischen den Bildelementen x_{ij} und x_{gh} bestimmt. Durch die Parameter α und β lassen sich auch Statistiken von Bildern beschreiben, die in horizontaler und vertikaler Richtung unterschiedliche Korrelationen besitzen.

Für den diskreten Fall mit $\langle X \rangle = 0$ entspricht die Karhunen-Loève Transformation der Hotelling Transformation und stellt damit eine Verallgemeinerung der Hotelling Transformation dar. Bei einer Betrachtung von diskreten Bilddaten müssen lediglich die Integrale der Karhunen-Loève Transformation durch diskrete Summen und die Funktionen durch Matrizen oder höherdimensionale Punkttensoren ersetzt werden.

Basisbilder und Eigenbilder

Eine lineare Transformation für zweidimensionale Bilder, die ein Teilbild, das durch eine n × n Matrix X beschrieben wird, in eine n × n Matrix Y von Koeffizienten transformiert (siehe Gleichung 2-33), kann auf besondere Art und Weise interpretiert werden. Dazu wird die inverse Transformation aus Gleichung 2-34 dargestellt als

$$X = \sum_{i=1}^{n} \sum_{j=1}^{n} y_{ij} A'_{ij}, \tag{2-45}$$

so daß sich eine Linearkombination von Basisbildern

$$A'_{ij} = \begin{pmatrix} a_{ij_{11}} & a_{ij_{12}} & \cdots & a_{ij_{1n}} \\ a_{ij_{21}} & a_{ij_{22}} & \cdots & a_{ij_{2n}} \\ \vdots & \vdots & \ddots & \vdots \\ a_{ij_{n1}} & a_{ij_{n2}} & \cdots & a_{ij_{nn}} \end{pmatrix} \qquad i, j = 1, 2, \ldots, n \tag{2-46}$$

mit den Koeffizienten y_{ij} i, j = 1, 2, …, n ergibt. Daher läßt sich ein Teilbild X durch eine gewichtete Summe von orthonormalen Basisbildern A'_{ij} darstellen, wobei die Gewichte y_{ij} nach Gleichung 2-33 gegeben sind durch

$$y_{ij} = \overline{A'_{ij}} \, X \qquad i, j = 1, 2, \ldots, n \tag{2-47}$$

und somit als Korrelation zwischen dem Teilbild X und einem Basisbild A'_{ij} interpretiert werden können. Bezieht man obige Sachverhalte nun auf die Hotelling Transformation, dann handelt es sich bei den Basisbildern um Eigenvektoren, und diese werden nach [WIN72] als Eigenbilder bezeichnet. Wie stellen sich Eigenbilder in der Realität dar ?

Um die Frage zu klären, wird in meiner Diplomarbeit eine Visualisierung von Eigenbildern in Graustufen durchgeführt. Dies entspricht dem Datenmaterial aus [WIN72], so daß es sich bei der folgenden Visualisierung um eine Überprüfung von Versuchsergebnissen handelt.

Zur Durchführung der Hotelling Transformation muß aus vorhandenem Bildmaterial eine $n^2 \times n^2$ Kovarianzmatrix C_X bestimmt und deren Eigenwerte und Eigenvektoren berechnet werden. Für die Visualisierung werden die Eigenvektoren gemäß ihrer Eigenwerte absteigend sortiert. Dabei besitzt jeder der n^2 Eigenvektoren genau n^2 Komponenten. Anschließend werden die Eigenvektoren in Teilstücke der Länge n zerlegt und in $n \times n$ Matrizen A'_{ij} mit i, j = 1, 2, …, n zeilenweise abgelegt. Jeder einzelne der n^2 Eigenvektoren beschreibt damit eines der n^2 Eigenbilder A'_{ij}.

Um eine aufwendige Verarbeitung von Bilddaten zu vermeiden, wird für die Visualisierung eine *synthetische* Korrelationsmatrix C_{XX} mit der Korrelationsfunktion aus [HAB71] berechnet (siehe Abschnitt Karhunen-Loève Transformation).

Die für das Experiment entwickelte Software beruht auf der von mir selbständig erstellten objektorientierten Klassenbibliothek MAC, welche in Kapitel 4. ausführlich beschrieben wird.

Für die folgende Abbildung wurden die Eigenbilder einer 64×64 Korrelationsmatrix C_{XX} berechnet. Zur Bestimmung der Korrelationsmatrix wurde die oben genannte Korrelationsfunktion mit den Parametern $\alpha = 0.125$ und $\beta = 0.249$ verwendet. Diese Parameter liefern eine gute Approximation für die Korrelationen des Bildes eines Kameramanns aus [HAB71]. Die den Eigenbildern zugeordneten Eigenwerte werden in der nachfolgenden Abbildung zeilenweise von links oben nach rechts unten kleiner.

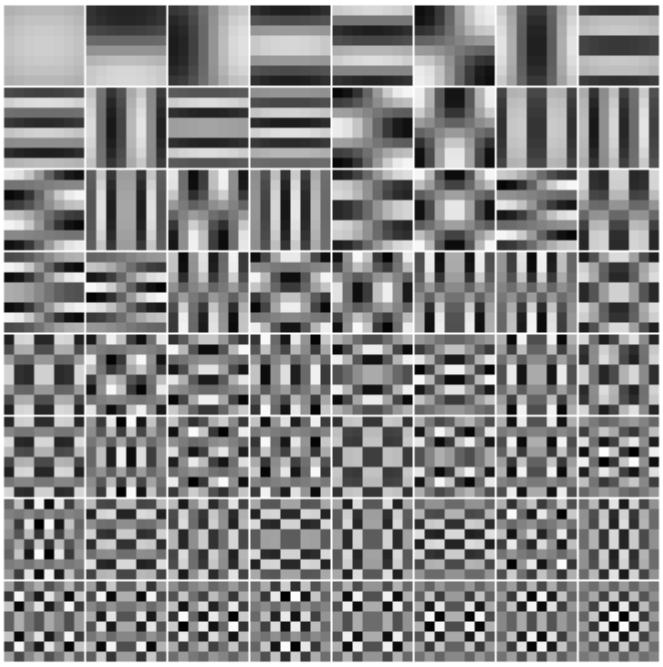


Abbildung 2-12 Visualisierte Eigenbilder mit 128 Graustufen einer 64×64 Korrelationsmatrix C_{XX}

Nach diesem Versuch stellte sich heraus, daß die Ergebnisse meiner Diplomarbeit und aus [WIN72] nicht identisch sind. Sie sind aber durchaus ähnlich. Aussagen über die Differenzen der Ergebnisse sind nur schwer möglich, da das Experiment aus [WIN72] nur sehr ungenau beschrieben wurde.

2.2.3. Quantisierung einer Punktmenge

Unter einer Quantisierung soll hier ein Verfahren verstanden werden, daß in der Lage ist, eine kontinuierliche Punktmenge Γ aus einem euklidischen Raum E^n in disjunkte Teilmengen zu zerlegen und anschließend für jede dieser Teilmengen einen Repräsentanten zu ermitteln. Dieser Repräsentant wird als Prototyp bezeichnet, da alle Punkte einer Teilmenge auf ihn abgebildet werden. Somit kann eine kontinuierliche Punktmenge Γ näherungsweise durch eine diskrete Anzahl von Prototypen beschrieben werden.

Als Beispiel für eine kontinuierliche Punktmenge Γ soll hier ein Bereich der reellen Zahlen $\Re$ dienen, wie in der folgenden Abbildung dargestellt.

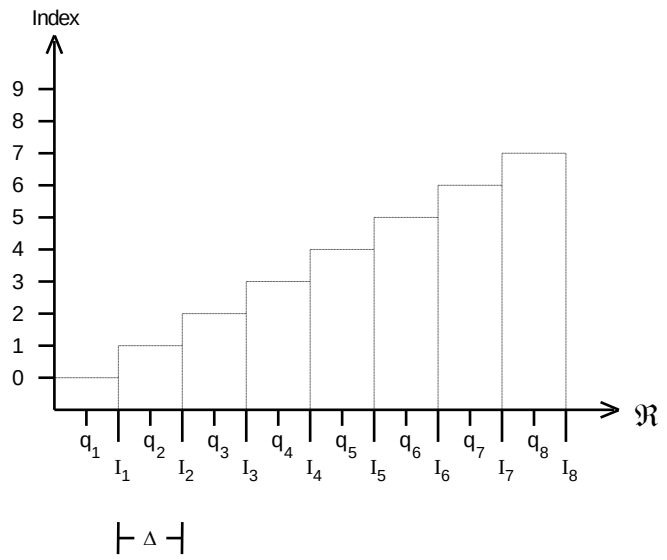


Abbildung 2-13 Quantisierung eines Bereichs der reellen Zahlen $\Re$

Um nun eine Quantisierung des reellwertigen Bereichs durchzuführen, wird dieser in disjunkte Intervalle I_1, I_2, …, I_8 aufgeteilt. Für jedes Intervall I_k wird ein Prototyp q_k gewählt, wobei alle Punkte eines Intervalls auf seinen Prototyp abgebildet werden sollen. Dabei ist die Wahl eines Prototyps nicht von den Intervallgrenzen abhängig, sondern nur von einem Ziel, das mit Hilfe der Wahl dieses Prototyps erreicht werden soll. In diesem Beispiel wurde willkürlich jeder Prototyp q_k als Mittelwert seines Intervalls I_k definiert. Jetzt kann der reellwertige Bereich näherungsweise durch eine diskrete Anzahl von Prototypen beschrieben werden.

Diese Prototypen werden im folgenden als Quantisierungslevel bezeichnet. Besitzen die Teilmengen der Quantisierungslevel alle die gleiche Größe Δ und sind alle Quantisierungslevel in der betrachteten kontinuierlichen Punktmenge Γ gleichverteilt, dann soll hier von einem uniformen Quantisierer gesprochen werden.

Existiert eine diskrete Menge von Werten aus dem reellwertigen Bereich des obigen Beispiels, die alle quantisiert und in binärer Form zwischengespeichert werden sollen (siehe Abbildung 2-9), dann muß eine Kodierung dieser Werte durchgeführt werden.

Dazu wird für jeden Wert ermittelt, in welchem Intervall I_1, I_2, …, I_8 er sich befindet. Liegt er im Intervall I_k, dann wird er, wie alle Werte aus I_k, auf den Quantisierungslevel q_k abgebildet. Es wäre nun aber nicht effizient für jeden Wert den zugehörigen Quantisierungslevel direkt abzuspeichern.

Vielmehr wird in der Praxis jedem Quantisierungslevel ein Index zugeordnet, indem alle Level nacheinander in ein sogenanntes Codebook gespeichert werden[11]. Für die Kodierung der Indizes 0, 1, …, n - 1 werden dann ld n Bits benötigt, im Fall des obigen Beispiels also ld 8 = 3 Bits.

Statt der Quantisierungslevel werden die zugeordneten Indizes gespeichert, wobei für jeden Index ld n Bits benötigt werden. Nach der binären Zwischenspeicherung kann mit Hilfe des Codebooks eine Dekodierung durchgeführt werden, indem für jeden Index der zugehörige Quantisierungslevel ermittelt wird.

Es zeigt sich, daß mit Hilfe einer Quantisierung nicht nur eine kontinuierliche Punktmenge Γ durch eine diskrete Anzahl von Quantisierungsleveln näherungsweise beschrieben werden kann, sondern daß damit eine fehlerbehaftete Datenreduktion einer diskreten Menge von Werten möglich ist.

Die Anwendung dieses Verfahrens auf die resultierenden Koeffizienten einer linearen Transformation (siehe Abbildung 2-9) und die Minimierung des dabei entstehenden Fehlers sind Gegenstand des Abschnitts Quantisierungs-Fehler und Datenkompression.

Für eine Vertiefung dieses Lehrstoffs wird an dieser Stelle auf weiterführende Literatur [JAY84], [WOD69], [HUA63], [MAX60] verwiesen.

[11] Wäre für das hier betrachtete Beispiel ein nicht-uniformer Quantisierer verwendet worden, dann würden statt der Quantisierungslevel die entsprechenden Intervallgrenzen nacheinander in das Codebook gespeichert.

2.2.4. Approximation von Teilbildern

In den vorangegangenen Kapiteln wurden Grundlagen der linearen Transformation und der Quantisierung vermittelt und deren Eigenschaften untersucht. Es stellt sich nun die Frage, wie eine lineare Transformation und eine Quantisierung zur Datenkompression eingesetzt werden können.

Bei der hier angestrebten Kompression handelt es sich um eine approximative (und damit fehlerbehaftete) Komprimierung von Daten, so daß auch die Qualität der dekomprimierten Daten eine wichtige Rolle spielt.

Für die Qualität von dekomprimierten Bilddaten stellt der mittlere quadratische Fehler zwischen den ursprünglichen Bilddaten und den dekomprimierten Bilddaten ein sehr brauchbares Gütekriterium dar, denn eine Minimierung dieses Fehlers bewirkt anscheinend auch eine Verbesserung des psycho-physiologischen Eindrucks, den ein menschlicher Beobachter von einem dekomprimierten Bild hat [HAB71].

Da das Verfahren des Transform Coding eine Sequenz der beiden Operationen

- Lineare Transformation
- Quantisierung von Koeffizienten

darstellt und jede der beiden Operationen bei einer Komprimierung von Bilddaten auch einen gewissen Fehler verursacht, setzt sich der resultierende mittlere quadratische Gesamtfehler ε^2 zusammen aus einem Sampling-Fehler ε_s^2 und einem Quantisierungs-Fehler ε_q^2. Damit ergibt sich

$$\varepsilon^2 = \varepsilon_s^2 + \varepsilon_q^2. \tag{2-48}$$

In den folgenden beiden Abschnitten werden die Datenkompression und der resultierende mittlere quadratische Fehler für beide Operationen des Transform Coding separat untersucht. Um die dabei betrachteten Sätze und Beweise zu vereinfachen, wird auf die Notation der eindimensionalen linearen Transformationen zurückgegriffen.

Sampling-Fehler und Datenkompression

In Kapitel 2.2.2. wurde bereits gezeigt, daß ein $n \times n$ Teilbild eines $N \times N$ Bildes in einem Vektor x mit n^2 Komponenten abgelegt, anschließend mit Hilfe einer orthogonalen linearen Transformation $y = A\,x$ abgebildet und dann mit der entsprechenden inversen Transformation $x = A^T y$ fehlerfrei rekonstruiert werden kann.

Die dabei verwendete inverse Transformation

$$x_k = \sum_{i=1}^{n^2} a_{ik}\, y_i \qquad k = 1, 2, \ldots, n^2 \tag{2-49}$$

kann auch umgeschrieben werden in

$$x = \sum_{i=1}^{n^2} y_i\, a^{iT}, \tag{2-50}$$

so daß sich eine Linearkombination von Basisvektoren

$$a^{iT} = a_{ik}\,_{k=1,2,\ldots,n^2} \qquad i = 1, 2, \ldots, n^2 \tag{2-51}$$

mit den Koeffizienten y_i $i = 1, 2, \ldots, n^2$ ergibt. Daher läßt sich ein Teilbild x durch eine gewichtete Summe von orthonormalen Basisvektoren a^{iT} aus den Zeilen der Transformationsmatrix A darstellen, wobei die Gewichte y_i nach Gleichung 2-24 gegeben sind durch

$$y_i = a^{iT} x \qquad i = 1, 2, \ldots, n^2 \tag{2-52}$$

und somit als Korrelation zwischen dem Teilbild x und einem Basisvektor a^{iT} interpretiert werden können.

Um nun eine Datenkompression durchzuführen, kann die Summierung in Gleichung 2-50 nach $m \le n^2$ Schritten abgebrochen werden, so daß die Koeffizienten y_i und Basisvektoren a^{iT} für $i > m$ nichts mehr zur Summe beitragen. Das entspricht einer Aufteilung des Bereichs der Koeffizienten in zwei Zonen, wobei eine Zone die beibehaltenen Koeffizienten und die andere Zone die vernachlässigten Koeffzienten enthält. Daher wird diese Methode auch als *zonal-filtering* oder *zonal-sampling* bezeichnet. Die Methode des *zonal-sampling* stellt eine nicht-adaptive Technik dar, bei der nur jene Koeffizienten beibehalten werden, die im Mittelwert die größte Energie besitzen [WIN72].

Wird die obige Methode auf Gleichung 2-50 angewendet, so erhält man statt eines fehlerfreien Teilbildes x ein approximatives Teilbild x'

$$x = \sum_{i=1}^{n^2} y_i\, a^{iT} \approx \sum_{i=1}^{m} y_i\, a^{iT} = x' \qquad m \le n^2. \tag{2-53}$$

Wie groß ist jetzt der durch diese Approximation verursachte mittlere quadratische Sampling-Fehler ? Diese Frage beantwortet der folgende Satz.

Satz 2-6: Es sei $y = A\, x$ eine lineare Transformation mit einer orthogonalen $n^2 \times n^2$ Matrix A. Jedes $n \times n$ Teilbild x sei durch die inverse Transformation

$$x = \sum_{i=1}^{n^2} y_i \, a^{iT}$$

gegeben. Zu jedem Teilbild x existiere ein approximatives Teilbild x', daß durch eine approximative inverse Transformation

$$x' = \sum_{i=1}^{m} y_i \, a^{iT} \qquad\qquad m \le n^2$$

bestimmt wird, indem die Koeffizienten y_i und Basisvektoren a^{iT} für $m + 1 \le i \le n^2$ vernachlässigt werden. Der durch

$$\varepsilon_s^2 = \left\langle \left| x - x' \right|^2 \right\rangle$$

definierte mittlere quadratische Sampling-Fehler ist dann gegeben durch

$$\varepsilon_s^2 = \sum_{i=m+1}^{n^2} \left\langle y_i^2 \right\rangle .$$

Gilt zusätzlich die Bedingung $\langle x \rangle = 0$, so folgt

$$\varepsilon_s^2 = \sum_{i=m+1}^{n^2} \sigma_{y_i}^2 \, .$$

Beweis 2-6:

$$\varepsilon_s^2 = \left\langle \left| x - x' \right|^2 \right\rangle$$

$$= \left\langle \left| \sum_{i=1}^{n^2} y_i \, a^{iT} - \sum_{i=1}^{m} y_i \, a^{iT} \right|^2 \right\rangle = \left\langle \left| \sum_{i=m+1}^{n^2} y_i \, a^{iT} \right|^2 \right\rangle = \left\langle \sum_{i=m+1}^{n^2} y_i \, a^{iT} , \sum_{j=m+1}^{n^2} y_j \, a^{jT} \right\rangle$$

$$= \left\langle \sum_{i=m+1}^{n^2} \sum_{j=m+1}^{n^2} y_i \, y_j \, a^{iT} , a^{jT} \right\rangle = \left\langle \sum_{i=m+1}^{n^2} y_i^2 \right\rangle = \sum_{i=m+1}^{n^2} \left\langle y_i^2 \right\rangle$$

Gilt zusätzlich die Bedingung $\langle x \rangle = 0$, dann folgt mit $y = A\,x$

$$\langle y \rangle = \langle A\,x \rangle = A \langle x \rangle = 0$$

und somit

$$\varepsilon_s^2 = \sum_{i=m+1}^{n^2} \left\langle y_i^2 \right\rangle = \sum_{i=m+1}^{n^2} \left\langle \left(y_i - \langle y_i \rangle \right)^2 \right\rangle = \sum_{i=m+1}^{n^2} \sigma_{y_i}^2$$ q.e.d.

Der mittlere quadratische Sampling-Fehler ε_s^2 ist also nur von den vernachlässigten Koeffizienten abhängig. Mit folgendem Satz wird nun gezeigt, wie die Basisvektoren in den Zeilen der Transformationsmatrix A gewählt werden müssen, damit dieser Fehler minimiert wird.

Satz 2-7: Es sei y = A x eine lineare Transformation mit einer orthogonalen $n^2 \times n^2$ Matrix A, die als Zeilenvektoren die orthonormierten Eigenvektoren $\{e^1, e^2, ..., e^{n^2}\}$ der Korrelationsmatrix C_{XX} enthält. Diese Eigenvektoren seien gemäß der Größe der zugehörigen Eigenwerte $\lambda_1, \lambda_2, \ldots, \lambda_{n^2}$ absteigend geordnet. Dann wird der durch

$$\varepsilon_s^2 = \left\langle \left| x - x' \right|^2 \right\rangle$$

definierte mittlere quadratische Sampling-Fehler minimal.

Beweis 2-7: (siehe Anhang 2)

Bezieht man die obigen Sachverhalte auf die Hotelling Transformation, dann lassen sich drei wichtige Folgerungen formulieren.

(i) Unter Berücksichtigung des mittleren quadratischen Sampling-Fehlers ε_s^2 sind die Eigenvektoren der Korrelationsmatrix C_{XX} für die Rekonstruktion eines Teilbildes x' unterschiedlich wichtig. Je größer der Eigenwert λ_i eines Eigenvektors e^i ist, desto mehr trägt dieser Eigenvektor im Mittelwert zur Rekonstruktion eines Teilbildes bei.

(ii) Die Eigenvektoren der Korrelationsmatrix C_{XX} und die Eigenvektoren der Kovarianzmatrix C_X eignen sich für unterschiedliche Zwecke. Während die Eigenvektoren der Korrelationsmatrix den mittleren quadratischen Sampling-Fehler ε_s^2 bei der Rekonstruktion eines Teilbildes x' minimieren (siehe Satz 2-7), liefern die Eigenvektoren der Kovarianzmatrix unkorrelierte Koeffizienten y_i (siehe Satz 2-5).

(iii) Existiert ein bildverarbeitendes Pre-Processing, welches in der Lage ist, die mittlere Intensität eines Bildes zu messen und die Intensität eines Teilbildes x anschließend passend zu skalieren, so kann vom Spezialfall $\langle x \rangle = 0$ ausgegangen werden. Dieser Spezialfall ist besonders günstig, da dann die Korrelationsmatrix C_{XX} und die Kovarianzmatrix C_X identisch sind. In diesem Fall minimieren die Eigenvektoren der Korrelationsmatrix den mittleren quadratischen Sampling-Fehler ε_s^2 und liefern unkorrelierte Koeffizienten y_i.

Da unter der Bedingung $\langle x \rangle = 0$ auch

$$\varepsilon_s^2 = \sum_{i=m+1}^{n^2} \left\langle y_i^2 \right\rangle = \sum_{i=m+1}^{n^2} \left\langle \left(y_i - \langle y_i \rangle \right)^2 \right\rangle = \sum_{i=m+1}^{n^2} \sigma_{y_i}^2 = \sum_{i=m+1}^{n^2} \lambda_i$$

folgt (siehe Beweis 2-7), ist es gleichgültig, ob man bei einer Rekonstruktion eines Teilbildes x' die Koeffizienten y_i mit der geringsten Varianz $\sigma_{y_i}^2$ vernachlässigt oder die Koeffizienten $y_i = e^{iT} x$, die mit Hilfe der Eigenvektoren e^i mit den kleinsten Eigenwerten λ_i berechnet wurden. Beide Fälle sind äquivalent und bewirken eine Minimierung des mittleren quadratischen Sampling-Fehlers.

Quantisierungs-Fehler und Datenkompression

Nach Kapitel 2.2.1. stellt das Transform Coding ein Verfahren dar, mit dem Bilder in einer Sequenz von zwei Operationen kodiert werden können. Ziel dieser beiden Operationen ist es, die Datenmenge eines Bildes zu reduzieren. Die Datenreduktion mit Hilfe einer linearen Transformation wurde im letzten Abschnitt ausführlich untersucht. Dieses Kapitel beschäftigt sich mit der Quantisierung jener Koeffizienten, die mit Hilfe einer linearen Transformation aus einem n × n Teilbild ermittelt wurden (siehe Abbildung 2-9).

Nachdem ein n × n Teilbild eines N × N Bildes in einem Vektor mit n^2 Komponenten abgelegt und in Koeffizienten y_k transformiert wurde, wird jeder dieser Koeffizienten quantisiert und kodiert, um eine weitere Datenkompression in der zweiten Stufe des Transform Coding zu erreichen (siehe Kapitel 2.2.3.).

Weil die bisher vorgestellten Transformationen Koeffizienten mit sehr unterschiedlichen Varianzen liefern [HAB71], wäre es nicht effizient denselben Quantisierer für alle Koeffizienten zu verwenden. Falls die Quantisierungslevel so festgelegt werden, daß sie den Bereich des Koeffizienten mit der größten Varianz abdecken, dann würden für die Koeffizienten mit viel geringeren Varianzen die meisten Quantisierungslevel nicht benutzt.

Da zudem nach [WIN72] die Koeffizienten mit größereren Varianzen im Mittelwert mehr zur Rekonstruktion eines Teilbildes beitragen, als die Koeffizienten mit geringeren Varianzen, scheint es, daß eine Verbesserung des psycho-physiologischen Eindrucks, den ein menschlicher Beobachter von einem rekonstruiertem Bild hat, erreicht werden kann, indem den Koeffizienten mit einer größeren Varianz proportional mehr Quantisierungslevel zugeteilt werden, als den Koeffizienten mit einer geringeren Varianz. Dies bedeutet, daß für Koeffizienten mit größeren Varianzen mehr Bits vergeben werden, als für Koeffizienten mit geringeren Varianzen, um die Indizes der zugeteilten Quantisierungslevel zu kodieren (siehe Kapitel 2.2.3.).

Es stellt sich nun die Frage, wie der durch die Quantisierung und Kodierung verursachte mittlere quadratische Quantisierungs-Fehler

$$\varepsilon_q^2 = \left\langle \left| y - y' \right|^2 \right\rangle \tag{2-54}$$

minimiert werden kann, der sich hier auf die resultierenden Koeffizientenvektoren y aller betrachteten n × n Teilbilder x bezieht. Dabei wird der quantisierte Wert eines Koeffizienten y_k mit y_k' bezeichnet. Für die Minimierung dieses Fehlers wurden in [WIN72] und [HAB71] eine Reihe von Vorschlägen gemacht, von denen im folgenden zwei kurz vorgestellt werden.

Huang und Schultheiss [HUA63] waren die ersten, die sich mit dem Problem beschäftigten, eine Anzahl von insgesamt M Bits an n^2 Koeffizienten mit unterschiedlichen Varianzen zu vergeben [HAB71]. Demnach soll die Anzahl der Bits m_k, die zur Kodierung des Koeffizienten y_k verwendet werden, proportional sein zu $\lg \sigma_{y_k}^2$, also

$$M = \sum_{i=1}^{n^2} m_i ,$$

$$m_k \sim \lg \sigma_{y_k}^2 \qquad k = 1, 2, \ldots, n^2. \qquad (2\text{-}55)$$

Es wurde ein Verfahren angegeben, um die Bitanzahl m_k zu berechnen, so daß der mittlere quadratische Quantisierungs-Fehler ε_q^2 für gegebenes M und $\sigma_{y_k}^2$ k = 1, 2, …, n^2 minimal wird.

Wintz und Kurtenbach [WIN68] gaben folgenden Algorithmus an, um insgesamt M Bits an n^2 Koeffizienten mit Varianzen $\sigma_{y_k}^2$ k = 1, 2, …, n^2 zu vergeben.

Berechne die n^2 Werte

$$m_k' = \frac{M}{n^2} + 2 \lg \sigma_{y_k}^2 - \frac{1}{n^2} \sum_{i=1}^{n^2} \lg \sigma_{y_i}^2 \qquad k = 1, 2, \ldots, n^2 \qquad (2\text{-}56)$$

und runde anschließend jedes m_k' zur nächsten Integerzahl m_k. Folgt anschließend

$$\sum_{i=1}^{n^2} m_i \neq M,$$

dann modifiziere einige willkürlich gewählte Werte m_k, bis die obige Summe genau M ergibt. Die hier verwendete Technik wird als Block-Quantisierung bezeichnet.

Nach [HAB71] gibt es einige Unterschiede zwischen den beiden hier vorgestellten Verfahren. So scheint es, daß Huang und Schultheiss den optimalen nicht-uniformen Quantisierer einsetzen, während Wintz und Kurtenbach den optimalen uniformen Quantisierer verwenden.

2.3. Neuronale Netze zur Datenkompression

Seit Anfang der achtziger Jahre ist es weltweit zu einem enormen Aufschwung der Technik Neuronaler Netze gekommen. Einer der Gründe für diesen Aufschwung ist, daß Neuronale Netze in ihrem Aufbau und ihrer Konzeption stärker an der Funktionsweise des menschlichen Gehirns orientiert sind, als an der Arbeitsweise konventioneller Rechner der klassischen von-Neumann-Architektur.

Daher können Neuronale Netze leichter dazu genutzt werden, wichtige Fähigkeiten des Menschen wie das

- Lernen aus Beispielen,
- Verallgemeinern von Beispielen,
- Abstrahieren,
- schnelle Erkennen und Vervollständigen komplizierter Muster,
- assoziative Speichern und Abrufen von Informationen

und viele andere mehr nachzubilden und zu simulieren. Menschen sind in der Lage in Sekundenbruchteilen komplizierte Signale und Bilder oder Ton- und Lautfolgen zu erkennen und inhaltlich zu interpretieren. Sie können ohne Mühen Personen wiedererkennen, die sie lange nicht gesehen haben, und die sich in der Zwischenzeit sehr verändert haben.

Jeder Computer konventioneller Bauart wäre mit solchen Aufgaben hoffnungslos überfordert, und das interessanterweise, obwohl die Daten in den heutigen Computern wesentlich schneller verarbeitet werden können als im menschlichen Gehirn. Worauf beruht die erstaunliche Leistungsfähigkeit des menschlichen Gehirns ?

Der Schlüssel zum Verständnis der Arbeitsweise des Gehirns und einer möglichen maschinellen Intelligenz liegt nicht in der Verarbeitungsgeschwindigkeit der Informationen, sondern in der hochgradig parallelen Natur der Informationsverarbeitung im Gehirn. Ein künstliches Neuronales Netz als DV-technisches Modell des Gehirns besteht folglich nicht aus einer komplexen Zentraleinheit (CPU) und einem Arbeitsspeicher (RAM) wie ein konventioneller Computer, sondern wie das Gehirn aus sehr vielen einfachen Rechenelementen, den Neuronen (Nervernzellen). Diese Neuronen sind hochgradig miteinander verknüpft und können dadurch gleichzeitig untereinander Informationen austauschen.

Indem die für das Lernen verantwortlichen synaptischen Veränderungen zwischen den Neuronen DV-technisch durch reelle Zahlen (Gewichte) und Lernregeln (Algorithmen zur Veränderung der Gewichte während der Lernphase) simuliert werden, können künstliche Neuronale Netze in einem gewissen Sinne aus Beispielen lernen.Aufgrund ihrer Lernfähigkeit müssen Neuronale Netze nicht mehr wie konventionelle Computer programmiert, sondern lediglich trainiert werden [SCE90].

In den folgenden Kapiteln werden die Grundbegriffe Neuronaler Netze dargestellt, wobei auf eine Einführung in die biologischen Grundlagen verzichtet wurde. Vielmehr wurde Wert darauf gelegt, künstliche Neuronale Netze als DV-technisches Modell einzuführen. Dabei wird der Schwerpunkt auf den Einsatz künstlicher Neuronaler Netze im Bereich der Komprimierung von Bilddaten gelegt.

2.3.1. Grundbegriffe

Im Unterschied zur traditionellen Unterteilung eines Computers in Hardware und den ausgeführten Algorithmen, der Software, lassen sich bei den Neuronalen Netzen diese beiden Aspekte nicht streng voneinander trennen. Die Hardware und der Algorithmus bilden eine Einheit, wobei die Hardwarearchitektur den Algorithmus implementiert. Eine Anpassung dieses allgemeinen Algorithmus an eine spezielle Aufgabe wird durch Eingaben von Trainingsmustern erreicht.

Ein Neuronales Netz soll hier als ein System betrachtet werden, das n Eingänge und m Ausgänge besitzt. An den Eingängen liegt ein Eingabevektor und an den Ausgängen ein Ausgabevektor an.

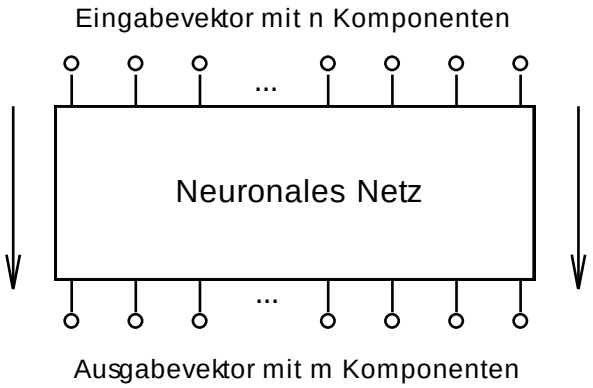


Abbildung 2-14 Modell eines Neuronalen Netzes mit n Eingängen und m Ausgängen

Das Neuronale Netz läßt sich in eine bestimmte Anzahl von Schichten unterteilen. Jede einzelne Schicht besitzt eine Reihe von Eingängen und Ausgängen, durch die sie mit anderen Schichten verbunden wird, so daß man die Informationsverarbeitung in einem Neuronalen Netz als pipeline-artige Folge von informationsverarbeitenden Schichten auffassen kann.

Werden die Eingänge einer Schicht ausschließlich von den Ausgängen der Schicht zuvor gespeist, dann wird ein solches Netz als *Feedforward-Netz* bezeichnet. Existieren Verbindungen zwischen den Ausgängen einer Schicht und den Eingängen von Vorgänger-Schichten, dann wird das Neuronale Netz als *Feedback-Netz* bezeichnet.

Der hier beschriebene Informationsfluß bezieht sich ausschließlich auf ein Neuronales Netz in der Funktionsphase. In diesem Modus werden bereits trainierten Netzen Eingabedaten in Form von Eingabevektoren präsentiert. Das Netz wird dadurch veranlaßt, auf Basis der Erfahrung, die im Training gesammelt wurde, einen Ausgabevektor zu generieren.

Der Informationsfluß in der Lernphase eines Netzes ist dagegen nicht festgelegt und geht meist nicht in die Bezeichnung eines Neuronalen Netzes ein. Öfters wird die Information von den Ausgängen des Netzes zu den Eingängen einzelner Schichten zurückgeführt und stellt damit eine Rückkopplung des Neuronalen Netzes in der Lernphase dar. Die Struktur eines Netzes in der Lernphase und in der anschließenden Funktionsphase kann also unterschiedlich sein.

Die einzelnen Schichten bestehen aus einer Anzahl von Prozessorelementen, den Neuronen. Das Grundmodell eines Neurons entspricht hier im Wesentlichen dem Modell von McCulloch und Pitts aus dem Jahre 1943, die ein Neuron als eine Art Addierer mit Schwellwert betrachteten [BRA91]. Die Verbindungen eines Neurons nehmen eine Aktivierung x_i mit einer bestimmten Stärke w_i von anderen Neuronen auf, summieren diese und lassen dann am Ausgang y des Neurons eine Aktivität entstehen, sofern die Summe vorher einen Schwellwert T überschritten hat [BRA91].

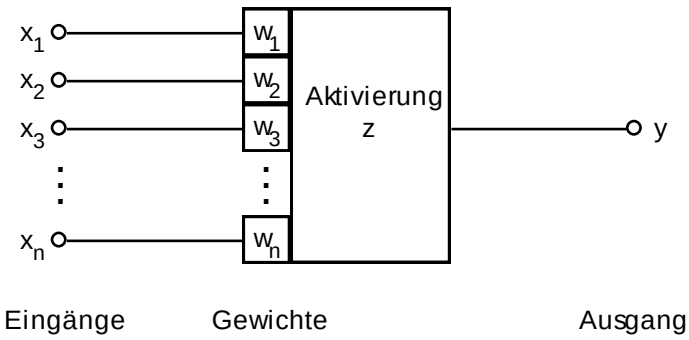


Abbildung 2-15 Modell eines Neurons nach McCulloch und Pitts

Alle in der obigen Abbildung dargestellten Größen werden durch reelle Zahlen beschrieben. Weiterhin werden hier im Gegensatz zu McCulloch und Pitts die einzelnen Gewichte w_i nicht als gleich angenommen. Zur Vereinfachung der mathematischen Gleichungen werden die Eingaben $\{x_1, x_2, \ldots, x_n\}$ und die Gewichte $\{w_1, w_2, \ldots, w_n\}$ zu Vektoren x und w zusammengefaßt. Jedes Neuron verarbeitet eine Eingabe an den Eingängen nach einem bestimmten Schema. Zuerst werden die Eingaben mittels einer Aktivitätsfunktion z(.) bearbeitet. Gewöhnlich ist dies die einfache Summation der Eingaben x_i multipliziert mit den entsprechenden Gewichten w_i, was formal dem Standard-Skalarprodukt der beiden Vektoren x und w entspricht

$$z(w, x) := \sum_{i=1}^{n} w_i x_i = w^T x. \qquad (2\text{-}57)$$

Sehr oft muß die Aktivität erst einen Schwellwert T überschreiten, bevor sie sich beim Ausgang des Neurons auswirkt. Dies läßt sich nach [BRA91] durch die Minderung der Aktivität um den Schwellwert modellieren, so daß

$$z(w,x) = w^T x - T. \tag{2-58}$$

Ergänzt man allerdings die Vektoren x und w um eine weitere Komponente

$$x^T = (x_1, x_2, \ldots, x_n, 1),$$

$$w^T = (w_1, w_2, \ldots, w_n, -T), \tag{2-59}$$

so erhält man wieder die Notation mit dem Standard-Skalarprodukt aus Gleichung 2-57.

Das Ergebnis der Aktivitätsfunktion z(.) wird an eine Ausgabefunktion S(.) weitergegeben

$$y := S(z), \tag{2-60}$$

so daß man die gesamte Reaktion eines Neurons auf eine Eingabe auch als Ergebnis nur einer Funktion, der Transferfunktion

$$y = f(x, w, z, S) \tag{2-61}$$

auffassen kann [BRA91].

Da die Aktivitäten in Neuronalen Netzen nicht konstant sind, sondern sich mit der Zeit ändern, sind die Größen x, w und y abhängig von einem Zeitschritt t. Daher wird Gleichung 2-57 modifiziert zu

$$z(t) = w^T(t-1)\, x(t). \tag{2-62}$$

Wird eine lineare Ausgabefunktion y = S(z) = z verwendet, dann erhält man

$$y(t) = w^T(t-1)\, x(t), \tag{2-63}$$

wobei im folgenden zur Vereinfachung der Schreibweise die Zeitschritte für die Größen x und y nicht mehr angegeben werden.

Viele Modelle Neuronaler Netze sind zeitkontinuierlich und werden durch Differenzialgleichungen beschrieben. Da aber alle in den folgenden Kapiteln vorgestellten Modelle durch zeitdiskrete Formalismen beschrieben werden können, wird hier auf den Aspekt der zeitkontinuierlichen Neuronalen Netze nicht eingegangen.

Befindet sich ein Neuron in der Lernphase, dann werden dessen Gewichte verändert, und zwar in der Regel so lange, bis eine Zuordnung der Eingabemuster zu einer gewünschten Ausgabe stattgefunden hat, daß heißt gelernt wurde. Das Verändern der Gewichte geschieht mit Hilfe einer Lernregel. Es gibt eine Vielzahl von unterschiedlichen Lernregeln, die jeweils auf ein bestimmtes Problem angewendet werden.

In den folgenden Kapiteln werden nun Modelle vorgestellt, die als Grundlage für eine Komprimierung von Bilddaten mit Hilfe Neuronaler Netze betrachtet werden können. Die dabei verwendeten Ausgabefunktionen sind stets linear, also y = S(z) = z. Weitere Beispiele für Aktivitäts- und Ausgabefunktionen sind in [BRA91] zu finden.

2.3.2. Die Lernregeln von Hebb und Oja

Im Jahr 1943 veröffentlichten McCulloch und Pitts eine Arbeit, die zeigte, das jede aussagenlogische Funktion mittels eines Neuronalen Netzes von einfachen binären Schwellwertelementen, den nach ihnen benannten McCulloch-Pitts-Neuronen, simuliert werden kann. Die McCulloch-Pitts-Neuronen hatten jedoch den Nachteil, daß Sie starr miteinander verbunden waren. Modelle dieses Typs waren daher nicht lernfähig [SCE90].

Im Jahr 1949 veröffenlichte Hebb eine grundlegende Arbeit und erläuterte explizit, wie man Lernvorgänge physiologisch erklären kann. Die wichtige Idee von Hebb war, das Lernen auf die physiologische Veränderung von Nervenzellen und deren Verbindung zurückzuführen. Er formulierte die nach ihm benannte Hebbsche Lernregel, welche sinngemäß nach [SCE90] lautet:

> Ist der Ausgang einer Nervenzelle nahe genug an einer zweiten Nervenzelle, um sie zu erregen oder zu ihrer Erregung beizutragen, so führt eine zeitliche Koinzidenz ihrer (positiven) Aktivitäten zu einer Veränderung (Wachstum, Stoffwechselveränderung) einer oder beider Zellen, so daß die Effektivität des Einflusses der ersten auf die zweite Zelle sich erhöht.

Diese Aussage läßt sich auch mathematisch formulieren und auf zwei hier betrachtete künstliche Neuronen A und B anwenden. Wird die Änderung der Effektivität als Differenz der Gewichte von B zu den Zeitpunkten t und t-1 interpretiert, dann läßt sich das Anwachsen des Gewichts w_{AB} zwischen den Neuronen A und B als Produkt der Ausgaben von A und B schreiben. Damit ergibt sich

$$\Delta w_{AB}(t) = w_{AB}(t) - w_{AB}(t-1) = x_A \, y_B. \tag{2-64}$$

Mit einer zeitabhängigen Lernrate $\gamma(t)$ läßt sich die Hebbsche Lernregel schreiben als

$$\Delta w(t) = \gamma(t) x y \tag{2-65}$$

und in einer iterativen Form für das i-te Neuron als

$$w^i(t+1) = w^i(t) + \gamma(t)\, y(t)\, x_i(t) \qquad i = 1, 2, \ldots, m. \qquad (2\text{-}66)$$

Es zeigt sich jedoch, daß bei häufiger hoher Aktivität beider Neuronen die Gewichte im Laufe der Zeit ins unendliche wachsen. Da diese Tatsache nach [BRA91] auch biologisch nicht erklärbar ist, wird angenommen, daß zusätzlich zur Hebbschen Lernregel ein Normierungsmechanismus für die Gewichte vorhanden ist, so daß

$$|\hat{w}(t)| = 1 \quad \Leftrightarrow \quad \sum_{j=1}^{n} \hat{w}_j^2(t) = 1,$$

wobei mit $\hat{w}(t)$ die normierten Gewichte bezeichnet werden. Mit

$$\hat{w}_j^2(t) = \frac{w_j(t)}{|w(t)|} \qquad j = 1, 2, \ldots, n \qquad (2\text{-}67)$$

und durch Einsetzen von Gleichung 2-66 in Gleichung 2-67 und einer Tayler-Entwicklung der Funktion $f(\gamma) = 1 / |w(t)|$ unter anschließender Vernachlässigung der Summanden mit $\gamma^2(t)$ und höheren Potenzen erhält man nach [OJA82] eine Lernregel, die nicht nur die Hebbsche Lernregel beinhaltet, sondern auch eine Normierung der Gewichte durchführt:

$$w(t+1) = w(t) + \gamma(t)\, y(t)\, [x(t) - y(t)\, w(t)]. \qquad (2\text{-}68)$$

Diese Lernregel wurde 1982 von Oja vorgeschlagen. Bei einer Untersuchung des Konvergenzziels der Lernregel in [OJA82] wurde ein erstaunliches Ergebnis gefunden. Demnach liefert obige Lernregel für $t \rightarrow \infty$ als Gewichtsvektor den normierten Eigenvektor zum größten Eigenwert der Korrelationsmatrix C_{XX} der Eingaben des Neurons, falls die Matrix C_{XX} positiv semidefinit ist.

Werden nun mehrere Neuronen verwendet und geeignet miteinander gekoppelt, dann kann ein solches Neuronales Netz alle Eigenvektoren der Korrelationsmatrix C_{XX} als Gewichte lernen und wäre mit einer linearen Ausgabefunktion y = z in der Lage, eine Dekomposition einer Eingabemenge in eine Menge von unkorrelierten Koeffizienten bezüglich einer orthonormierten Eigenvektorbasis durchzuführen (siehe auch Abschnitt Hotelling Transformation), sofern die Bedingung $\langle x \rangle = 0$ erfüllt ist.

2.3.3. Eigenvektorzerlegung

In Kapitel 2.2.4. wurde gezeigt, daß mit Hilfe einer orthogonalen linearen Transformation $y = A\ x$ eine fehlerbehaftete Datenkompression durchgeführt werden kann, wenn geeignete Koeffizienten y_k bei der inversen Transformation vernachlässigt werden. Dort wurde weiterhin bewiesen, daß der mittlere quadratische Sampling-Fehler ε_s^2 minimal wird, wenn die Transformationsmatrix A die orthonormierten Eigenvektoren der Korrelationsmatrix C_{XX} als Zeilenvektoren enthält. Diese Eigenvektoren müssen gemäß der Größe der zugehörigen Eigenwerte absteigend in A geordnet sein.

Wird nun das Modell eines einschichtigen Feedforward-Netzes aus Abbildung 2-14 mit einem geeigneten Lernverfahren kombiniert, dann lassen sich mit der linearen Aktivierung $y = W\ x$ die orthonormierten Eigenvektoren der Korrelationsmatrix C_{XX} als Gewichte lernen. Dabei repräsentiert der Vektor x den Eingabevektor und der Vektor y den Ausgabvektor des Neuronalen Netzes. Bei der Größe W handelt es sich um die Gewichtsmatrix, welche die Gewichtsvektoren aller Neuronen als Zeilenvektoren enthält.

Mit Hilfe eines solchen Neuronalen Netzes lassen sich abhängige Eingabekomponenten x_k in unkorrelierte Koeffizienten y_k transformieren, falls $\langle x \rangle = 0$. Die Darstellung von Daten als unkorrelierte Koeffizienten mit Hilfe einer verbesserten Beschreibungsbasis wird auch als Hauptkomponentenanalyse bezeichnet.

In den folgenden Abschnitten werden mehrere Lernverfahren vorgestellt, die alle Eigenvektoren der Korrelationsmatrix C_{XX} als Gewichte lernen und damit eine vollständige Eigenvektorzerlegung durchführen können. Dabei werden zwei Klassen von Lernverfahren unterschieden, nämlich die *ungeordnete* und die *geordnete* Zerlegung.

Ungeordnete Zerlegung

Bei der ungeordneten Zerlegung handelt es sich um eine Klasse von Lernverfahren für Neuronale Netze, bei denen die Wechselwirkungen aller Neuronen auf die Gewichte in der Lernphase völlig symmetrisch sind. Daher ist das Lernziel jedes Gewichtsvektors nicht fest vorbestimmt, sondern von seinen initialen Werten und der Lernregel abhängig.

Das Subspace-Netzwerk

Ein einfaches Modell für ein Neuronales Netz, das eine Hauptkomponentenanalyse durchführen kann, wurde von Oja entwickelt und als Subspace-Netzwerk bezeichnet [OJA89]. Dieses Modell stellt eine Erweiterung der in Gleichung 2-68 angegebenen Lernregel von Oja für ein einzelnes Neuron auf ein Neuronales Netz mit m Neuronen dar.

Sind nun m Neuronen in einer Schicht angeordnet, welche alle dieselbe Eingabe x erhalten, dann lautet die Lernregel für das i-te Neuron

$$w^i(t) = w^i(t-1) + \gamma\, y_i(x - z) \qquad i = 1, 2, \ldots, m. \qquad (2\text{-}69)$$

Wird als negativer inhibitorischer Korrekturterm z_j für eine Eingabekomponente x_j nicht nur der Beitrag $w_j^i(t\text{-}1)\ y_i$ des eigenen Neurons wie in Gleichung 2-68 eingesetzt, sondern auch die Beiträge aller anderen Neuronen in der Schicht, dann wird ein Neuron gehindert eine Gewichtskomponente auszubilden, falls andere Neuronen bereits starke Ausprägungen für diese Gewichtskomponente besitzen. Ist also ein Neuron bereits sehr *empfindlich* für eine Eingabekomponente x_j, so hindert es die anderen Neuronen daran, ebenfalls *empfindlich* für diese Komponente zu werden [BRA91].

Der inhibitorische Korrekturterm ist dann gegeben durch

$$z_j = \sum_{i=1}^{m} w_j^i(t-1)\, y_i \qquad j = 1, 2, \ldots, n \qquad (2\text{-}70)$$

oder in der Vektorschreibweise

$$z = \sum_{i=1}^{m} w^i(t-1)\, y_i. \qquad (2\text{-}71)$$

Es zeigt sich, daß die Wechselwirkungen der Neuronen auf die Gewichte in der Lernphase völlig symmetrisch sind und damit das Lernziel jedes Gewichtsvektors von seiner Anfangsbelegung abhängig ist.

Jedoch liegt das Lernziel dieses Neuronalen Netzes nicht unbedingt bei den orthonormierten Eigenvektoren der Korrelationsmatrix C_{XX}. Krogh und Hertz zeigten in [KRO90], daß das Subspace-Netzwerk von Oja eine orthonormierte Basis als Lernziel besitzt, die man durch eine Rotation der orthonormierten Eigenvektorbasis von C_{XX} erhalten kann. Damit liegt das Lernziel des Netzes bei einer beliebigen orthonormierten Basis, die denselben *Subspace* aufspannt wie die orthonormierte Eigenvektorbasis, welche durch die Korrelationsmatrix C_{XX} festgelegt ist.

Eine neue Version des Modells wurde 1992 von Oja, Ogawa und Wangviwattana vorgeschlagen [OJA92] und als *gewichtetes Subspace-Netzwerk* bezeichnet. Die neue Lernregel ist gegeben durch

$$w^i(t) = w^i(t-1) + \gamma\, y_i(x - \theta_i z) \qquad i = 1, 2, \ldots, m \qquad (2\text{-}72)$$

und entspricht bis auf den skalaren Parameter θ_i genau der Lernregel aus Gleichung 2-69. Werden all diese Parameter unterschiedlich und positiv gewählt

$$0 < \theta_1 < \theta_2 \ldots < \theta_m,$$

dann konvergieren nach [OJA92] die Gewichtsvektoren w^1, w^2, …, w^m zu den wahren Eigenvektoren der Korrelationsmatrix C_{XX}. Allerdings zerstören diese zusätzlichen Parameter die Symmetrie der Lernregel aus Gleichung 2-69.

Geordnete Zerlegung

Im letzten Abschnitt wurde eine Klasse von Lernverfahren betrachtet, bei denen die Wechselwirkungen aller Neuronen auf die Gewichte in der Lernphase symmetrisch sind. Dadurch hängt das Lernziel jedes Gewichtsvektors von seinen initialen Werten und der Lernregel ab. Sollen die Gewichtsvektoren aber so gelernt werden, daß ihre Gewichtsmatrix W der orthogonalen Transformationsmatrix A aus Kapitel 2.2.4. entspricht, dann muß eine Eigenvektorzerlegung derart durchgeführt werden, daß sich die Eigenvektoren gemäß der Größe ihrer zugehörigen Eigenwerte geordnet ergeben.

Die generalisierte Hebb-Regel

Ein Lernverfahren, welches eine geordnete Eigenvektorzerlegung durchführt, wurde von Sanger [SAN89] entwickelt. Es verwendet die Idee des Gram-Schmidt'schen Orthogonalisierungsverfahrens zu einer gegebenen Menge von orthogonalen Vektoren einen weiteren orthogonalen Vektor zu finden.

Verwendet man nun ein einschichtiges Feedforward-Netz aus Abbildung 2-14 und nimmt für jedes Neuron die generalisierte Hebbsche Lernregel

$$w^i(t+1) = w^i(t) + \gamma\, y^i(t)\,\big(x(t) - \tilde{x}^i(t)\big) \qquad i = 1, 2, \ldots, m \qquad (2\text{-}73)$$

als Lernverfahren an, bei der das i-te Neuron eine gefilterte Eingabe

$$\tilde{x}^i := x - \sum_{k=1}^{i} y_k\, w^k(t) \qquad i = 1, 2, \ldots, m \qquad (2\text{-}74)$$

erhält, dann konvergieren nach [SAN89] die Gewichtsvektoren w^1, w^2, …, w^m zu den wahren Eigenvektoren der Korrelationsmatrix C_{XX}. Dabei sind die Eigenvektoren gemäß der Größe ihrer zugehörigen Eigenwerte absteigend geordnet.

Dieses Modell ist durch seine asymmetrische Formulierung für biologische Anwendungen weniger plausibel, besitzt aber für die Komprimierung von Bilddaten größere praktische Bedeutung.

Die Anti-Hebb Regel

Die Lernverfahren der vorgestellten Modelle von Oja und Sanger verwenden Neuronen, die interne Informationen (Gewichte) anderer Neuronen in einem Neuronalen Netz für den eigenen Lernprozeß benötigen. Dies ist eine biologisch nicht sehr einleuchtende Annahme.

Ein Modell, welches stärkere Bezüge zu neurobiologischen Beobachtungen besitzt, wurde von Rubner, Schulten und Tavan entwickelt [RUB90]. Dieses Modell beruht auf einem einschichtigen Feedforward-Netz aus Abbildung 2-14, wobei für das erste Neuron die Hebbsche Lernregel mit anschließender Normierung der Gewichte verwendet wird. Nach Kapitel 2.3.2. lernt dieses Neuron den normierten Eigenvektor zum größten Eigenwert der Korrelationsmatrix C_{XX} als Gewicht.

Alle weiteren Neuronen erhalten ebenfalls die gleiche ungefilterte Eingabe x und verwenden die Hebbsche Lernregel mit anschließender Normierung der Gewichte. Der Ausgang des ersten Neurons unterdrückt über einen zusätzlichen gewichteten Eingang zum zweiten Neuron alle Korrelationen mit seiner Ausgabe, so daß das zweite Neuron daran gehindert wird, denselben Eigenvektor wie das erste Neuron zu lernen. Wird dieser Gedanke weitergeführt, dann erhält jedes Neuron i einseitig die gewichtete Ausgabe y_k aller vorherigen Neuronen $k < i$ als Gegenkopplung. Diese Art der Kopplung wird in der Biologie als laterale Inhibition bezeichnet und ist durchaus in vielen Nervensystemen anzutreffen, allerdings in vollsymmetrischer Anordnung [BRA91].

Die Ausgabe des i-ten Neurons ist dann gegeben durch

$$y_i = w^{iT} \left(x - \sum_{k=1}^{i-1} u_{ik} \left(1 \; y_k \right) \right) \qquad i = 1, 2, \ldots, m, \qquad (2\text{-}75)$$

wobei es sich bei der Größe u_{ik} um das laterale Gewicht zwischen den Neuronen i und k handelt. Werden nun die Gewichte w^i mit der Hebbschen Lernregel

$$w^i \left(t + 1 \right) = w^i \left(t \right) + \gamma_i \left(t \right) y_i \left(t \right) x \qquad i = 1, 2, \ldots, m \qquad (2\text{-}76)$$

und die lateralen Gewichte u_{ik} mit der Anti-Hebb Lernregel

$$u_{ik} \left(t + 1 \right) = u_{ik} \left(t \right) - \gamma_i \left(t \right) y_i y_k \qquad i = 1, 2, \ldots, m; \; k < i \qquad (2\text{-}77)$$

verändert, dann werden nach [RUB90] die Ausgaben y_i der Neuronen unkorreliert und die lateralen Gewichte u_{ik} konvergieren gegen Null. Zudem konvergieren die Gewichtsvektoren w^1, w^2, …, w^m zu den wahren Eigenvektoren der Korrelationsmatrix C_{XX} und sind gemäß der Größe der zugehörigen Eigenwerte absteigend geordnet [BRA91].

2.3.4. Das Modell von Silva und Almeida

Die bisher vorgestellten Modelle für Neuronale Netze beruhen auf iterativen Lernverfahren, die als Lernziel ein orthogonales Transformationskernel wie in Kapitel 2.2.2. besitzen. Mit Hilfe dieses Transformationskernels können die Neuronalen Netze für eine fehlerbehaftete Datenkompression eingesetzt werden. In den Kapiteln 2.1.2. und 2.1.3. wurde aber gezeigt, daß eine Datenorthonormalisierung bei der Kompression und Übertragung von Informationen wesentliche Vorteile bietet.

Die in Kapitel 2.3.3. beschriebenen Neuronalen Netze führen aber keine Orthonormalisierung der Eingabedaten durch. Sie liefern lediglich eine dekorrelierte Ausgabemenge. Ein recht populäres Modell zur Datenorthonormalisierung wurde 1991 von F. M. Silva und L. B. Almeida in [SIL91]vorgestellt.

Dieses Modell beruht auf einem einschichtigen Feedforward-Netz aus Abbildung 2-14 und besitzt eine lineare Aktivierung y = W x. Allerdings besitzen die Ein- *und* Ausgabevektoren jeweils n Komponenten. Die i-te Ausgabekomponente erhält man durch

$$y_i = w^{iT} x \qquad i = 1, 2, \ldots, n$$

und für die Korrelation zwischen der j-ten und k-ten Ausgabekomponente gilt

$$r_{jk} := \langle y_j y_k \rangle$$

$$= \langle w^{jT} x x^T w^k \rangle = w^{jT} \langle x x^T \rangle w^k = w^{jT} C_{XX} w^k \qquad j, k = 1, 2, \ldots, n, \qquad (2\text{-}78)$$

wobei es sich bei w^i um den i-ten Zeilenvektor der Gewichtsmatrix W handelt. Es ist nun das Ziel Gewichtsvektoren w^j und w^k zu finden, so daß die Komponenten y_j und y_k dekorreliert sind. Nach [SIL91] wird dazu die normierte Korrelation zwischen y_j und y_k betrachtet.

Ist diese nahe bei 1, dann ist der Winkel zwischen w^j und w^k klein und muß vergrößert werden. Ist sie nahe bei -1, dann ist der Winkel zwischen w^j und w^k bei π und muß verkleinert werden. Dies kann durch das folgende Modell erreicht werden, wo jeder Gewichtsvektor um einen kleinen Betrag in Richtung parallel zum anderen Vektor verschoben wird.

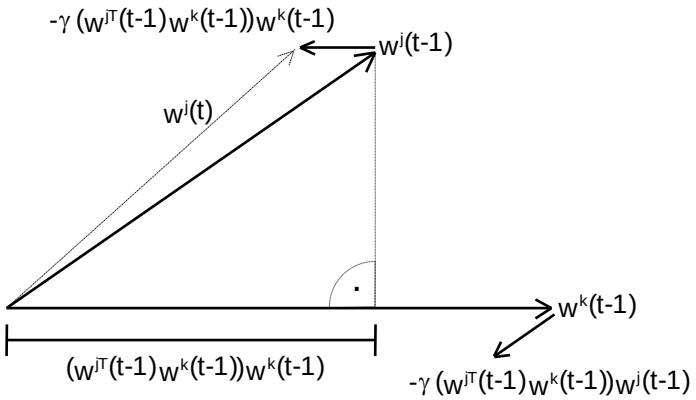


Abbildung 2-16 Vektor-Orthogonalisierung nach einem Modell von Silva und Almeida

Nach [SIL91] kann in diesem Fall r_{jk} als ein Skalarprodukt von w^j und w^k in einem Raum mit der Metrik C_{XX} angesehen werden. Eine Orthogonalisierung muß unter Berücksichtigung dieser Metrik durchgeführt werden.

Demnach kann eine Orthogonalisierung von w^j in Bezug auf w^k erreicht werden durch

$$w^j(t+1) = w^j(t) - \gamma\, w^k(t)^T C_{XX}\, w^k(t)$$

$$= w^j(t) - \gamma\, r_{jk}\, w^k(t) \qquad j, k = 1, 2, \ldots, n. \qquad (2\text{-}79)$$

Für den verallgemeinerten Fall von n Gewichtskomponenten erhält man

$$w^j(t+1) = w^j(t) - \gamma \sum_{k=1, k\neq j}^{n} r_{jk}\, w^k(t) \qquad j = 1, 2, \ldots, n. \qquad (2\text{-}80)$$

Da eine Normalisierung der Daten ebenfalls wünschenswert ist, wird die obige Gleichung in [SIL91] modifiziert. Sie beinhaltet zusätzlich einen kleinen positiven oder negativen Anteil des adaptierten Vektors $w^j(t)$ selbst. Dieser Anteil ist abhängig davon, ob die Größe r_{jj} größer oder kleiner als 1 ist.

$$w^j(t+1) = w^j(t) - \gamma \sum_{k=1, k\neq j}^{n} r_{jk}\, w^k(t) - \gamma\, r_{jj}\, w^j(t)$$

$$= w^j(t) - \underbrace{\gamma \sum_{k=1, k\neq j}^{n} r_{jk}\, w^k(t)}_{\text{Orthogonalisierung}} \underbrace{- \gamma\, w^j(t) - \gamma\, r_{jj}\, w^j(t)}_{\text{Normierung}}$$

$$= (1-\gamma)\, w^j(t) - \gamma \sum_{k=1}^{n} r_{jk}\, w^k(t) \qquad j = 1, 2, \ldots, n \qquad (2\text{-}81)$$

Falls bestimmte Bedingungen eingehalten werden, besitzen nach [SIL91] die Gewichtsvektoren aus dem obigen iterativen Lernverfahren ein Konvergenzziel derart, daß die Korrelationsmatrix der Ausgabemenge sich darstellen läßt als $C_{YY} = I$.

In Kapitel 3. meiner Diplomarbeit wird ein selbständig erarbeitetes Modell zur Datenorthonormalisierung vorgestellt und gezeigt, daß das Transformationskernel W für eine Datenorthonormalisierung nicht orthogonal ist und auch keine Eigenvektoren in den Zeilen enthalten muß.

3. Modell für informationsoptimale Kompression

In diesem Kapitel wird mit Hilfe der Ergebnisse aus den Abschnitten

- Informationstheorie
- Transform Coding
- Neuronale Netze

ein von mir entwickeltes Modell für ein Neuronales Netz vorgestellt, mit dessen Hilfe eine informationsoptimale Kompression gestörter Daten durchgeführt werden kann.

Was ist dabei unter dem Begriff der *informationsoptimalen Kompression* zu verstehen ? Zur Klärung dieses Begriffs möchte ich zunächst auf das Modell der Informationsübertragung in Abbildung 2-7 aus Kapitel 2.1.3. zurückgreifen.

In diesem Modell wird eine Übertragung von Signalen über einen gestörten Übertragungskanal beschrieben. Vor der eigentlichen Datenübertragung können die Signale mit Hilfe einer linearen Transformation derart kodiert werden, daß durch die Störungen im Übertragungskanal möglichst wenig Information vom ursprünglichen Signal verloren geht.

Eine lineare Transformation bildet nach Kapitel 2.2. aber auch die erste Stufe für das Transform Coding, einem Verfahren der Signalkodierung, mit dem eine Datenkompression durchgeführt werden kann.

In der Realität müssen Signale oftmals *vor* ihrer Übertragung über ein Medium passend kodiert und komprimiert werden. Eine Sequenz der Operationen Kodierung, Übertragung und Dekodierung wird in der folgenden Abbildung schematisch dargestellt.

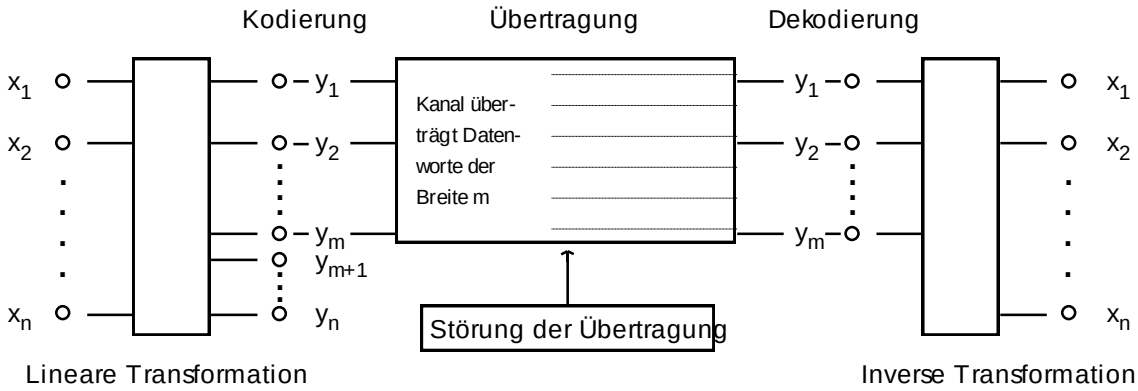


Abbildung 3-1 Schema einer Signalkodierung mit anschließender Übertragung über einen Kanal

Die Kodierung erfolgt dabei mit Hilfe einer linearen Transformation, welche die Eingabemenge X in eine Menge von stochastisch unabhängigeren Koeffizienten Y transformiert (siehe Kapitel 2.2.2.).

Die Datenkompression wird gemäß dem Transform Coding durch eine Vernachlässigung geeigneter Koeffizienten erreicht, die bei einer Dekodierung nicht mehr verwendet werden (siehe Kapitel 2.2.4.).

Eine Kompression soll hier als *informationsoptimal* bezeichnet werden, wenn die zu komprimierenden Daten einen maximalen mittleren Informationsgehalt besitzen und bei einer anschließenden Übertragung über einen gestörten Übertragungskanal nur einen minimalen Informationsverlust durch Störungen aufweisen.

In Kapitel 2.1. wurde gezeigt, daß die oben genannten Bedingungen für eine informationsoptimale Kompression für die sehr wichtige Klasse der normalverteilten Signale erfüllt sind, wenn die Kodierung dieser Signale durch eine Datenorthonormalisierung erfolgt. Daher beschäftigt sich dieses Kapitel mit der Entwicklung eines symmetrischen Neuronalen Netzwerks, mit dessen Hilfe eine Orthonormalisierung von Daten durchgeführt werden kann.

3.1. Modellbeschreibung

Das hier entwickelte Modell eines Neuronalen Netzes beruht auf einem einschichtigen System, wie es in Kapitel 2.3.1. vorgestellt wurde. Demnach besitzt das Modell einen Eingabevektor, einen Ausgabevektor und mehrere Gewichtsvektoren. Die Gewichtsvektoren werden während der Lernphase des Netzes trainiert.

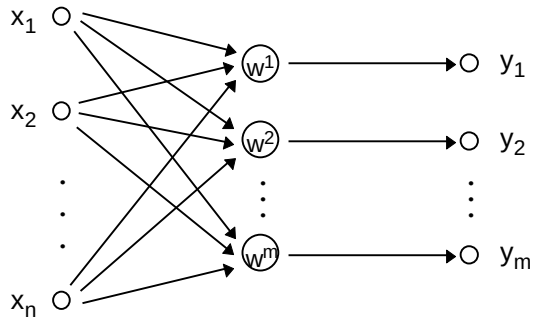


Abbildung 3-2 Modell des Neuronalen Netzes in der Funktionsphase

Jeder Eingabevektor $x = (x_1, x_2, \ldots, x_n)$ wird von jedem Neuron parallel in gleicher Art und Weise verarbeitet, indem ein Ausgabevektor $y = (y_1, y_2, \ldots, y_m)$ mit Hilfe einer linearen Ausgabefunktion berechnet wird. Gemäß Gleichung 2-62 ergibt sich dann für die Ausgabe des i-ten Neurons

$$y_i = w^{iT} x \qquad i = 1, 2, \ldots, m. \tag{3-1}$$

Werden die Gewichtsvektoren w^i der Neuronen zeilenweise in eine Matrix W abgelegt, dann ist der Ausgabevektor y des gesamten Netzes gegeben durch

$$y = W x . \tag{3-2}$$

Aus der Sicht des Transform Coding (siehe Abschnitt Eindimensionale Transformationen) handelt es sich bei Gleichung 3-2 um eine lineare eindimensionale Transformation mit einer Transformationsmatrix W.

Das bisher betrachtete System aus Abbildung 3-3 beschreibt das Modell des Neuronalen Netzes in der Funktionsphase, wo für einen gegebenen Eingabevektor x ein Ausgabevektor y generiert wird.

Die Struktur des Netzes in der Lernphase unterscheidet sich bei dem hier entwickelten Modell von der Struktur des Netzes in der Funktionsphase. Während der Lernphase benötigt das i-te Neuron für den Lernprozeß seine eigene Ausgabe y_i sowie die Ausgaben y_j, $j \neq i$ aller anderen Neuronen. Daher besitzt das Modell in der Lernphase zusätzliche laterale Verbindungen, deren Einfluß im Verlauf des Lernens zu Null reduziert wird. Diese lateralen Verbindungen zwischen den einzelnen Neuronen werden in der folgenden Abbildung gestrichelt dargestellt.

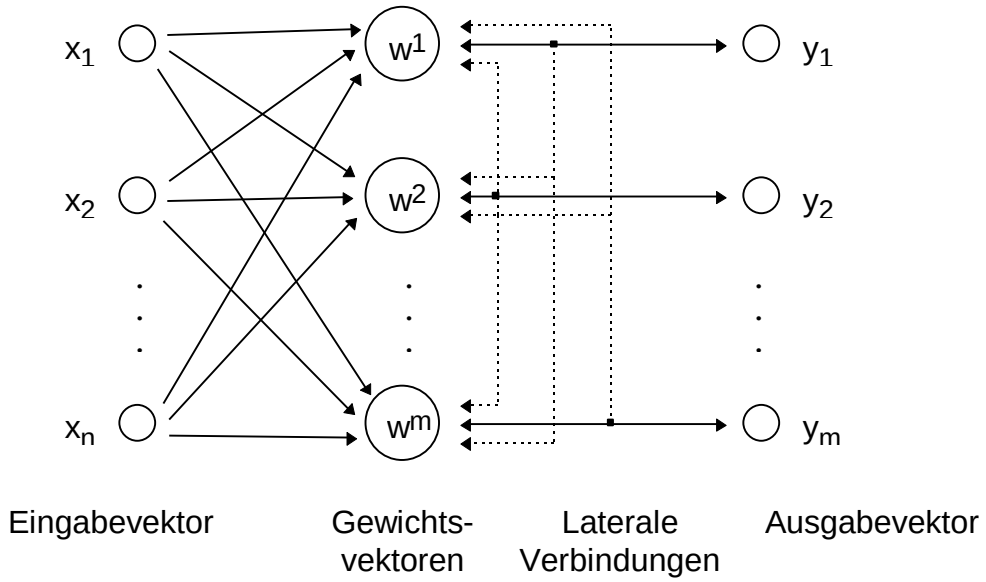


Abbildung 3-3 Modell des Neuronalen Netzes in der Lernphase

Durch die völlig symmetrische Struktur des Modells sowie der Tatsache, daß ein Neuron für seinen Lernprozeß keine internen Daten anderer Neuronen benötigt (beispielsweise deren Gewichte wie im Modell von Silva und Almeida aus Kapitel 2.3.4.), ist das hier entwickelte Neuronale Netz auch für biologische Anwendungen plausibel.

3.2. Herleitung der Lernregel

Dieser Abschnitt beschäftigt sich mit der Herleitung einer Lernregel für das Neuronale Netz aus Abbildung 3-3, so daß das Netz nach einer unüberwachten Trainingsphase in der Lage ist, eine Datenorthonormalisierung durchzuführen. Das iterative Lernverfahren für die Gewichte des Netzes ist daher durch die zwei folgenden Bedingungen bestimmt, die bereits in Kapitel 2.1.3. eingeführt wurden.

$$\left\langle y_i\, y_j \right\rangle = \begin{cases} i \neq j & \text{(Dekorrelation)} \\ i = j & \text{(Normierung)} \end{cases} \qquad i, j = 1, 2, \ldots, m \qquad (3\text{-}3)$$

Die erste Bedingung besagt, daß unterschiedliche Komponenten der Ausgabe des Neuronalen Netzes dekorreliert sein sollen. Nach Kapitel 2.1.2. bewirkt diese Bedingung eine Maximierung des mittleren Informationsgehalts der Ausgaben vom Netz, falls diese normalverteilt sind.

Die zweite Bedingung bewirkt eine Normierung der mittleren Leistung des Ausgabesignals in jeder Komponente. Sind die Eingabedaten zusätzlich mittelwertfrei, dann entspricht die mittlere Leistung der Varianz. In diesem Fall handelt es sich um eine Normierung der Varianz des Ausgabesignals (siehe auch Kapitel 2.1.3.).

Die eigentliche Herleitung der Lernregel erfolgt durch die Modellierung einer passenden Kostenfunktion

$$R(w^1, w^2, \ldots, w^m) = \underbrace{\frac{1}{4}\sum_{i=1}^{m}\sum_{j=1, j\neq i}^{m}\left\langle y_i\, y_j \right\rangle^2}_{R_1} + \underbrace{\frac{1}{4}\sum_{i=1}^{m}\left(P - \left\langle y_i^2 \right\rangle\right)^2}_{R_2}, \qquad (3\text{-}4)$$

die mit $P = (P_Y / m)$ eine Bewertung des Neuronalen Netzes durch den aktuellen Zustand seiner Gewichte ermöglicht. Mit Hilfe des Gradientenverfahrens aus [STU82] wird dann ein Minimum der Kostenfunktion gesucht, indem deren Parameter w^1, w^2, …, w^m entsprechend optimiert werden.

Die Kostenfunktion setzt sich aus zwei Ausdrücken R_1 und R_2 zusammen. Die Quadrate $(\ldots)^2$ in diesen Ausdrücken stellen sicher, daß die einzelnen Summenterme stets positiv sind und sich daher nicht gegenseitig aufheben. Der erste Ausdruck R_1 wird genau dann 0, wenn unterschiedliche Komponenten der Ausgabe des Neuronalen Netzes dekorreliert sind. Der zweite Ausdruck R_2 ergibt 0, falls die mittlere Leistung bzw. die Varianz jeder Ausgabekomponente des Netzes auf den Wert P normiert ist. Da beide Teilausdrücke R_1 und R_2 stets positiv sind und unter bestimmten Bedingungen den Wert 0 annehmen, liegt das globale Minimum der Kostenfunktion bei 0.

Dieses Minimum wird mit Hilfe eines Gradientenabstiegs

$$w^k(t+1) = w^k(t) - \gamma(t)\,\nabla_{w^k} R(w^1, w^2, \ldots, w^m) \qquad k = 1, 2, \ldots, m \tag{3-5}$$

ermittelt, wobei γ eine vom diskreten Iterationsschritt[12] t abhängige Lernrate angibt. Die obige Iterationsgleichung stellt die Lernregel für das Gewicht des k-ten Neurons in Abbildung 3-3 dar. Während einer Trainingsphase wird diese Lernregel auf alle Gewichte w^1, w^2, …, w^m des einschichtigen Neuronalen Netzes angewendet. Der Gradient wird durch den Nabla-Operator

$$\nabla_{w^k} R(w^1, w^2, \ldots, w^m) = \left(\frac{\partial R}{\partial w_1^k}, \frac{\partial R}{\partial w_2^k}, \ldots, \frac{\partial R}{\partial w_n^k}\right) \qquad k = 1, 2, \ldots, m \tag{3-6}$$

beschrieben. Welches Ergebnis erhält man nun für den Gradienten der Kostenfunktion ? Diese Frage beantwortet der folgende Satz.

Satz 3-1: Es sei durch R die Kostenfunktion aus Gleichung 3-4 gegeben. Für den Gradienten von R erhält man dann für k = 1, 2, …, m

$$\nabla_{w^k} R(w^1, w^2, \ldots, w^m) = \left(\sum_{i=1, i\neq k}^{m} \langle y_k\, y_i\rangle \langle x\, y_i\rangle - P\langle x\, y_k\rangle + \langle y_k^2\rangle \langle x\, y_k\rangle\right).$$

Beweis 3-1: (siehe Anhang 3)

Wird der durch Satz 3-1 berechnete Gradient in Gleichung **Error! Bookmark not defined.-Error! Bookmark not defined.** eingesetzt, dann erhält man die deterministische Lernregel

$$w^k(t+1) = w^k(t) + \gamma(t)\left[\underbrace{\sum_{i=1, i\neq k}^{m} \langle y_k\, y_i\rangle \langle x\, y_i\rangle}_{\text{Kreuzkorrelation}} - \underbrace{P\langle x\, y_k\rangle}_{\text{Normierung}} + \underbrace{\langle y_k^2\rangle \langle x\, y_k\rangle}_{\text{Autokorrelation}}\right], \tag{3-7}$$

welche sich aus drei wichtigen Termen zusammensetzt. Während der Kreuzkorrelationsterm für die Dekorrelation der Daten verantwortlich ist, führen die Normierungs- und Autokorrelationsterme eine Normierung der mittleren Leistung bzw. der Varianz jeder Ausgabekomponente durch. Durch eine Umformung von Gleichung 3-7 in

$$w^k(t+1) = w^k(t) + \gamma(t)\left[\underbrace{\sum_{i=1, i\neq k}^{m} \langle y_k\, y_i\rangle \langle x\, y_i\rangle}_{\text{Dekorrelation}} + \underbrace{\langle y_k^2\rangle - P}_{\text{Normierung}} \langle x\, y_k\rangle\right] \tag{3-8}$$

[12] Die Variable *t* wird im folgenden auch als Zeitschritt bezeichnet.

wird auch deutlich, daß für die Normierung ein kleiner positiver oder negativer Anteil des Vektors $\langle x\ y_k \rangle$ verwendet wird, abhängig davon, ob $\langle y_k^2 \rangle$ größer oder kleiner als P ist. Dies ähnelt einer grundlegenden Idee von Silva und Almeida, deren Verfahren zur Datenorthonormalisierung einen kleinen positiven oder negativen Anteil des Vektors w^k benötigt, um die Normierung durchzuführen (siehe Kapitel 2.3.4.).

Mit der Konvergenz des Gradientenverfahrens aus Gleichung **Error! Bookmark not defined.**-**Error! Bookmark not defined.** beschäftigt sich der folgende Satz.

Satz 3-2: Es sei der Gradientenabstieg **Error! Bookmark not defined.**-**Error! Bookmark not defined.** und die Kostenfunktion R aus Gleichung 3-4 gegeben. Wird bei dem Gradientenabstieg eine sequentielle Gewichtskorrektur durchgeführt, so daß in jedem Zeitschritt t ein einzelner Gewichtsvektor w^k für k = 1, 2, …, m adaptiert wird, dann bewirkt die Anwendung der Iterationsgleichung **Error! Bookmark not defined.**-**Error! Bookmark not defined.** eine Konvergenz der Gewichtsvektoren $w^1, w^2, \ldots, w^m$ derart, daß das globale Minimum 0 der Kostenfunktion R gefunden wird.

Beweis 3-2: (siehe Anhang 3)

Die durch Gleichung 3-8 gegebene deterministische Lernregel besitzt einen entscheidenden Nachteil. Sie enthält sehr viele Erwartungswerte und ist daher sehr rechenintensiv. Außerdem läßt sich die Lernregel in dieser Form nur sehr unbefriedigend auf das biologisch plausible Modell in Abbildung 3-3 anwenden.

Bildet man in Gleichung 3-8 keine Erwartungswerte, sondern benutzt direkt die stochastischen Werte, so wird aus der deterministischen Form der Lernregel eine stochastische Approximation, die nach [BRA91] unter bestimmten Voraussetzungen ebenfalls das Minimum der Kostenfunktion R(.) findet.

Mit Hilfe einiger Simulationen konnte ich allerdings feststellen, daß nicht alle Erwartungswerte aus Gleichung 3-8 durch ihre stochastischen Werte ersetzt werden können. Insbesondere die grobe Abschätzung der Terme $\langle y_k\ y_i \rangle$ und $\langle y_k^2 \rangle$ durch $(y_k\ y_i)$ und (y_k^2) führte zu sehr starken Konvergenzproblemen. Dies läßt sich dadurch erklären, daß nach Gleichung 3-3 gerade diese beiden Terme das Konvergenzziel des iterativen Lernverfahrens beschreiben. Eine Randomisierung dieser Terme behindert daher die Konvergenz des Verfahrens in entscheidender Weise.

Unter Berücksichtigung dieser Tatsache ist dann die stochastische Version der Lernregel gegeben durch

$$w^k \leftarrow w^k + \eta \left(\sum_{i \neq k} \langle y_k\ y_i \rangle \, x\, y_i + \left(\langle y_k^2 \rangle - P \right) x\, y_k \right)$$

$$= w^k \left(1 - \eta \left(\sum_{i \neq k} \langle y_k\, y_i \rangle\, y_i \; + \left(\langle y_k^2 \rangle - P \right) y_k \right) \right) \tag{3-9}$$

Bereits in dieser Darstellung der stochastischen Lernregel wird deutlich, daß der Lernprozeß mit Hilfe des Modells aus Abbildung 3-3 beschrieben werden kann. Das k-te Neuron benötigt für seinen Lernprozeß lediglich seine eigene Ausgabe y_k sowie die Ausgaben y_i, $i \neq k$ aller anderen Neuronen. Diese Daten erhält das Neuron über laterale Verbindungen, die in der Lernphase des Modells zusätzlich vorhanden sind.

Aus der stochastischen Lernregel geht auch hervor, daß das k-te Neuron seine eigene Ausgabe sowie die der anderen Neuronen nicht direkt verarbeitet. Es gewichtet die Ausgaben y_i der anderen Neuronen mit dem Term $\langle y_k\, y_i \rangle$ und seine eigene Ausgabe y_k mit dem Term $(\langle y_k^2 \rangle - P)$. Es ist daher sinnvoll für die lateralen Verbindungen auch laterale Gewichte einzuführen, die folgendermaßen definiert werden

$$u_{ij} := \langle y_i\, y_j \rangle \qquad\qquad i, j = 1, 2, \ldots, m.$$

Die stochastische Lernregel aus Gleichung 3-9 läßt sich dann darstellen als

$$w^k \leftarrow w^k \left(1 - \eta \left(\sum_{i \neq k} u_{ki}\, y_i \; + \left(u_{kk} - P \right) y_k \right) \right)$$

$$= w^k \left(1 - \eta \left(\sum_{i \neq k} u_{ki}\, y_i \; + u_{kk}\, y_k - P\, y_k \right) \right)$$

$$= w^k \left(1 - \eta \left(\sum_{i=1} u_{ki}\, y_i \; - P\, y_k \right) \right) \qquad\qquad k = 1, 2, \ldots, m. \tag{3-10}$$

Gemäß ihrer Definition besitzen die lateralen Gewichte nach einer erfolgreichen Konvergenz des Verfahrens die folgenden Werte

$$u_{ij} = \begin{cases} \langle y_i\, y_j \rangle = 0 & i \neq j \\ \langle y_i^2 \rangle = P & i = j \end{cases} \qquad\qquad i, j = 1, 2, \ldots, m,$$

denn in diesem Fall wurde das globale Minimum der Kostenfunktion aus Gleichung 3-4 gefunden. Wie kann nun die stochastische Version der Lernregel in der Praxis eingesetzt werden ? Dazu werden hier die folgenden Berechnungsschritte angegeben.

Für jeden diskreten Zeitschritt t

1) Berechnung des Ausgabevektors y gemäß Gleichung 3-2

$$y \leftarrow W \cdot (1\ x)^T.$$

2) Bestimmung der lateralen Gewichte mit Hilfe eines gleitenden Erwartungswerts

$$u_{ij} = \frac{1}{q} \sum_{k=t-q+1}^{t} y_i \cdot y_j \qquad i, j = 1, 2, \ldots, m.$$

3) Lernen der Gewichte aller Neuronen gemäß der Lernregel

$$w^k = w^k + \eta \cdot (x - \sum_{i=1}^{k-1} u_{ki} \cdot y_i) \cdot y_k \qquad k = 1, 2, \ldots, m.$$

Nachdem nun ein iteratives Lernverfahren für das Modell des Neuronalen Netzes aus Abbildung 3-3 angegeben wurde, stellt sich die Frage, welche Gewichtsvektoren dieses Verfahren als Konvergenzziel besitzt. Dann könnten Aussagen über die Gewichtsmatrix W formuliert werden, die aus der Sicht des Transform Coding ein Transformationskernel darstellt und deren Inverse für eine Rücktransformation und Datenkompression benötigt wird. Mit dieser Thematik beschäftigt sich der folgende Abschnitt.

3.3. Datenorthonormalisierung und Transform Coding

In diesem Abschnitt werden die Bereiche Datenorthonormalisierung und Transform Coding zusammengeführt, um die Frage zu klären, wie die Datenorthonormalisierung im Sinne des Transform Coding zur Datenkompression eingesetzt werden kann.

In Kapitel 2.2.2. wurde gezeigt, daß mit Hilfe einer linearen Transformation

$$y = A\,x \tag{3-11}$$

Daten $x \in X = \Re^n$ auf einen Code $y \in Y = \Re^m$ abgebildet und durch die inverse Transformation

$$x = A^T y = \sum_{i=1}^{n} y_i\, a^{iT} \qquad a^{iT} = A\ i, j\ _{j=1,2,\ldots,n} \tag{3-12}$$

fehlerfrei rekonstruiert werden können. Dabei wurde vorausgesetzt, daß es sich bei A um eine orthogonale Transformationsmatrix handelt, denn in diesem Fall entspricht die Inverse A^{-1} der transponierten Matrix A^T (siehe Satz 2-4).

Es ist nun möglich, eine fehlerbehaftete Datenkompression durchzuführen, indem geeignete Koeffizienten y_i bei der Rekonstruktion von x vernachlässigt werden (siehe Abschnitt Sampling-Fehler und Datenkompression). Diese Form der Kompression bildet die erste Stufe des Transform Coding, wobei die zweite Stufe, welche eine Quantisierung der Koeffizienten durchführt (siehe Kapitel 2.2.3.), in diesem Abschnitt nicht betrachtet wird.

Um diese generellen Überlegungen auch auf die Gewichtsmatrix W des Neuronalen Netzes aus Abbildung 3-2 anwenden zu können, muß geklärt werden, welche Gewichtsvektoren die deterministische Lernregel aus Gleichung 3-7 als Konvergenzziel besitzt.

Dazu wird untersucht, wann die Gradienten der Kostenfunktion R(.) aus Gleichung 3-4 den Wert 0 annehmen. In diesem Fall führt die deterministische Lernregel keine Gewichtskorrekturen mehr durch, denn dann gilt $w^k(t) = w^k(t\text{-}1)$ für k = 1, 2, …, m.

Dies entspricht einer Lösung des Gleichungssystems

$$\begin{aligned} \nabla_{w^1} R(w^1, w^2, \ldots, w^m) &= 0 \\ \nabla_{w^2} R(w^1, w^2, \ldots, w^m) &= 0 \\ &\vdots \\ \nabla_{w^m} R(w^1, w^2, \ldots, w^m) &= 0 \end{aligned} \qquad m \le n \tag{3-13}$$

mit den Gradienten

$$\nabla_{w^k} R\left(w^1, w^2, \ldots, w^m\right) = \sum_{i=1, i\neq k}^{m} \langle y_k\, y_i\rangle \langle x\, y_i\rangle - P\langle x\, y_k\rangle + \langle y_k^2\rangle \langle x\, y_k\rangle$$

$$= \sum_{i=1, i\neq k}^{m} \left\langle w^{kT} x\, x^T w^i \right\rangle \left\langle x\, x^T w^i \right\rangle - P\left\langle x\, x^T w^k \right\rangle + \left\langle w^{kT} x\, x^T w^k \right\rangle \left\langle x\, x^T w^k \right\rangle$$

$$= \sum_{i=1, i\neq k}^{m} w^{kT} \left\langle x\, x^T \right\rangle w^i \left\langle x\, x^T \right\rangle w^i - P\left\langle x\, x^T \right\rangle w^k + w^{kT} \left\langle x\, x^T \right\rangle w^k \left\langle x\, x^T \right\rangle w^k$$

$$= \sum_{i=1, i\neq k}^{m} w^{kT} C_{XX}\, w^i C_{XX}\, w^i - P\, C_{XX}\, w^k + w^{kT} C_{XX}\, w^k C_{XX}\, w^k \qquad (3\text{-}14)$$

Mit Hilfe des folgenden Satzes wird gezeigt, daß das Gleichungssystem 3-13 nicht eine eindeutige Lösung für die Gewichtsvektoren $\{w^1, w^2, \ldots, w^m\}$ besitzt, sondern einen Lösungsraum. Als Basissystem für diesen Lösungsraum wählte ich unter Berücksichtigung von Korollar 2-7 die Menge der orthogonalen Eigenvektoren $\{e^1, e^2, \ldots, e^n\}$ der Korrelationsmatrix C_{XX} zu verschiedenen Eigenwerten $\{\lambda_1, \lambda_2, \ldots, \lambda_n\}$. Dabei soll für die Norm jedes Eigenvektors gelten $|e^r|^2 = 1/\lambda_r$ für r = 1, 2, …, n.

Satz 3-3: Es sei die Kostenfunktion R aus Gleichung 3-4 und das Gleichungssystem 3-13 mit den Gradienten aus 3-14 gegeben. Dann erhält man für die Lösung des Gleichungssystems die Gewichtsvektoren

$$w^i = \sum_{r=1}^{n} a_{ri}\, e^r \qquad i = 1, 2, \ldots, m$$

mit den Bedingungen für deren Koordinaten

$$\sum_{r=1}^{n} a_{ri}^2 = P \qquad i = 1, 2, \ldots, m$$

$$\sum_{r=1}^{n} a_{ri}\, a_{rj} = 0 \qquad i, j = 1, 2, \ldots, m;\ i \neq j$$

bezüglich der orthogonalen Eigenvektorbasis $\{e^1, e^2, \ldots, e^n\}$ zu verschiedenen Eigenwerten $\{\lambda_1, \lambda_2, \ldots, \lambda_n\}$. Bei der Eigenvektorbasis handle es sich um die Eigenvektoren der reellwertigen symmetrischen Korrelationsmatrix C_{XX}, wobei für die Norm jedes Eigenvektors die Bedingung gelte

$$\left|e^r\right|^2 = 1/\lambda_r \qquad r = 1, 2, \ldots, n.$$

Beweis 3-3: (siehe Anhang 3)

Es ist leicht einzusehen, daß nach einer erfolgreichen Konvergenz des iterativen Lernverfahrens die oben beschriebenen Gewichtsvektoren eine Datenorthonormalisierung durchführen, denn mit der linearen Ausgabefunktion aus Gleichung **Error! Bookmark not defined.-Error! Bookmark not defined.** bekommt man eine dekorrelierte Ausgabe

$$\left\langle y_i\, y_j \right\rangle = \left\langle w^{iT} x\, x^T w^j \right\rangle = w^{iT} \left\langle x\, x^T \right\rangle w^j = w^{iT}\, C_{XX}\, w^j$$

$$= \sum_{r=1}^{n} a_{ri}\, e^{rT} \; C_{XX} \; \sum_{s=1}^{n} a_{sj}\, e^{s} \sum_{r=1}^{n} a_{ri}\, e^{rT} \sum_{s=1}^{n} a_{sj}\, C_{XX}\, e^{s}$$

$$= \sum_{r=1}^{n} a_{ri}\, e^{rT} \sum_{s=1}^{n} a_{sj}\, \lambda_s\, e^{s} \sum_{r=1}^{n}\sum_{s=1}^{n} a_{ri}\, a_{sj}\, \lambda_s \underbrace{e^{rT} e^{s}}_{=0\; r\neq s}$$

$$= \sum_{r=1}^{n} a_{ri}\, a_{rj}\, \lambda_r \underbrace{\left| e^{r} \right|^2}_{=1/\lambda_r} = \sum_{r=1}^{n} a_{ri}\, a_{rj} = 0 \qquad\qquad i, j = 1, 2, \ldots, m;\; i \neq j$$

und eine Normierung der mittleren Leistung jeder Ausgabekomponente auf den Wert P

$$\left\langle y_i^2 \right\rangle = \left\langle w^{iT} x\, x^T w^i \right\rangle = w^{iT} \left\langle x\, x^T \right\rangle w^i = w^{iT}\, C_{XX}\, w^i$$

$$= \sum_{r=1}^{n} a_{ri}\, e^{rT} \; C_{XX} \; \sum_{s=1}^{n} a_{si}\, e^{s} \sum_{r=1}^{n} a_{ri}\, e^{rT} \sum_{s=1}^{n} a_{si}\, C_{XX}\, e^{s}$$

$$= \sum_{r=1}^{n} a_{ri}\, e^{rT} \sum_{s=1}^{n} a_{si}\, \lambda_s\, e^{s} \sum_{r=1}^{n}\sum_{s=1}^{n} a_{ri}\, a_{si}\, \lambda_s \underbrace{e^{rT} e^{s}}_{=0\; r\neq s}$$

$$= \sum_{r=1}^{n} a_{ri}^2\, \lambda_r \underbrace{\left| e^{r} \right|^2}_{=1/\lambda_r} = \sum_{r=1}^{n} a_{ri}^2 = P \qquad\qquad i = 1, 2, \ldots, m.$$

Was kann nun über die Gewichtsmatrix W im Sinne des Transform Coding ausgesagt werden, um durch eine inverse Transformation W^{-1} die Eingabedaten $x \in X$ wieder zu rekonstruieren ? Zunächst läßt sich sehr einfach feststellen, daß es sich bei W nicht um eine orthogonale Transformationsmatrix handelt, denn deren Gewichtsvektoren sind im allgemeinen weder orthogonal[13]

$$w^{iT}\, w^j = \sum_{r=1}^{n} a_{ri}\, e^{rT} \sum_{s=1}^{n} a_{sj}\, e^{s} \sum_{r=1}^{n}\sum_{s=1}^{n} a_{ri}\, a_{sj} \underbrace{e^{rT} e^{s}}_{=0\; r\neq s} = \sum_{r=1}^{n} a_{ri}\, a_{rj}\, \frac{1}{\lambda_r} \qquad i \neq j$$

[13] Es ist nicht auszuschließen, daß die Gewichtsvektoren in den Zeilen von W orthogonal zueinander sind. Beispielsweise gehört die Eigenvektorbasis $\{e^1, e^2, \ldots, e^n\}$ aus Satz 3-3 für P = 1 zum Lösungsraum des Gleichungssystems 3-13. Diese Eigenvektoren sind mit Sicherheit orthogonal zueinander.

noch normiert

$$\left|w^i\right|^2 = w^{iT}\, w^i = \sum_{r=1}^{n} a_{ri}\, e^{rT} \sum_{s=1}^{n} a_{si}\, e^s = \sum_{r=1}^{n}\sum_{s=1}^{n} a_{ri}\, a_{si}\, \underbrace{e^{rT} e^s}_{=0\ r\neq s} = \sum_{r=1}^{n} a_{ri}^2 \neq 1$$

und bilden daher keine orthonormierte Basis des Vektorraums $\Re^n$. Nach Satz 2-4 ist eine Matrix aber nur dann orthogonal, wenn sie in ihren Zeilen und Spalten eine orthonormierte Basis des Vektorraums $\Re^n$ enthält. Aus diesem Grund kann die Inverse W^{-1} nicht bestimmt werden, indem die Eigenschaft $A^T = A^{-1}$ einer orthogonalen Matrix ausgenutzt wird.

Dennoch läßt sich die Inverse W^{-1} sehr einfach bestimmen, wenn man berücksichtigt, daß die Transformationsmatrix W eine Datenorthonormalisierung durchführt. In diesem Fall ist die Korrelationsmatrix der Ausgabe C_{YY} gegeben durch

$$C_{YY}\ i,j = \left\langle y_i\, y_j \right\rangle = \begin{cases} 0 & i \neq j \\ 1 & i = j \end{cases} \qquad i, j = 1, 2, \ldots, m \tag{3-15}$$

und läßt sich auch darstellen als

$$C_{YY} = \left\langle y\, y^T \right\rangle = \left\langle Wx\, x^T W^T \right\rangle = W\left\langle x\, x^T \right\rangle W^T = W C_{XX} W^T. \tag{3-16}$$

Unter Berücksichtigung von Gleichung 3-15 läßt sich 3-16 schreiben als

$$W C_{XX} W^T = P\, I$$

und es folgt

$$W^{-1} W C_{XX} W^T = P\, W^{-1} I$$

$$\Leftrightarrow C_{XX} W^T = P\, W^{-1}$$

$$\Leftrightarrow \neq C_{XX} W^T = W^{-1}. \tag{3-17}$$

Für eine inverse Transformation W^{-1} wird also die Korrelationsmatrix C_{XX} der Eingabedaten benötigt.

Unter der Voraussetzung einer erfolgreichen Konvergenz des iterativen Lernverfahrens aus Kapitel 3.2. läßt sich also mit dem Modell des Neuronalen Netzes aus Abbildung 3-2 und der linearen Ausgabefunktion in Gleichung 3-2 eine lineare Transformation der Art

$$y = W x$$

zur Orthonormalisierung der Eingabedaten $x \in X$ einsetzen.

Wird eine Matrix B für die Rekonstruktion eines Eingabevektors x definiert durch

$$x = By \qquad\qquad B := W^{-1},$$

dann läßt sich der Vektor x als folgende Linearkombination darstellen

$$\Leftrightarrow x_k = \sum_{i=1}^{n} b_{ki}\, y_i \qquad\qquad k = 1, 2, \ldots, n$$

$$\Leftrightarrow x_k = \sum_{i=1}^{n} y_i\, b_{ki} = \sum_{i=1}^{n} y_i \left(\frac{1}{P} c^{kT} w^i \right) \qquad\qquad c^{kT} = C_{XX}\ k, j\ _{j=1,2,\ldots,n}$$

$$\Leftrightarrow x = \sum_{i=1}^{n} y_i \left(\frac{1}{P} C_{XX} w^i \right). \qquad (3\text{-}18)$$

Also können durch die inverse Transformation

$$x = \left(\frac{1}{P} C_{XX} W^T \right) y \qquad (3\text{-}19)$$

die Eingabedaten wieder fehlerfrei rekonstruiert werden, wenn es sich bei der Gewichtsmatrix W um eine n × n Matrix handelt, so daß für m = n die Eingabe- und Ausgabevektoren die gleiche Anzahl von Komponenten besitzen.

Mit dieser Darstellung der Eingabedaten x ∈ X wird deutlich, daß die Datenorthonormalisierung zur Datenkompression eingesetzt werden kann, obwohl die zugrundeliegende Transformationsmatrix W im allgemeinen nicht orthogonal ist. Wie beim Transform Coding kann auch hier eine fehlerbehaftete Datenkompression durchgeführt werden, indem geeignete Koeffizienten y_i in Gleichung 3-18 bei der Rekonstruktion der Eingabedaten x ∈ X vernachlässigt werden.

Da es sich bei der hier angestrebten Kompression um eine approximative (und damit fehlerbehaftete) Komprimierung von Daten handelt, spielt die Qualität der dekomprimierten Daten eine wichtige Rolle. Hier stellt der mittlere quadratische Fehler zwischen Ursprungsdaten und rekonstruierten Daten ein sehr brauchbares Gütekriterium dar.

Der durch die unvollständige inverse Transformation verursachte mittlere quadratische Fehler wird hier als Sampling-Fehler bezeichnet und wurde bereits in Kapitel 2.2.4. für das Transform Coding als Gütekriterium verwendet.

Um nun eine Datenkompression durchzuführen, kann bei der Kodierung die Summierung in Gleichung 3-18 nach m ≤ n Schritten abgebrochen werden, so daß die Koeffizienten y_i und Vektoren ((1/P) C_{XX} w^i) bei der Rekonstruktion für i > m nichts mehr zur Summe beitragen.

Wird diese Methode auf Gleichung 3-18 angewendet, dann erhält man statt eines fehlerfreien Eingabevektors x einen approximativen Eingabevektor x'

$$x = \sum_{i=1}^{n} y_i \left(\frac{1}{\sqrt{P}} C_{XX} w^i \right) \approx \left(\sum_{i=1}^{m} y_i \left(\frac{1}{\sqrt{P}} C_{XX} w^i \right) \right) = x' \qquad m \le n. \qquad (3\text{-}20)$$

Wie groß ist der durch diese Approximation verursachte mittlere quadratische Sampling-Fehler ? Auf diese Frage gibt der folgende Satz eine Antwort.

Satz 3-4: Es sei y = W x eine lineare Transformation mit einer Matrix W derart, daß die Transformation eine Datenorthonormalisierung durchführt. Jeder Eingabevektor $x \in X$ sei durch die inverse Transformation

$$x = \sum_{i=1}^{n} y_i \left(\frac{1}{\sqrt{P}} C_{XX} w^i \right)$$

gegeben. Zu jedem Eingabevektor x existiere ein approximativer Vektor x', der durch eine approximative inverse Transformation

$$x' = \sum_{i=1}^{m} y_i \left(\frac{1}{\sqrt{P}} C_{XX} w^i \right) \qquad m \le n$$

bestimmt wird, indem alle Summenterme für $m+1 \le i \le n$ vernachlässigt werden. Der hier definierte mittlere quadratische Sampling-Fehler

$$\varepsilon_s^2 = \left\langle \left| x - x' \right|^2 \right\rangle$$

ist dann gegeben durch

$$\varepsilon_s^2 = \frac{1}{P} \sum_{r=1} \lambda_r \sum_{i=m+1}^{n} a_{ri}^2 .$$

Beweis 3-4: (siehe Anhang 3)

Mit Hilfe des obigen Satzes wird ersichtlich, daß der mittlere quadratische Sampling-Fehler von den Eigenwerten aller Eigenvektoren $\{e^1, e^2, \ldots, e^n\}$ der Korrelationsmatrix C_{XX} und von den Koordinaten der vernachlässigten Gewichte $\{w^{m+1}, w^{m+2}, \ldots, w^n\}$ abhängig ist. Der folgende Satz zeigt nun, wie die Zeilenvektoren der Transformationsmatrix W gewählt werden müssen, damit dieser Fehler minimal wird.

Satz 3-5: Es sei y = W x eine lineare Transformation mit einer Matrix W derart, daß die Transformation eine Datenorthonormalisierung durchführt. Weiterhin seien die Zeilen von W durch die Vektoren

$$w^{iT} = \sqrt{P}\, e^{iT} \qquad i = 1, 2, \ldots, n;\ P > 0$$

gegeben, wobei es sich bei $\{e^1, e^2, \ldots, e^n\}$ um die Eigenvektoren der Korrelationsmatrix C_{XX} zu verschiedenen Eigenwerten $\{\lambda_1, \lambda_2, \ldots, \lambda_n\}$ handelt. Die Eigenvektoren seien gemäß der Größe ihrer zugehörigen Eigenwerte absteigend geordnet und für deren Norm gelte die Bedingung

$$\left|e^i\right|^2 = \frac{1}{\lambda_i} \qquad i = 1, 2, \ldots, n.$$

Besitzt die Transformationsmatrix W die oben beschriebene Gestalt, dann wird der durch

$$\varepsilon_s^2 = \left\langle \left|x - x'\right|^2 \right\rangle$$

definierte mittlere quadratische Sampling-Fehler minimal und vereinfacht sich zu

$$\varepsilon_s^2 = \sum_{r=m+1}^{n} \lambda_r .$$

Beweis 3-5: (siehe Anhang 3)

Die letzten beiden Sätze haben gezeigt, daß die Datenorthonormalisierung nicht nur ein datenverarbeitendes Pre-Processing für andere Kodierungsverfahren darstellt, sondern das sie auch als ein Verfahren zur approximativen Datenkompression im Sinne des Transform Coding eingesetzt werden kann.

Ähnlich wie beim Transform Coding wird der mittlere quadratische Sampling-Fehler zwischen den Ursprungsdaten und den rekonstruierten Daten minimal, wenn die Zeilenvektoren der Transformationsmatrix durch die Eigenvektoren der Korrelationsmatrix der Eingabedaten gegeben und gemäß der Größe ihrer zugehörigen Eigenwerte absteigend sortiert sind.

Der Unterschied zu einer Hotelling Transformation beruht lediglich auf der Norm dieser Eigenvektoren. Die Hotelling Transformation besitzt eine orthogonale Transformationsmatrix mit orthonormierten Eigenvektoren in den Zeilen, wohingegen die Norm der Eigenvektoren bei der Datenorthonormalisierung von der vorgegebenen mittleren Leistung des Ausgabesignals abhängig ist. Diese Norm ist nach Satz 3-5 gegeben durch

$$\left|w^i\right|^2 = \left|\sqrt{P}\, e^i\right|^2 = \left\langle \sqrt{P}\, e^i, \sqrt{P}\, e^i \right\rangle = P \left\langle e^i, e^i \right\rangle = P \left|e^i\right|^2 = P/\lambda_i \qquad i = 1, 2, \ldots, n.$$

Durch die Sortierung der Eigenvektoren wird erreicht, daß bei einer fehlerbehafteten Rekonstruktion der Daten genau jene Koeffizienten vernachlässigt werden, die mit Hilfe der Eigenvektoren mit den kleinsten Eigenwerten berechnet wurden.

Daraus ergibt sich nach Satz 3-5, daß der mittlere quadratische Sampling-Fehler nur noch von den kleinsten Eigenwerten $\lambda_{m+1}, \ldots, \lambda_n$ abhängig ist und damit minimal wird.

3.4. Simulationen

Nachdem in den beiden Abschnitten 3.1. und 3.2. ein von mir entwickeltes Modell für ein Neuronales Netz und eine zugehörige Lernregel vorgestellt wurden, beschäftigt sich dieser Abschnitt mit einigen Simulationen dieses Neuronalen Netzes, um dessen Korrektheit und Konvergenzverhalten in der Praxis zu überprüfen. Dabei wurden die Simulationen unter zwei verschiedenen Kriterien durchgeführt.

Zum ersten sollen Konvergenzeigenschaften betrachtet werden, um einen Eindruck davon zu bekommen, wie schnell das iterative Lernverfahren aus Abschnitt 3.2. sein Konvergenzziel findet und ob es einem Vergleich mit dem Verfahren von Silva und Almeida aus Kapitel 2.3.4. standhält. Zum zweiten soll eine Rücktransformation gestörter Daten durchgeführt werden, um die inverse Transformation aus Abschnitt

3.3. zu überprüfen und um die informationstheoretischen Untersuchungen aus Kapitel 2.1.3. zu bestätigen.

3.4.1. Konvergenzbeispiele

In diesem Abschnitt werden zwei Simulationen vorgestellt, die jeweils für drei unterschiedliche Modelle Neuronaler Netze durchgeführt wurden. Zwei dieser Modelle wurden bereits in der Diplomarbeit vorgestellt. Bei ihnen handelt es sich um das selbstentwickelte Neuronale Netz aus Abschnitt 3.1., das im folgenden als *RipNet-Modell*[14] bezeichnet wird, und um das Modell von Silva und Almeida aus Kapitel 2.3.4.

Bei dem dritten Modell handelt es sich um eine Variante des RipNet-Modells, die im folgenden als *RipNet2-Modell* bezeichnet und in der Diplomarbeit mathematisch nicht weiter untersucht wird. Diese Variante entstand aus der Beobachtung, daß das Verfahren von Silva und Almeida (Gleichung 2-81) und das RipNet-Modell (Gleichung 3-8) einen gemeinsamen Term

$$\left(1 - \langle y_k^2 \rangle\right) \qquad k = 1, 2, \ldots, n$$

zur Normierung der Ausgabe besitzen. Während bei Silva und Almeida dieser Term mit dem adaptierten Gewichtsvektor multipliziert wird

$$\left(1 - \langle y_k^2 \rangle\right) w^k \qquad k = 1, 2, \ldots, n,$$

enthält das RipNet-Modell den Produktterm

$$\left(1 - \langle y_k^2 \rangle\right) \langle x\, y_k \rangle \qquad k = 1, 2, \ldots, n.$$

Das RipNet2-Modell erhält man nun, indem beim RipNet-Modell der Term $\langle x\, y_k \rangle$ durch den Gewichtsvektor w^k ersetzt wird. Bei einem Vergleich des Verfahrens von Silva und Almeida mit dem RipNet-Modell stellte ich fest, daß diese trotz der sehr unterschiedlichen mathematischen Herleitungen gewisse Ähnlichkeiten aufweisen. Daher ersetzte ich versuchsweise einige Terme im RipNet-Modell und fand heraus, daß die oben beschriebene Modifikation eine beachtliche Beschleunigung der Konvergenz des RipNet-Verfahrens zur Folge hat.

Trotz dieser Modifikation bleibt das RipNet2-Modell für biologische Anwendungen plausibel, da es für seinen Lernprozeß keine internen Daten anderer Neuronen benötigt (siehe auch Abbildung 3-3). Die Modifikation der ursprünglichen Lernregel konnte ich auf Grund der zeitlichen Rahmenbedingungen nicht näher untersuchen.

---

[14] Die ersten drei Buchstaben der Bezeichnung *RipNet* stammen aus meinem Nachnamen *Rippl*.

In der ersten Simulation wurden die determinstischen Versionen der Lernregeln von Silva und Almeida

$$w^k(t) = w^k(t-1) + \gamma\, w^k(t-1)\left(1 - w^k(t-1)^T \sum_{i=1}^{n} w^i(t-1)\, C_{XX}\, w^i(t-1)\right)$$

$$= w^k(t-1) + \gamma\, w^k(t-1)\left(1 - w^k(t-1)^T \sum_{i=1}^{n} w^i(t-1)\, C_{XX}\, w^i(t-1)\right)$$

$$= w^k(t-1) + \gamma\, w^k(t-1)\left(1 - w^k(t-1)^T \sum_{i=1,i\neq k}^{n} w^i(t-1)\, C_{XX}\, w^i(t-1) + w^k(t-1)^T C_{XX}\, w^k(t-1)\right)$$

$$= w^k(t-1) + \gamma\, w^k(t-1)\left(1 - w^k(t-1)^T \sum_{i=1,i\neq k}^{n} w^i(t-1)\, C_{XX}\, w^i(t-1) + w^k(t-1)^T C_{XX}\, w^k(t-1) - 1\right)$$

und von den Modellen RipNet

$$w^k(t) = w^k(t-1) + \gamma\, w^k(t-1)\left(1 - \sum_{i=1,i\neq k}^{n} w^i(t-1)\, C_{XX}\, w^i(t-1) + w^k(t-1)^T C_{XX}\, w^k(t-1)\right)$$

und RipNet2

$$w^k(t) = w^k(t-1) + \gamma\, w^k(t-1)\left(1 - \sum_{i=1,i\neq k}^{n} w^i(t-1)\, C_{XX}\, w^i(t-1) + w^k(t-1)^T C_{XX}\, w^k(t-1) - 1\right)$$

für P = 1 und k = 1, 2, …, n verwendet, um die Konvergenz der drei Verfahren zu vergleichen. Der Übersichtlichkeit halber wurden in den Korrekturtermen der Lernregeln die Zeitschritte (t-1) für die Gewichtsvektoren nicht angegeben.

Für alle drei Modelle wurden identische Rahmenbedingungen vorgegeben, um die Simulationsergebnisse miteinander vergleichen zu können. Bei jedem Verfahren wurde die Gewichtsmatrix als Einheitsmatrix W = I initialisiert und eine konstante Lernrate von $\gamma = 0.05$ verwendet. Um einen Gradientenabstieg der Verfahren zu garantieren, wurden sequentielle Gewichtskorrekturen durchgeführt, so daß in jedem Zeitschritt t ein einzelner Gewichtsvektor adaptiert wird (siehe auch Satz 3-2).

Auf eine Menge von Eingabedaten konnte in dieser Simulation ganz verzichtet werden, da die oben angegebenen determinstischen Lernregeln nur von den Erwartungswerten, das heißt von den Korrelationen der Eingabe abhängig sind.

Die Korrelationsmatrix C_{XX} der Eingabedaten wurde synthetisch mit Hilfe der Korrelationsfunktion von Wintz und Habibi berechnet (siehe Abschnitt Karhunen-Loève Transformation). Für die Parameter der Korrelationsfunktion wurden die Werte $\alpha = 0.125$ und $\beta = 0.249$ verwendet, die eine gute Approximation für die Korrelationen des Bildes eines Kameramanns aus [HAB71] liefern.

Um beispielhaft einen typischen Simulationsverlauf darzustellen, wurde jede der drei oben angegebenen Lernregeln eine bestimmte Anzahl von Schritten simuliert und während der Simulation durch die Kostenfunktion aus Gleichung 3-4 bewertet. Diese Daten wurden mit Hilfe eines Plot-Programms ausgewertet und werden im folgenden grafisch dargestellt.

Das Verfahren von Silva und Almeida konvergierte sehr schnell,

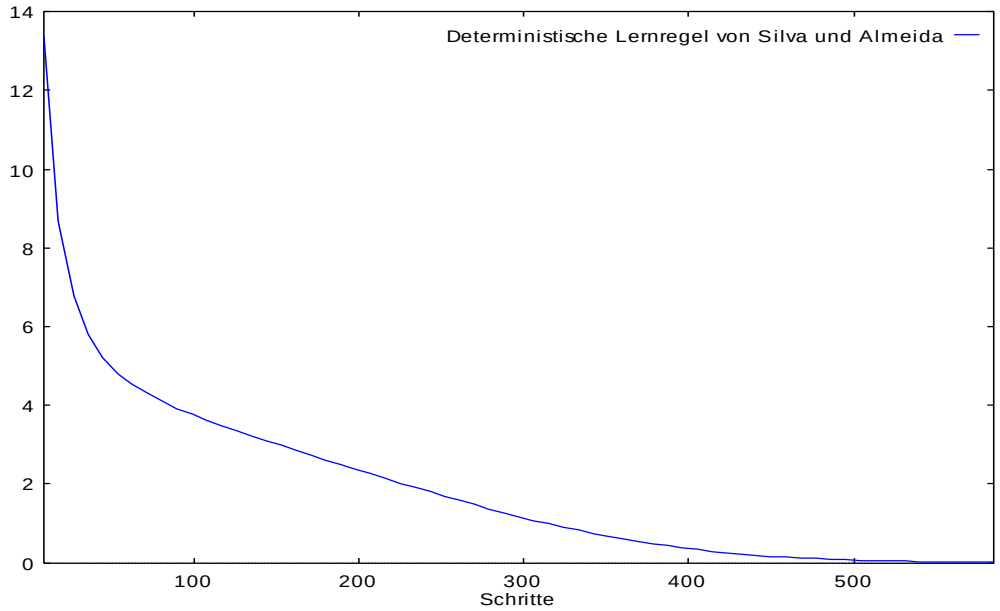


Abbildung 3-4 Konvergenz der deterministischen Lernregel von Silva und Almeida

während die deterministische Lernregel für das RipNet-Modell sich nur sehr langsam dem Konvergenzziel näherte.

Dies wird in der folgenden Abbildung sehr deutlich dargestellt.

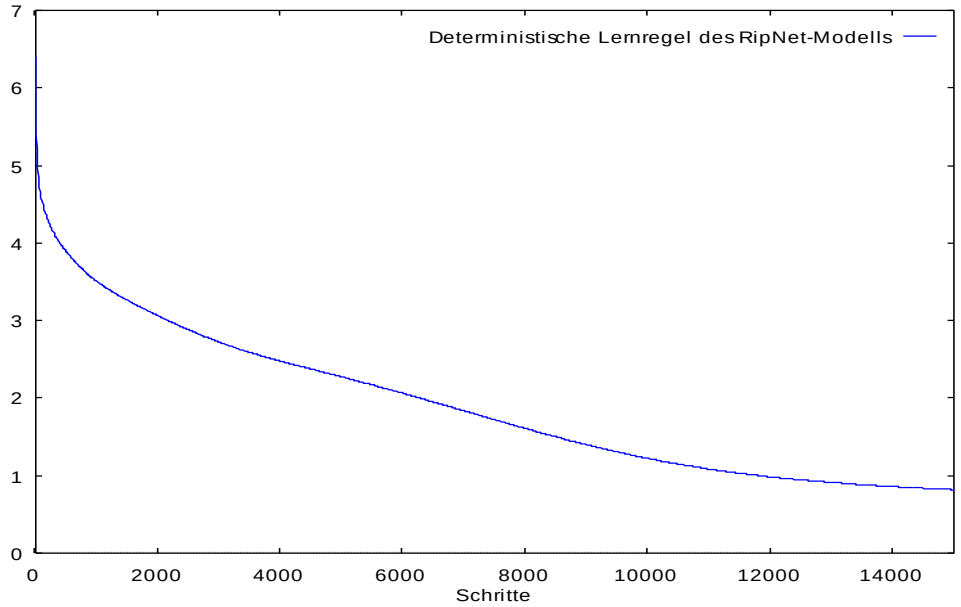


Abbildung 3-5 Konvergenz der deterministischen Lernregel des RipNet-Modells

Es stellt sich natürlich sofort die Frage, warum die deterministische Version des RipNet-Verfahrens derart langsam konvergiert. Mit Hilfe dieser und der folgenden Simulation konnte ich feststellen, daß die Erwartungswerte $\langle x\ y_i \rangle = (C_{XX}\ w^i)$ für i = 1, 2, …, n der deterministischen Lernregel die Konvergenz stark verlangsamen, denn die stochastische Version des RipNet-Modells zeigt ein sehr gutes Konvergenzverhalten.

Eine Bestätigung dieser Annahme liefert die Variante RipNet2,

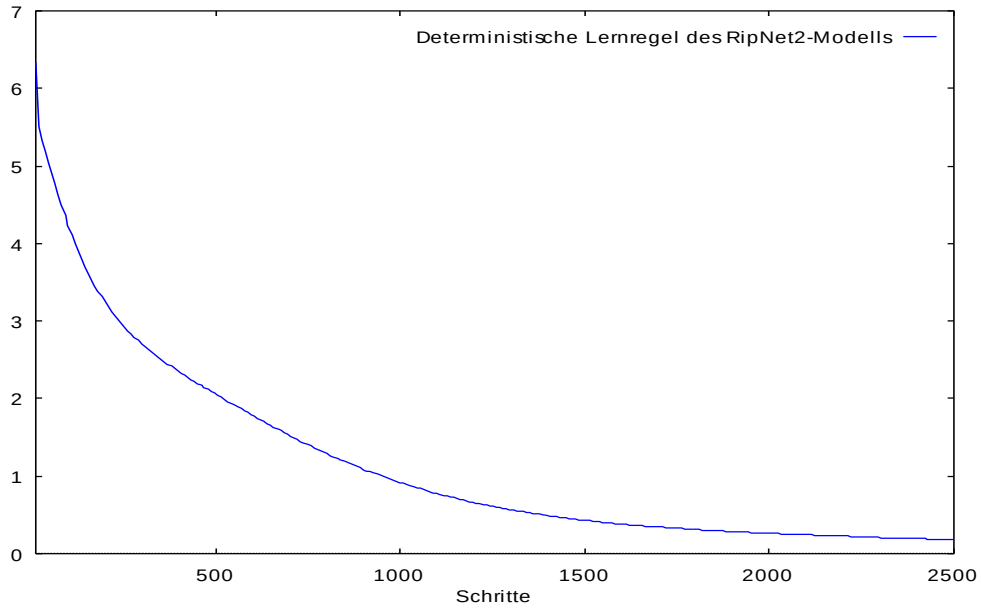


Abbildung 3-6 Konvergenz der deterministischen Lernregel des RipNet2-Modells

bei der ein Erwartungswert $\langle x\ y_k \rangle$ durch den Gewichtsvektor w^k ersetzt wurde.

Die zweite Simulation in diesem Abschnitt vergleicht die Konvergenz der stochastischen Lernregeln von Silva und Almeida

$$w^k(t+1) = w^k(t) + \gamma(t)\,y_k(t)\left[x - (1-\gamma(t))\sum_{i=1}^{k} y_i(t)\,w^i(t)\right]$$

$$= w^k(t) + \gamma(t)\,y_k(t)\left[x - (1-\gamma(t))\sum_{i=1,\,i\neq k}^{k-1} y_i(t)\,w^i(t) - \left(y_k^2(t) - (1-\gamma(t))\right)w^k(t)\right],$$

vom RipNet-Modell

$$w^k(t+1) = w^k(t) + \gamma(t)\,y_k(t)\left[x - (1-\gamma(t))\sum_{i=1,\,i\neq k}^{k-1} y_i(t)\,x - \left(y_k^2(t) - (1-\gamma(t))\right)x\right]$$

und vom RipNet2-Modell

$$w^k(t+1) = w^k(t) + \gamma(t)\,y_k(t)\left[x - (1-\gamma(t))\sum_{i=1,\,i\neq k}^{k-1} y_i(t)\,x - \left(y_k^2(t) - (1-\gamma(t))\right)w^k(t)\right]$$

miteinander für P = 1 und k = 1, 2, …, n. Dabei ist die stochastische und deterministische Lernregel bei Silva und Almeida identisch, da der Erwartungwert $\langle y_k\, y_i\rangle$ nicht durch den Term $(y_k\, y_i)$ ersetzt wurde. Dies ermöglicht einen fairen Vergleich mit den RipNet-Verfahren, da dort der Term $\langle y_k\, y_i\rangle$ in den stochastischen Versionen ebenfalls erhalten bleibt und durch einen gleitenden Erwartungswert modelliert wird (siehe Abschnitt 3.2.).

Wie in den Berechnungsschritten aus Abschnitt 3.2. bereits angegeben, wird für die folgende Simulation eine parallele Gewichtskorrektur durchgeführt, so daß in jedem Zeitschritt t nach der Präsentation einer Eingabe und der Berechnung einer Ausgabe alle Gewichtsvektoren adaptiert werden.

Es wurden auch hier identische Rahmenbedingungen für alle drei Modelle vorgegeben, um die Simulationsergebnisse miteinander vergleichen zu können. Die Gewichtsmatrix wurde als Einheitsmatrix und die Lernrate mit dem Wert $\gamma = 0.08$ initialisiert. Um die Konvergenz der stochastischen Verfahren zu gewährleisten, wurde die Lernrate alle 20 Simulationsschritte um den Wert 0.004 vermindert. Die für die stochastischen Lernregeln notwendigen Eingabedaten wurden in Form von 20 Vektoren mit je 5 gaußverteilten mittelwertfreien Komponenten zur Verfügung gestellt. Dabei besitzen die einzelnen Komponenten unterschiedliche Varianzen.

Um beispielhaft einen typischen Simulationsverlauf darzustellen, wurden auch bei der zweiten Simulation in diesem Abschnitt die oben angegebenen stochastischen Lernregeln eine bestimmte Anzahl von Schritten simuliert und während der Simulation durch die Kostenfunktion aus Gleichung 3-4 bewertet.

Wiederrum konvergierte das Verfahren von Silva und Almeida sehr schnell, wie die folgende Abbildung zeigt.

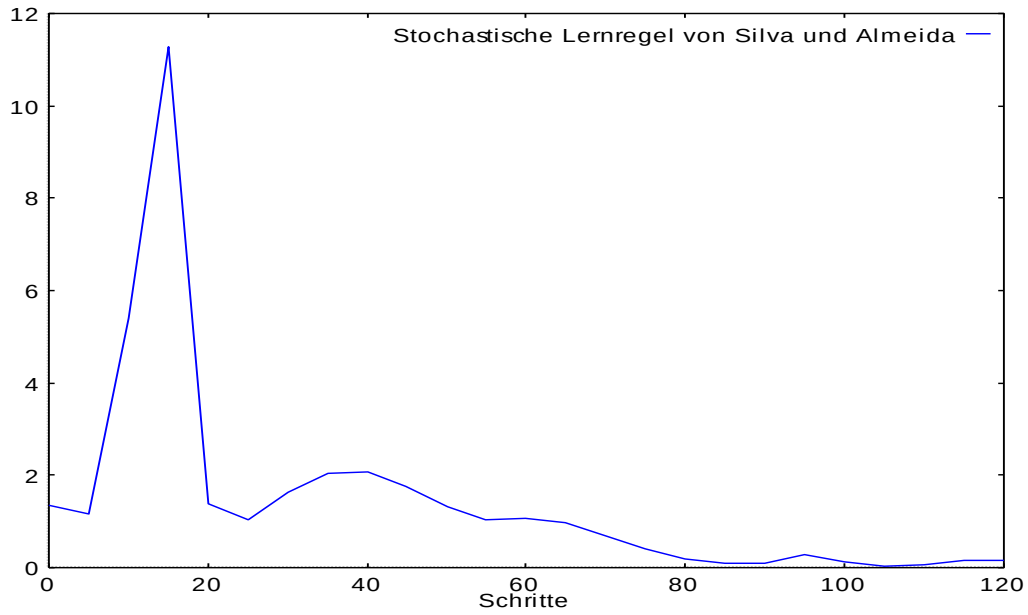
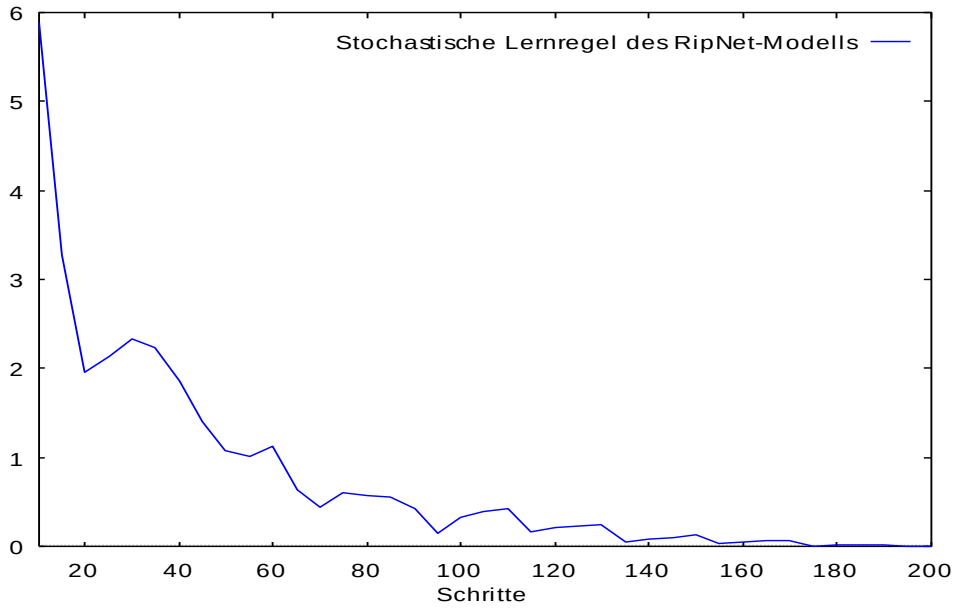


Abbildung 3-7 Konvergenz der stochastischen Lernregel von Silva und Almeida

Der hohe Wert der Kostenfunktion zu Beginn der Simulation resultiert aus der Wahl des Startwertes der Lernrate. Dieser wurde möglichst groß gewählt, um die Konvergenz aller drei Verfahren zu beschleunigen. Wird der Startwert für ein Verfahren zu groß gewählt, dann findet keine Konvergenz mehr statt.

Auch das RipNet-Modell zeigt diesmal ein sehr gutes Konvergenzverhalten



Abbilding 3-8 Konvergenz der stochastischen Lernregel des RipNet-Modells

und das RipNet2-Modell konvergierte in dieser Simulation sogar am schnellsten.

Dies zeigt die folgende Abbildung.

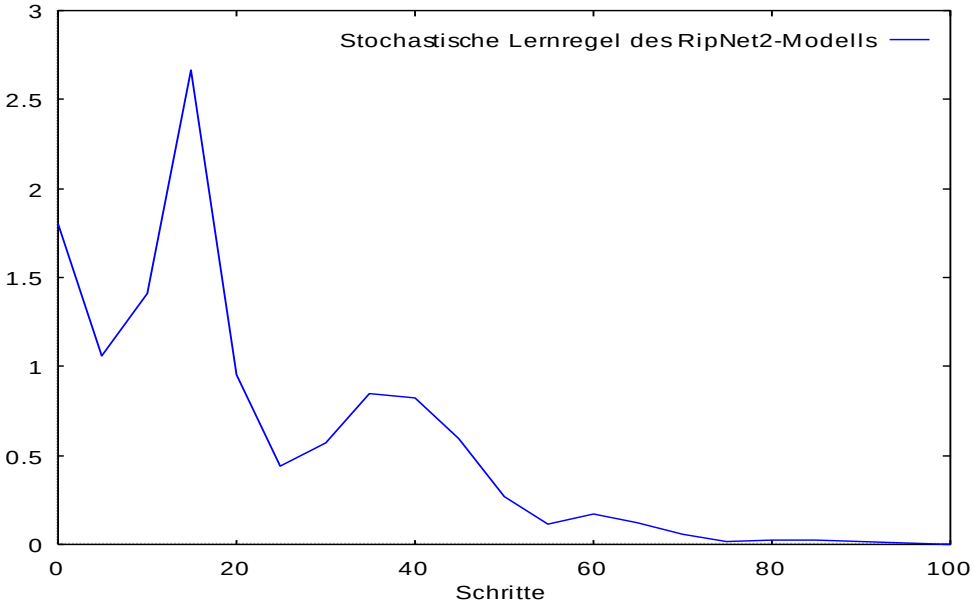


Abbildung 3-9 Konvergenz der stochastischen Lernregel des RipNet2-Modells

Als Ergebnis der oben durchgeführten Simulationen kann man festhalten, daß die deterministische Version des RipNet-Verfahrens keine große Bedeutung in der Praxis zu besitzen scheint. Jedoch kann die stochastische Version durch ihr gutes Konvergenzverhalten durchaus in der Praxis eingesetzt werden. Zudem ist sie durch das zugrundeliegende Modell eines symmetrischen Neuronalen Netzes auch für biologische Anwendungen plausibel (siehe Abbildung 3-3 und Abschnitt 3.2.).

3.4.2. Rücktransformation gestörter Daten

Zu Beginn des 3. Kapitels wurde in Abbildung 3-1 das Modell einer Signalkodierung mit anschließender Übertragung der Signale über einen gestörten Kanal angegeben. Es wurde festgestellt, daß Signale oftmals *vor* ihrer Übertragung über ein Medium passend kodiert werden müssen, wobei die Kodierung mit Hilfe einer linearen Transformation erfolgen sollte.

In Kapitel 2.1.3. wurde aufgrund der Vermutung, daß kodierte Datenworte mit einer einheitlichen Varianz in jeder Komponente durch additive Störungen bei einer Übertragung weniger beeinflußt werden, als Datenworte mit unterschiedlichen Varianzen, eine informationstheoretische Untersuchung durchgeführt, die zeigte, daß in den Datenworten ein Maximum an Information über einen gestörten Kanal übertragen werden kann, wenn vor der Übertragung eine Orthonormalisierung dieser Daten durchgeführt wird.

Das Schema der Signalkodierung aus Abbildung 3-1 und die informationstheoretischen Untersuchungen in Kapitel 2.1.3. bilden die Grundlage für die in diesem Abschnitt durchgeführte Simulation.

Diese vergleicht die Sequenz der Operationen Kodierung, Übertragung über einen gestörten Kanal und Dekodierung der klassischen Hotelling Transformation mit der Datenorthonormalisierung des RipNet-Modells aus Abschnitt 3.4.1. Mit den informationstheoretischen Grundlagen aus Kapitel 2.1. kann man erwarten, daß die klassische Hotelling Transformation einen größeren Fehler zur Folge hat als die Datenorthonormalisierung des RipNet-Modells.

Um den Einfluß der Störungen eindeutig vom Kompressionsfehler der Datenkompression unterscheiden zu können, wurde auf eine Komprimierung der Daten verzichtet. Daher besitzen die Eingabe- und Ausgabevektoren der linearen Transformation

$$y = W x \tag{3-21}$$

die gleiche Anzahl von Komponenten (m = n). Die Transformationsmatrix W wurde im Falle der Datenorthonormalisierung durch eine Anwendung der Berechnungsschritte aus Abschnitt 3.2. und der dort angegebenen stochastischen Lernregel des RipNet-Modells bestimmt. Die für die Hotelling Transformation notwendigen Eigenvektoren der Korrelationsmatrix C_{XX} wurden synthetisch mit Hilfe des klassischen Jacobi-Verfahrens aus [STU82] berechnet.

Bei der Durchführung der Datenorthonormalisierung muß in dieser Simulation allerdings beachtet werden, daß die vorgegebene mittlere Leistung P eine wichtige Rolle spielt. Bisher wurde, wie in der Datenorthonormalisierung üblich, die Varianz bzw. die mittlere Leistung jeder Ausgabekomponente auf den Wert 1 normiert.

Es ist hier nicht mehr sinnvoll, für P den Wert 1 anzunehmen, wenn die zu übertragenden Signale der Hotelling Transformation[15] eine größere mittlere Leistung besitzen. Dies würde zu einer Benachteiligung der Datenorthonormalisierung führen, da sich additive Störungen bei Signalen mit einer geringen mittleren Leistung stärker auswirken.

Um einen fairen Vergleich beider Verfahren zu gewährleisten, wurde für die Simulation angenommen, daß der gestörte Übertragungskanal Datenworte übertragen kann, deren maximale Leistung in jeder Komponente durch den größten aller Eigenwerte der Hotelling Transformation begrenzt ist. Für die Datenorthonormalisierung wurde eine Leistung P vorgegeben, die dem größten Eigenwert der Hotelling Transformation entspricht.

Die Eingabedaten der Simulation wurden (wie in Abschnitt 3.4.1.) in Form von 25 Vektoren mit je 5 gauß-verteilten mittelwertfreien Komponenten zur Verfügung gestellt.

Um die Übertragung über einen gestörten Kanal zu simulieren, wurden die Ausgabedaten beider Verfahren durch eine Addition von Vektoren ϕ mit sehr kleinen ebenfalls gauß-verteilten mittelwertfreien Komponenten verfälscht

$$y' = y + \phi. \tag{3-22}$$

Anschließend wurden diese verfälschten Ausgabedaten unter Anwendung der inversen Transformation des RipNet-Modells

$$x' = C_{XX} W_{rip}^{T} y' \tag{3-23}$$

bzw. der inversen Transformation der Hotelling Transformation

[15] Die mittlere Leistung in den einzelnen Komponenten der Ausgabevektoren ist bei der Hotelling Transformation gerade durch die Eigenwerte der Korrelationsmatrix C_{XX} gegeben, falls die Eingabedaten mittelwertfrei sind (siehe Satz 2-5).

$$x' = W_{pca}^{T}\, y' \tag{3-24}$$

dekodiert[16] und der durch die Verfälschungen resultierende Gesamtfehler

$$\varepsilon = \sum_{x} \left| x - x' \right|^2 \tag{3-25}$$

für beide Transformationen berechnet. Dieser Gesamtfehler wurde mit Hilfe eines Programms zur Darstellung von Balkendiagrammen für drei verschiedene Durchläufe der Simulation grafisch dargestellt.

Für sehr kleine Störungen der Ausgabesignale im Bereich $-0.001 \le \phi \le +0.001$ liefert die Datenorthonormalisierung einen deutlich kleineren Fehler

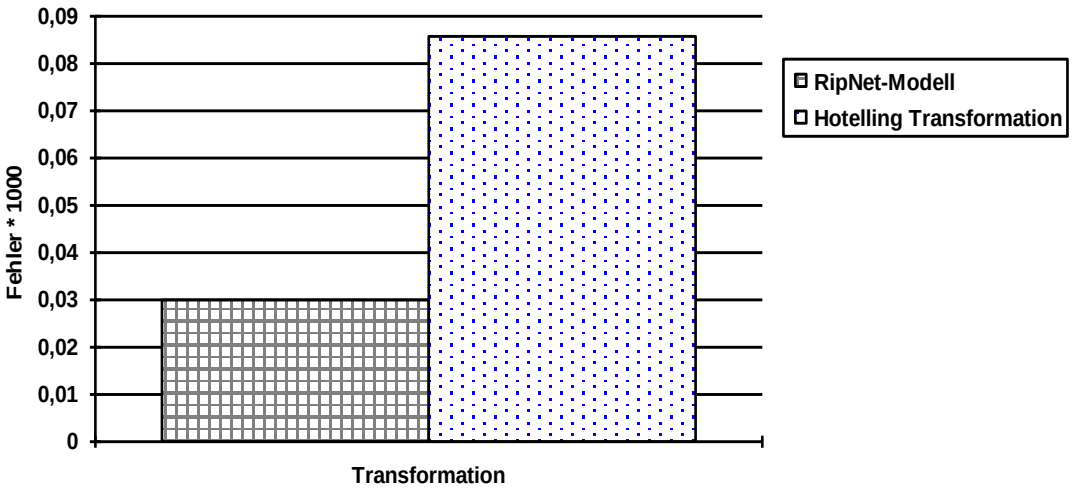


Abbildung 3-10 Rekonstruktionsfehler der beiden Verfahren bei sehr geringen Störungen

16 Es wurde darauf geachtet, daß der durch numerische Ungenauigkeiten in den Transformationsmatrizen W_{rip} und W_{pca} verursachte Fehler bei beiden Verfahren identisch ist.

und auch bei Störungen im Bereich $-0.01 \leq \phi \leq +0.01$

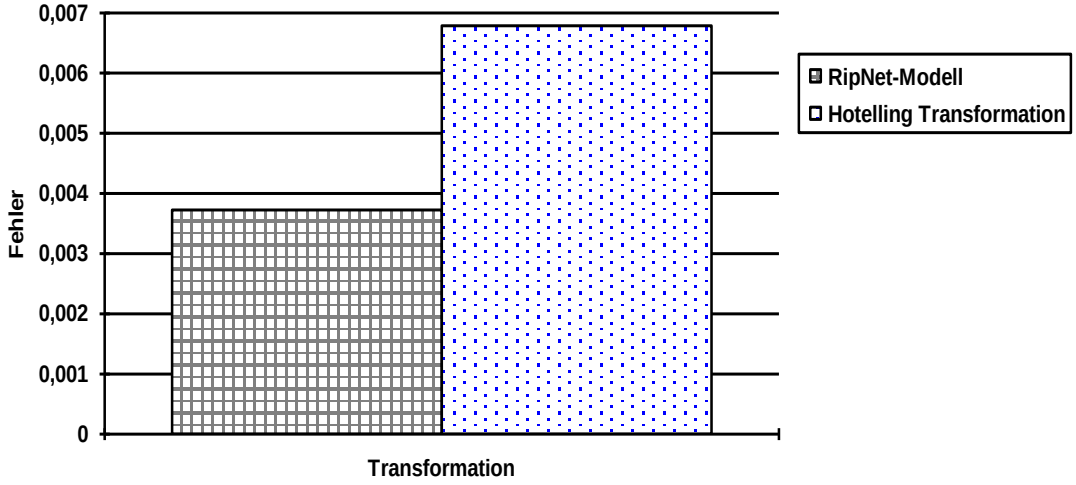


Abbildung 3-11 Rekonstruktionsfehler der beiden Verfahren bei Störungen mittlerer Intensität

sowie sehr starken Störungen im Bereich $-0.1 \leq \phi \leq +0.1$

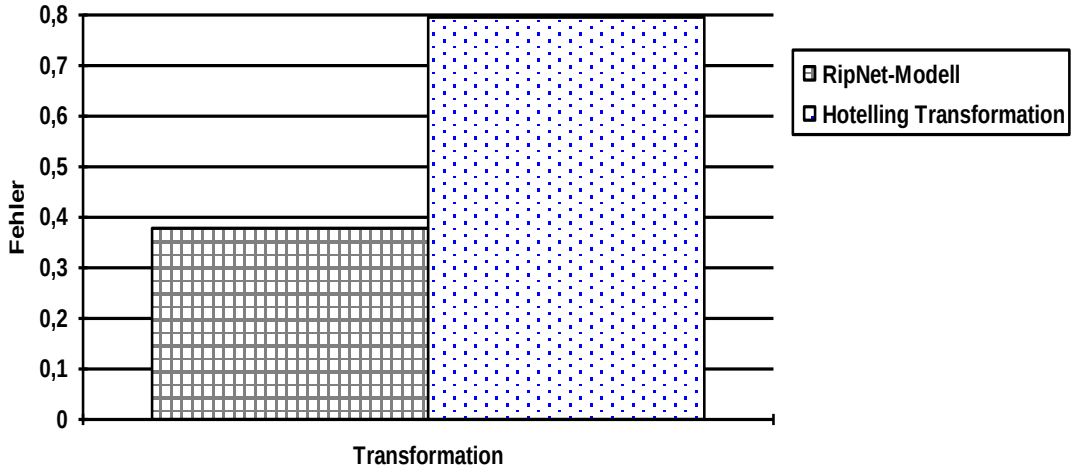


Abbildung 3-12 Rekonstruktionsfehler der beiden Verfahren bei sehr starken Störungen

bleibt der Vorsprung der Datenorthonormalisierung erhalten.

Wie in Kapitel 2.1. bereits informationstheoretisch hergeleitet, bietet die Datenorthonormalisierung wesentliche Vorteile, wenn normalverteilte Daten vor ihrer Übertragung über einen gestörten Übertragungskanal kodiert und komprimiert werden. Die zu übertragenden Datenworte besitzen im Idealfall nicht nur einen maximalen mittleren Informationsgehalt, sondern werden auch bei einer Übertragung durch additive Störungen nicht so stark verfälscht, wie die Datenworte von Verfahren, welche an eine Hotelling Transformation angelehnt sind.

4. Software-Architektur der Simulationen

Zur Durchführung der Simulationen meiner Diplomarbeit wurden in eigenständiger Arbeit zwei Bibliotheken entwickelt, welche die Simulationen stark vereinfachen und auf ein hohes Abstraktionniveau heben. Die Entwicklung der Bibliotheken fand parallel zur Ausarbeitung der Diplomarbeit statt. Daher wurden die Bibliotheken mehrfach überarbeitet und gründlich ausgetestet.

Zu Beginn der Diplomarbeit zeigte sich sehr schnell, daß sich bestimmte Abläufe bei Simulationen wiederholten und die Simulationsprogramme sehr unübersichtlich wurden. Die Ursachen für die schwer nachvollziehbaren Simulationsprogramme lagen in den beschränkten Ausdrucksmöglichkeiten komplexer mathematischer Formeln mit Hilfe prozeduraler Sprachen wie Pascal oder C. Schon für eine einzige Matrizenmultiplikation werden in der Regel mehrere verschachtelte Schleifen-Konstrukte benötigt.

Daraufhin entschloß ich mich eine Bibliothek für die Simulation einfacher Neuronaler Netze in Turbo-Pascal 6.0 unter MS-DOS zu entwickeln. Dies war ein großer Fehler, denn der schwache objektorientierte Ansatz dieser Programmiersprache sowie die von mir entwickelte Bibliothek führten weiterhin zu unübersichtlichen und aufwendigen Simulationsprogrammen. Deshalb wurde diese Lösung wieder verworfen.

Mit Hilfe dieser Erfahrungen wurde mir klar, daß eine Bibliothek für die Simulationen meiner Diplomarbeit folgende Anforderungen erfüllen muß.

- Um modernen Gesichtspunkten der Software-Entwicklung gerecht zu werden, muß die Bibliothek in einer Programmiersprache geschrieben werden, die einen starken objektorientierten Ansatz besitzt.

- Die gewählte Programmiersprache soll einerseits nah am Problem sein, um einen hohen Abstraktionsgrad zu erzielen, andererseits muß sie nah an der Maschine sein, um effiziente und schnelle Programme erzeugen zu können.

- Mathematische Formeln sollen im Programmtext möglichst so hingeschrieben werden können, wie sie in der Literatur abgedruckt werden. Dies erfordert, daß die gewählte Programmiersprache Ausdrücke mit benutzertdefinierten Datentypen auswerten kann.

- Die grundlegenden Elemente der Simulationen sind nicht die Neuronen, sondern mathematische Matrizen und Vektoren, denn alle iterativen Lernverfahren für Neuronale Netze sind in der Literatur in Form von Matrix- oder Vektorgleichungen angegeben.

- Die gewählte Programmiersprache soll auf möglichst vielen Betriebssystemplattformen verfügbar sein.

Die oben aufgeführten Anforderungen führten schließlich zur Wahl der Programmiersprache C++, die in [STR92] ausführlich behandelt wird. In diesem Kapitel werden die damit von mir entwickelten Bibliotheken für

- Matrizen und Vektoren, kurz MAC (MathArray Classes)
- Simulationen, kurz SIC (Simulation Classes)

vorgestellt und deren Einsatzmöglichkeiten genau beschrieben. Dabei beschäftigt sich der folgende Abschnitt mit den Zielplattformen und dem gewählten Compiler.

4.1. Zielplattformen und Compiler

Die Sprache C++ wird fortwährend weiterentwickelt. Keinesfalls genügen alle auf dem Markt vorhandenen C++ Compiler *einem* Standard. An einem C++-Standard arbeitet zur Zeit ein ANSI-Gremium names *X3J16*. Doch bis ANSI-C++ den Weg der Instanzen genommen hat, existieren als De-facto-Standard die AT&T-cfront-Versionen und vorab veröffentlichte ANSI-C++ Arbeitspapiere.

Die Bibliotheken MAC und SIC beruhen auf der AT&T-cfront-Version 2.1 und sind auf den Betriebssystemplattformen

- MS-DOS (ab Version 5.0)
- Windows (ab Version 3.1)
- Windows NT (ab Version 3.1)

verfügbar. Die Schnittstellen der Bibliotheken sind auf allen drei Plattformen identisch. Dabei wurden die Klassenhierarchien von MAC und SIC voll in die Foundation-Classes der Firma Microsoft eingebunden.

Bei den Microsoft-Foundation-Classes, kurz MFC, handelt es sich um eine sehr umfangreiche Klassenbibliothek für die oben genannten Betriebssysteme. Sie setzt sich aus weit über 100 Klassen zusammen und kapselt unter anderem fast die komplette Windows-Programmierschnittstelle.

Die von mir entwickelten Klassenbibliotheken MAC und SIC wurden auf den Compilern

- Microsoft C/C++ 7.0 (MFC 1.0)
- Visual C/C++ 1.0 und 1.5 (MFC 2.0 und 2.5)
- Visual C/C++ 1.0 für Windows NT (MFC 2.1)

übersetzt und sorgfältig ausgetestet. Sie sind zum aktuellen Zeitpunkt mit allen verfügbaren Versionen der MFC übersetzbar.

4.2. Klassenhierarchie und Modulabhängigkeiten

Dieser Abschnitt beschäftigt sich mit dem Aufbau der Klassenhierarchie der Bibliotheken MAC und SIC. Die Modulabhängigkeiten und eine Zuordnung der Klassen zu den einzelnen Modulen werden ebenfalls angegeben.

Die folgende Abbildung zeigt die Klassenhierarchie von MAC und SIC, vernachlässigt dabei aber die nicht direkt beteiltigten Klassen der MFC. Um die Zuordnung der einzelnen Klassen zu den Bibliotheken MAC, SIC und MFC zu vereinfachen, wird für jede wichtige Basisklasse durch einen Index angegeben, zu welcher Bibliothek sie gehört. Alle abgeleiteten Klassen einer solchen Basisklasse befinden sich dann in derselben Bibliothek.

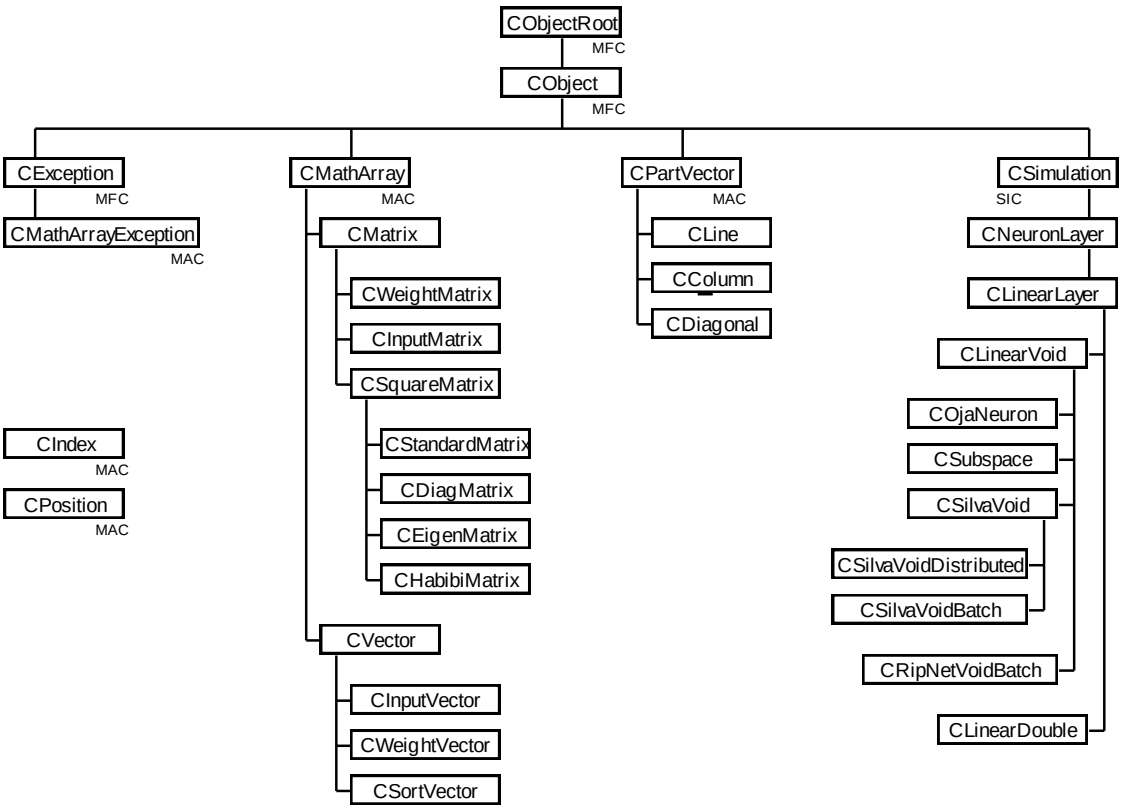


Abbildung 4-1 Klassenhierarchie der Bibliotheken MAC und SIC

Viele der oben aufgeführten Klassen besitzen einen beträchtlichen Funktionsumfang, der im nachfolgenden Abschnitt kurz beschrieben wird. Zusätzlich sind noch zahlreiche Operatorfunktionen vorhanden, die es erlauben mathematische Ausdrücke mit Matrix- *und* Vektorobjekten in den Programmtext zu schreiben. Beispielsweise lassen sich mit Hilfe der Klassenbibliothek MAC die mathematischen Ausdrücke

$$C_{YY} = W C_{XX} W^T$$

$$y = W x$$

und die Oja-Lernregel aus Kapitel 2.3.2.

$$w(t) = w(t-1) + \gamma \cdot y(t) \cdot (x(t) - w(t-1)\, y(t))$$

folgendermaßen als C++ Programmtext schreiben:

```
                                        // Definition der Konstanten
#define DIM   10                        // Dimension
#define GAMMA 0.1                       // Lernrate

                                        // Deklarationen
CSquareMatrix Cxx(DIM),                 // Korrelationsmatrizen
              Cyy(DIM);
CWeightMatrix W(DIM);                   // Gewichtsmatrix
CVector       y(DIM);                   // Ausgabevektor
CWeightVector w(DIM);                   // Gewichtsvektor

                                        // Lernregel
Cyy = W * Cxx * W.GetTransponate();     // Korrelationen
y = W * x;                              // Ausgabevektor

w += GAMMA * y * (x - w * y);           // Oja-Lernregel
```

Im Anhang meiner Diplomarbeit ist die Deklaration jeder einzelnen Klasse aufgeführt, so daß der Funktionsumfang einer Klasse dort entnommen werden kann.

Um die implementierten Module klein und überschaubar zu halten, wurden die Klassen der Bibliotheken MAC und SIC auf zahlreiche Dateien verteilt. Jedes Modul setzt sich dabei aus der Implementierung (Textdatei mit Endung .CPP) und einer öffentlichen Schnittstelle (Textdatei mit Endung .H) zusammen.

In der folgenden Abbildung wird dargestellt, in welchem Modul eine bestimmte Klasse der MAC gefunden werden kann.

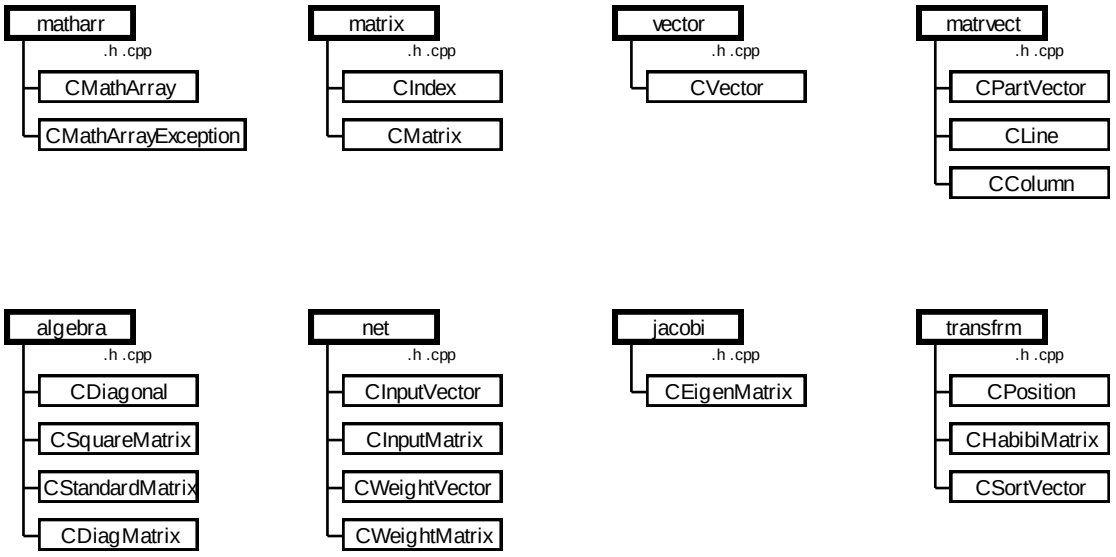


Abbildung 4-2 Zuordnung von Klassen der MAC zu den Modulen

Für die Klassenbibliothek der Simulationen SIC wird ebenfalls eine Zuordnung der Klassen zu den Modulen angegeben.

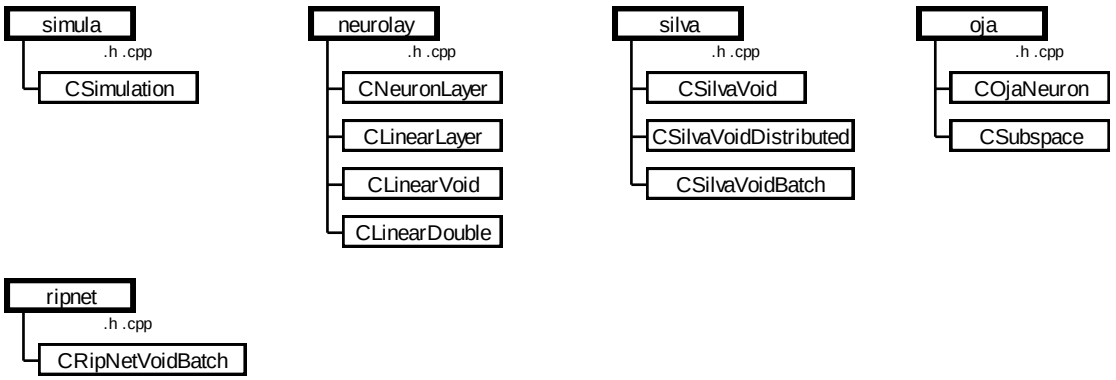


Abbildung 4-3 Zuordnung der Klassen von SIC zu den Modulen

Aus den obigen Abbildungen wird ersichtlich, daß sich die Bibliothek MAC aus acht und die Bibliothek SIC aus fünf Modulen zusammensetzt. Falls ein Benutzerprogramm bestimmte Klassen dieser Bibliotheken verwendet, müssen für dieses Programm beim Linken die zugeordneten Module angegeben werden.

Zusätzlich bestehen aber noch Abhängigkeiten der Module von MAC und SIC untereinander. Diese Abhängigkeiten werden in der folgenden Abbildung dargestellt.

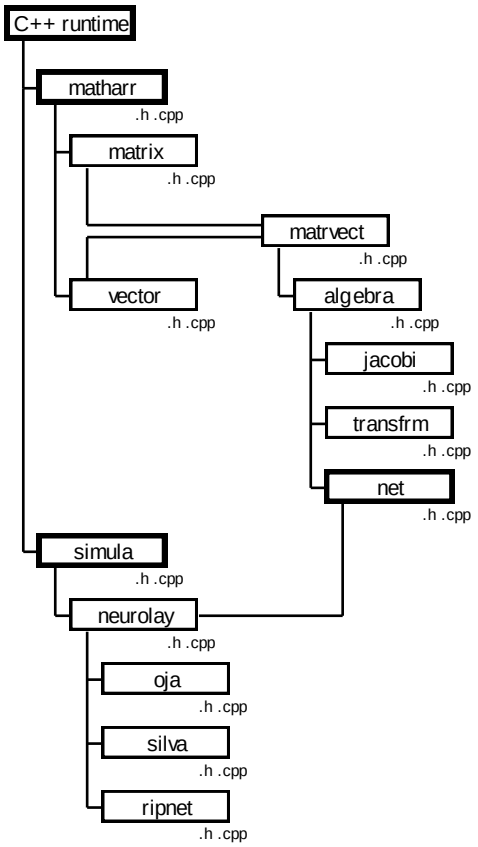


Abbildung 4-4 Modulabhängigkeiten der Klassenbibliotheken MAC und SIC

Die oben dargestellten Modulabhängigkeiten müssen beim Linken eines Benutzerprogramms berücksichtigt werden. Verwendet das Benutzerprogramm beispielsweise das Modul *matrvect*, dann müssen beim Linken die Module *matrvect*, *matrix*, *vector* und *matharr* angegeben werden, da *matrvect* von den Modulen *matrix*, *vector* und *matharr* abhängig ist.

Die Module *matharr* sowie *simula* bilden die Basis der Bibliotheken MAC bzw. SIC und werden deshalb in der obigen Abbildung hervorgehoben. Das Modul *net* wird hervorgehoben, weil es sich hierbei um die einzige (und wohl definierte) Schnittstelle zwischen den Bibliotheken MAC und SIC handelt.

4.3. Kurzbeschreibung der Klassen

In den folgenden beiden Abschnitten wird eine Kurzbeschreibung jeder einzelnen Klasse der Bibliotheken MAC und SIC angegeben. Um die Beschreibung kurz und übersichtlich zu halten, werden beide Bibliotheken gesondert betrachtet und die Klassen jeder Bibliothek in logische Gruppen eingeteilt.

4.3.1. Die Klassenbibliothek MAC

Die Bibliothek MAC ermöglicht eine einfache Verarbeitung von mathematischen Matrizen und Vektoren. Durch zahlreiche überladene Operatorfunktionen ist es möglich, sehr komplexe Formeln auf hohem Abstraktionsniveau direkt in den C++-Quelltext zu schreiben. Alle Klassen sind stream-fähig und unterstützen den Serialisierungsmechanismus der MFC, so daß für Matrizen und Vektoren eine Ein- und Ausgabe auf dem Bildschirm sowie eine Speicherung auf Datenträgern möglich ist.

Die Klassenbibliothek MAC setzt sich aus 20 Klassen mit über 250 Mitglieds- und Operatorfunktionen zusammen.

Basisklasse

Die Klasse *CMathArray* ist die Basisklasse der Bibliothek MAC. Sie enthält grundlegende Funktionen zur Speicherverwaltung und definiert Mitglieds- und Operatorfunktionen, die für Matrizen als auch Vektoren eingesetzt werden können.

Mit Hilfe der Klasse *CMathArrayException* wird eine Einbindung der Matrizen und Vektoren in den auf Makros beruhenden Exception-Mechanismus[17] der MFC erreicht.

[17] In der AT&T-cfront-Version 2.1 ist eine Exception-Unterstützung durch den C++-Compiler nicht vorgesehen.

Matrizen und Vektoren

Eine allgemeine Darstellung von Matrizen und Vektoren ermöglichen die Klassen *CMatrix* und *CVektor*. Sie stellen Operatorfunktionen für alle sinnvollen Grundrechenarten zur Verfügung und berücksichtigen auch Funktionen wie das Transponieren einer Matrix oder das Skalarprodukt von Vektoren.

Die Dimension aller Matrizen und Vektoren der MAC kann zur Laufzeit dynamisch geändert werden, ohne das der Inhalt einzelner Elemente verloren geht. Jede Klasse besitzt dabei ihren eigenen passenden Mechanismus zur Verkleinerung oder Vergrößerung der Elementanzahl.

Durch Vektoren darstellbare Teile einer Matrix werden durch die Klassen *CLine*, *CColumn* und *CDiagonal* zur Verfügung gestellt. Auf einzelne Elemente einer Matrix kann mit Hilfe des Zugriffsoperators [] und der Klasse *CIndex* zugegriffen werden.

Für die Simulation Neuronaler Netze sind die Klassen *CWeightMatrix*, *CInputMatrix*, *CWeightVector* und *CInputVector* vorgesehen. Mit Hilfe von *CWeightMatrix* und *CWeightVector* können Gewichte für iterative Lernverfahren Neuronaler Netze beschrieben werden, wobei eine Initialisierung mit Zufallswerten bereits integriert ist. Die Klassen *CInputMatrix* und *CInputVector* eignen sich zur Darstellung von Ein- und Ausgabemengen Neuronaler Netze. Für die einzelnen Elemente dieser Mengen können Auftrittswahrscheinlichkeiten angegeben werden.

Quadratische Matrizen

Für die Klasse der quadratischen Matrizen *CSquareMatrix* ergeben sich einige spezielle Aufgabengebiete, die mit Hilfe der von *CSquareMatrix* abgeleiteten Klassen abgedeckt werden können. Dabei beschreibt *CStandardMatrix* eine einfache Einheitsmatrix.

Die Klassen *CEigenMatrix* und *CDiagMatrix* ermöglichen eine Berechnung von Eigenvektoren und Eigenwerten einer reellwertigen symmetrischen Matrix nach dem klassischen Jacobi-Verfahren mit einer à priori Fehlerabschätzung. Mit Hilfe von *CSortVector* lassen sich anschließend die Eigenwerte mit dem Quicksort-Algorithmus sortieren und ihren entsprechenden Eigenvektoren zuordnen.

Die Klasse *CHabibiMatrix* beschreibt eine quadratische Korrelationsmatrix, die mit Hilfe der Korrelationsfunktion von Habibi und Wintz berechnet wird; siehe auch [HAB71]. In diesem Zusammenhang gibt *CPosition* die Position eines Punktes in einem zweidimensionalen Eingabefeld an.

4.3.2. Die Klassenbibliothek SIC

Die Bibliothek SIC ermöglicht in ihrer aktuellen Version die Simulation eines einschichtigen Neuronalen Netzes, wie es in Kapitel 2.3.1. vorgestellt wird. Dieses Netz besitzt einen Eingabevektor, einen Ausgabevektor und eine Menge von Gewichtsvektoren. Ein auf diesem Modell basierendes Lernverfahren kann durch Ergänzung einer Klasse und anschließender Definition einer einzigen Mitgliedsfunktion implementiert werden, wobei die Mitgliedsfunktion die Lernregel angibt. Simulationen können mit Hilfe des Serialisierungsmechanismus der MFC zwischengespeichert und zu einem späteren Zeitpunkt wieder fortgesetzt werden.

Die Klassenbibliothek SIC setzt sich aus 11 Klassen mit über 60 Mitglieds- und Operatorfunktionen zusammen.

Abstrakte Basisklassen

Bei der Klasse *CSimulation* handelt es sich um die abstrakte Basisklasse der Bibliothek SIC. Sie beschreibt eine beliebige Simulation, die eine vorgegebene Anzahl von Schritten läuft und zwischendurch unterbrochen werden kann.

Mit der abstrakten Klasse *CNeuronLayer* wird das oben beschriebene einschichtige Modell eines Neuronalen Netzes eingeführt und mit Hilfe der abstrakten Klasse *CLinearLayer* um eine lineare Ausgabefunktion ergänzt.

Die immer noch abstrakten Klassen *CLinearVoid* und *CLinearDouble* dienen als Basisklassen zur Ableitung benutzerdefinierter Simulationen. Beide Klassen unterscheiden sich durch die Datentypen der Zwischenergebnisse, die während einer Simulation anfallen können.

Durch die feine Aufteilung der oben beschriebenen Klassen ergeben sich sehr viele Eintrittspunkte für die Implementierung unterschiedlichster Arten von Simulationen. Beispielsweise könnte die Klasse *CSimulation* mit einer Klasse *CNeuronField* abgeleitet werden, um Simulationen mit Neuronenfeldern durchführen zu können.

Bereits implementierte Simulationen

Die Bibliothek enthält bereits eine Reihe konkreter Simulationen. Die in Kapitel 2.3.2. beschriebene Oja-Lernregel sowie das Subspace-Netzwerk aus Kapitel 2.3.3. werden durch die Klassen *COjaNeuron* und *CSubspace* zur Verfügung gestellt.

Die Klassen *CSilvaVoid*, *CSilvaVoidDistributed* und *CSilvaVoidBatch* ermöglichen Simulationen zur Datenorthonormalisierung nach einem Modell von Silva und Almeida, das in [SIL91] vorgestellt wurde.

Die Klasse *CRipNetVoidBatch* beschreibt die Simulation eines Modells zur neuronalen Datenkompression, welches in Kapitel 3. hergeleitet und untersucht wird.

4.4. Debugging-Unterstützung

Die Möglichkeiten zum Debuggen der Bibliotheken MAC und SIC beruhen auf den Mechanismen der MFC. Alle Klassen von MAC und SIC nutzen diese Mechanismen sehr intensiv. Da die eigentlichen Debugging-Möglichkeiten von der MFC zur Verfügung gestellt werden, beschränkt sich dieser Abschnitt auf eine Vorstellung der wichtigsten Gesamtkonzepte.

Grundsätzlich existieren zwei verschiedene Versionen der Bibliotheken MAC und SIC, nämlich je eine Debug-Version und eine Release-Version. Die Release-Versionen enthalten keinerlei Debug-Informationen und dienen zur Verwendung der Bibliotheken nach einer Testphase.

Während der Testphase kommen die Debug-Versionen von MAC und SIC zum Einsatz. Sie wurden mit Hilfe der definierten Konstanten _DEBUG übersetzt und enthalten zusätzlich volle Debug-Informationen. Die Konstante _DEBUG dient als Schalter für die bedingte Compilierung mit *#ifdef* oder *#ifndef* unter C++. Insbesondere werden mit

```
#ifdef _DEBUG

// Quelltext für die Debug-Version
// ...

#endif //_DEBUG
```

jene Abschnitte im Quelltext geklammert, die nur in den Debug-Versionen der Bibliotheken zur Verfügung stehen sollen.

Die Mitgliedsfunktionen *AssertValid* und *Dump* zur Debug-Unterstützung profitieren von diesem Mechanismus. Sie sind nur in den Debug-Versionen der Bibliotheken vorhanden, da sie mit

```
class CBeispiel                         // Nur eine Beispielklasse
{
      // ...

#ifdef _DEBUG                           // Debug-Unterstützung
      virtual void AssertValid() const;
      virtual void Dump(CDumpContext &dc) const;
#endif // _DEBUG

      // ...
};
```

deklariert und später auf dieselbe Art und Weise definiert werden.

Mit *AssertValid* kann ein Objekt einer bestimmten Klasse seinen internen Status überprüfen und im Falle eines Fehlers einen Programmabbruch durchführen. In diesem Fall werden Dateiname und Zeilennummer der Abbruchstelle im Quelltext zusätzlich auf dem Bildschirm ausgegeben. Beispielsweise überprüft die Klasse *CMathArray* mit dieser Funktion, ob verwendeter Speicher korrekt mit dem C++-Operator *new* reserviert wurde und vermeidet damit Zugriffe auf ungültige Speicherbereiche.

Die Mitgliedsfunktion *Dump* erlaubt es einem Objekt seinen Inhalt in kommentierter und lesbarer Form auf einem sogenannten Dump-Kontext auszugeben. Ein Dump-Kontext kann ein Textbildschirm, eine Datei oder ein spezielles Fenster vom Debugger sein.

Die Unterscheidung zwischen Debug- und Release-Version der Bibliotheken MAC und SIC bietet einen entscheidenden Vorteil. Sie erlaubt es eine Bibliothek mit einer großen Anzahl von Überprüfungen und Selbsttests auszustatten, ohne Rücksicht auf die Laufzeit nehmen zu müssen. Denn alle Überprüfungsmechanismen sind durch die bedingte Compilierung in den Release-Versionen der Bibliotheken nicht mehr vorhanden und beeinträchtigen daher auch nicht mehr die Laufzeit eines Programms.

Für jede einzelne Klasse der Bibliotheken MAC und SIC existiert ein eigenes Testprogramm, welches die oben beschriebenen Überprüfungsmechanismen nutzt und alle wichtigen Mitglieds- und Operatorfunktionen ausführlich testet. Alle Dateinamen der Testprogramme beginnen mit dem Wort *test*.

Es existieren noch eine ganze Reihe fortgeschrittener Debugging- und Analyse-Möglichkeiten, welche durch die MFC zur Verfügung gestellt und in den entsprechenden Handbüchern von Microsoft Visual C/C++ beschrieben werden.

5. Diskussion und Ausblick

Bereits nach dem Lesen der ersten Kapitel der vorliegenden Diplomarbeit wird deutlich, daß ich sehr viele Grundlagen aus den Bereichen Informationstheorie und Transform Coding *neu* erarbeitet habe. Es war für mich ein sehr wichtiges Anliegen, daß von mir selbständig erarbeitete Modell eines Neuronalen Netzes zur Datenorthonormalisierung auf ein sauberes mathematisches Fundament zu stellen.

Eine reine Literaturzusammenfassung wichtiger Ergebnisse der beiden oben genannten Wissenschaftsgebiete konnte dieser Anforderung nicht gerecht werden, da in diesem Fall sehr viele Fragen unbeantwortet geblieben wären. Daher habe ich alle wichtigen Aussagen meiner Diplomarbeit auch bewiesen. Die so entstandenen Beweisketten von den Grundlagen bis hin zu den zentralen Aussagen stützen sich auf wichtige Definitionen der Literatur.

Ich möchte an dieser Stelle die *wichtigsten* Ergebnisse meiner Diplomarbeit noch einmal zusammenfassen.

1. In Kapitel 2.1. wird mit Hilfe der Informationstheorie gezeigt, daß durch die Datenorthonormalisierung eine möglichst störsichere Übertragung von Daten über einen gestörten Übertragungskanal gegeben ist. Sind die zu übertragenden Daten zusätzlich normalverteilt, dann besitzen sie durch die Dekorrelation der Datenorthonormalisierung einen maximalen mittleren Informationsgehalt.

   Durch die Ergebnisse des Kapitels 2.1. wird geklärt, unter welchen Voraussetzungen der Einsatz der Datenorthonormalisierung wesentliche Vorteile bietet. Damit wird auch motiviert, *warum* es sich lohnt sich mit dieser Form von Transformation auseinanderzusetzen.

2. Es wird in Kapitel 2.2. eine durch Beweise fundierte Einführung in die Thematik des Transform Coding gegeben. Dabei werden die Eigenschaften der Transformationsmatrix, der Kovarianzmatrix und der Korrelationsmatrix genau untersucht. Es wird gezeigt, daß alle klassischen linearen Transformationen aus dem Bereich des Transform Coding ein orthogonales Transformationskernel besitzen, welches eine triviale Umkehrung der Transformation ermöglicht.

   Weiterhin wird bewiesen, daß der mittlere quadratische Sampling-Fehler bei der Rekonstruktion von approximativ komprimierten Daten minimiert werden kann, wenn für die lineare Transformation eine orthogonale Transformationsmatrix verwendet wird, welche in ihren Zeilen die orthonormierten Eigenvektoren der Korrelationsmatrix der Eingabedaten enthält.

Diese Ergebnisse sind für das Kapitel 3. meiner Diplomarbeit sehr wichtig, weil dort die bisher verwendeten Beweisverfahren zur Minimierung des mittleren quadratischen Sampling-Fehlers in ähnlicher Form für die Datenorthonormalisierung wiederverwendet werden können.

3. In Kapitel 3. wird ein von mir entwickeltes biologisch plausibles Modell eines Neuronalen Netzes vorgestellt, welches eine Datenorthonormalisierung durchführen kann.

   Darüberhinaus wird mit Hilfe einiger mathematischer Sätze gezeigt, daß die Datenorthonormalisierung nicht nur ein datenverarbeitendes Pre-Processing für andere Kodierungsverfahren darstellt, sondern das sie auch als Verfahren zur approximativen Datenkompression im Sinne des Transform Coding eingesetzt werden kann. Insbesondere wird eine inverse Transformation zur Datenorthonormalisierung angegeben und somit gezeigt, daß es sich bei der Datenorthonormalisierung um eine umkehrbare lineare Transformation handelt.

   Auch hier konnte bewiesen werden, daß der mittlere quadratische Sampling-Fehler bei der Rekonstruktion von approximativ komprimierten Daten minimiert werden kann, wenn für die Datenorthonormalisierung eine Transformationsmatrix verwendet wird, welche in ihren Zeilen passend normierte Eigenvektoren der Korrelationsmatrix der Eingabedaten enthält.

Meine Diplomarbeit setzt sich sehr intensiv mit der Datenorthonormalisierung auseinander und gibt auch ein Modell für ein Neuronales Netz an, welches diese Transformation durchführen kann. Bisher unerwähnt blieb aber, daß dieses Neuronale Netz *noch nicht* sinnvoll zur approximativen Datenkompression eingesetzt werden kann.

Da diese Form der Kompression fehlerbehaftet ist, spielt die Qualität der dekomprimierten Daten eine wichtige Rolle. Als Gütekriterium für diese Qualität wurde der mittlere quadratische Sampling-Fehler verwendet. Dieser kann minimiert werden, wenn die zur Datenorthonormalisierung verwendete Transformationsmatrix passend normierte Eigenvektoren in den Zeilen enthält.

In Kapitel

3.3. wurde jedoch gezeigt, daß es für eine vorgegebene Eingabemenge einen kompletten Lösungsraum für die Transformationsmatrix zur Datenorthonormalisierung gibt, welcher die Eigenvektoren als Lösung beinhaltet.

Das hier vorgestellte Modell eines Neuronalen Netzes findet nun aber nicht unbedingt die spezielle Lösung der Eigenvektoren, sondern eine *beliebige* Lösung aus dem kompletten Lösungsraum. Daher kann mit diesem Modell nicht garantiert werden, daß bei einer Rekonstruktion von approximativ komprimierten Daten der mittlere quadratische Sampling-Fehler minimal wird.

Im zeitlichen Rahmen meiner Diplomarbeit war es mir nicht möglich dieses Problem zu lösen. Daher möchte ich zum Abschluß zwei Ansatzpunkte angeben, mit denen dieses Problem in einer zukünftigen Diplomarbeit gelöst werden könnte.

1. Unter Umständen existiert eine Anfangsbelegung für die Transformationsmatrix, so daß das iterative Lernverfahren des Neuronalen Netzes garantiert zu den Eigenvektoren als Lösung konvergiert.

2. Es sollte in irgendeiner Form möglich sein, die dem Neuronalen Netz zugrundeliegende Kostenfunktion aus Kapitel 3.2. so zu modifizieren, daß eine Konvergenz zu einem Eigenvektorsystem garantiert werden kann.

Sollte in Zukunft mit einer der oben genannten Modifikationen das Problem gelöst werden können, dann hätte meine Diplomarbeit die Grundlagen für ein praxistaugliches und biologisch plausibles Modell eines Neuronalen Netzes geliefert, mit dem eine informationsoptimale Kompression gestörter Daten durchgeführt werden kann.

Anhang 1: Beweise zum Kapitel 2.1.

Korollar 2-1: Es sei X eine Verbundquelle, welche sich aus n stochastisch unabhängigen Teilquellen X^1, X^2, ###, X^n zusammensetzt. Dann gilt für die Entropie von X

$$H(X) = H(X^1) + H(X^2) + \ldots + H(X^n)$$

Beweis 2-1: Die Entropie von X läßt sich schreiben als

$$H(X) = H(X^1, X^2, \ldots, X^n) = -\sum_{i=1}^{N_1}\sum_{j=1}^{N_2}\ldots\sum_{k=1}^{N_n} P(x_{1i}, x_{2j}, \ldots, x_{nk}) \cdot \ln P(x_{1i}, x_{2j}, \ldots, x_{nk})$$

und mit der gemeinsamen Wahrscheinlichkeitsfunktion

$$P(x_{1i}, x_{2j}, \ldots, x_{nk}) = P(x_{1i}) \cdot P(x_{2j}) \cdot \ldots \cdot P(x_{nk})$$

von X^1, X^2, ###, X^n im Falle stochastischer Unabhängigkeit folgt dann

$$= -\sum_{i=1}^{N_1}\sum_{j=1}^{N_2}\ldots\sum_{k=1}^{N_n} P(x_{1i}) \cdot P(x_{2j}) \cdot \ldots \cdot P(x_{nk}) \cdot \ln\left[P(x_{1i}) \cdot P(x_{2j}) \cdot \ldots \cdot P(x_{nk})\right]$$

$$= -\sum_{i=1}^{N_1}\sum_{j=1}^{N_2}\ldots\sum_{k=1}^{N_n} P(x_{1i}) \cdot P(x_{2j}) \cdot \ldots \cdot P(x_{nk}) \cdot \left[\ln P(x_{1i}) + \ln P(x_{2j}) + \ldots + \ln P(x_{nk})\right]$$

$$= -\sum_{i=1}^{N_1}\sum_{j=1}^{N_2}\ldots\sum_{k=1}^{N_n} P(x_{1i}) \cdot P(x_{2j}) \cdot \ldots \cdot P(x_{nk}) \cdot \ln P(x_{1i})$$

$$-\sum_{i=1}^{N_1}\sum_{j=1}^{N_2}\ldots\sum_{k=1}^{N_n} P(x_{1i}) \cdot P(x_{2j}) \cdot \ldots \cdot P(x_{nk}) \cdot \ln P(x_{2j})$$

...

$$-\sum_{i=1}^{N_1}\sum_{j=1}^{N_2}\ldots\sum_{k=1}^{N_n} P(x_{1i}) \cdot P(x_{2j}) \cdot \ldots \cdot P(x_{nk}) \cdot \ln P(x_{nk})$$

$$= \underbrace{-\sum_{i=1}^{N_1} P(x_{1i}) \cdot \ln P(x_{1i})}_{H(X^1)} \underbrace{\sum_{j=1}^{N_2} P(x_{2j})}_{1} \cdot \ldots \cdot \underbrace{\sum_{k=1}^{N_n} P(x_{nk})}_{1}$$

$$\underbrace{-\sum_{j=1}^{N_2} P(x_{2j}) \ln P(x_{2j})}_{H(X^2)} \underbrace{\sum_{i=1}^{N_1} P(x_{1i})}_{1} \cdot \ldots \cdot \underbrace{\sum_{k=1}^{N_n} P(x_{nk})}_{1}$$

...

$$\underbrace{-\sum_{k=1}^{N_n} P(x_{nk}) \ln P(x_{nk})}_{H(X^n)} \underbrace{\sum_{i=1}^{N_1} P(x_{1i})}_{1} \cdot \ldots \cdot \underbrace{\sum_{l=1}^{N_{n-1}} P(x_{n-1\,l})}_{1}$$

$$= H(X^1) + H(X^2) + \ldots + H(X^n) \qquad \text{q.e.d.}$$

Korollar 2-2: Es sei X eine Verbundquelle, die aus n stochastisch abhängigen Teilquellen X^1, X^2, ###, X^n besteht. Dann gilt für die Entropie von X

$$H(X) = H(X^1, X^2, \ldots, X^n) = H(X^1) + H_{X^1}(X^2) + \ldots + H_{X^1, X^2, \ldots X^{n-1}}(X^n)$$

Beweis 2-2: Die Entropie von X läßt sich schreiben als

$$H(X) = H(X^1, X^2, \ldots, X^n) = -\sum_{i=1}^{N_1} \sum_{j=1}^{N_2} \cdots \sum_{k=1}^{N_n} P(x_{1i}, x_{2j}, \ldots, x_{nk}) \ln P(x_{1i}, x_{2j}, \ldots, x_{nk})$$

und mit der gemeinsamen Wahrscheinlichkeitsfunktion

$$P(x_{1i}, x_{2j}, \ldots, x_{nk}) = P(x_{1i}) P_{x_{1i}}(x_{2j}) \cdot \ldots \cdot P_{x_{1i}, x_{2j}, \ldots, x_{n-1\,l}}(x_{nk})$$

von X^1, X^2, ###, X^n im Falle stochastischer Abhängigkeit folgt dann

$$= -\sum_{i=1}^{N_1} \sum_{j=1}^{N_2} \cdots \sum_{k=1}^{N_n} P(x_{1i}) P_{x_{1i}}(x_{2j}) \cdot \ldots \cdot P_{x_{1i}, x_{2j}, \ldots, x_{n-1\,l}}(x_{nk})$$

$$\ln \left[P(x_{1i}) P_{x_{1i}}(x_{2j}) \cdot \ldots \cdot P_{x_{1i}, x_{2j}, \ldots, x_{n-1\,l}}(x_{nk}) \right]$$

$$= -\sum_{i=1}^{N_1} \sum_{j=1}^{N_2} \cdots \sum_{k=1}^{N_n} P(x_{1i}) P_{x_{1i}}(x_{2j}) \cdot \ldots \cdot P_{x_{1i}, x_{2j}, \ldots, x_{n-1\,l}}(x_{nk})$$

$$\left[\ln P(x_{1i}) + \ln P_{x_{1i}}(x_{2j}) + \ldots + \ln P_{x_{1i}, x_{2j}, \ldots, x_{n-1\,l}}(x_{nk}) \right]$$

$$= -\sum_{i=1}^{N_1} \sum_{j=1}^{N_2} \cdots \sum_{k=1}^{N_n} P(x_{1i}) P_{x_{1i}}(x_{2j}) \cdot \ldots \cdot P_{x_{1i}, x_{2j}, \ldots, x_{n-1\,l}}(x_{nk}) \ln P(x_{1i})$$

$$-\sum_{i=1}^{N_1} \sum_{j=1}^{N_2} \cdots \sum_{k=1}^{N_n} P(x_{1i}) P_{x_{1i}}(x_{2j}) \cdot \ldots \cdot P_{x_{1i}, x_{2j}, \ldots, x_{n-1\,l}}(x_{nk}) \ln P_{x_{1i}}(x_{2j})$$

...

$$-\sum_{i=1}^{N_1}\sum_{j=1}^{N_2}\cdots\sum_{k=1}^{N_n} P(x_{1i})\, P_{x_{1i}}(x_{2j}) \cdots P_{x_{1i},x_{2j},\ldots,x_{(n-1)l}}(x_{nk}) \ln P_{x_{1i},x_{2j},\ldots,x_{(n-1)l}}(x_{nk})$$

$$= -\underbrace{\sum_{i=1}^{N_1} P(x_{1i}) \ln P(x_{1i})}_{H(X^1)} \underbrace{\sum_{j=1}^{N_2} P_{x_{1i}}(x_{2j})}_{1} \cdots \underbrace{\sum_{k=1}^{N_n} P_{x_{1i},x_{2j},\ldots,x_{(n-1)l}}(x_{nk})}_{1}$$

$$-\underbrace{\sum_{i=1}^{N_1} P(x_{1i}) \underbrace{\sum_{j=1}^{N_2} P_{x_{1i}}(x_{2j}) \ln P_{x_{1i}}(x_{2j})}_{H_{x_{1i}}(X^2)}}_{H_{X^1}(X^2)} \underbrace{\sum_{g=1}^{N_3} P_{x_{1i},x_{2j}}(x_{3g})}_{1} \cdots \underbrace{\sum_{k=1}^{N_n} P_{x_{1i},x_{2j},\ldots,x_{(n-1)l}}(x_{nk})}_{1}$$

...

$$+\underbrace{\sum_{i=1}^{N_1}\cdots\sum_{l=1}^{N_{n-1}} \underbrace{P(x_{1i})\cdots P_{x_{1i},\ldots,x_{(n-2)g}}(x_{(n-1)l})}_{P(x_{1i},x_{2j},\ldots,x_{(n-1)l})} \underbrace{\left(-\sum_{k=1}^{N_n} P_{x_{1i},\ldots,x_{(n-1)l}}(x_{nk}) \ln P_{x_{1i},\ldots,x_{(n-1)l}}(x_{nk})\right)}_{H_{x_{1i},x_{2j},\ldots,x_{(n-1)l}}(X^n)}}_{H_{X^1,X^2,\ldots,X^{n-1}}(X^n)}$$

$$= H(X^1) + H_{X^1}(X^2) + \ldots + H_{X^1,X^2,\ldots,X^{n-1}}(X^n) \qquad \text{q.e.d.}$$

Korollar 2-3: Es sei X eine n-dimensional normalverteilte Zufallsgröße mit einer Dichtefunktion

$$p(x_1, x_2, \ldots x_n) = \frac{1}{\sqrt{(2\pi)^n \det C_{XX}}}\, e^{-\frac{1}{2}\underbrace{\sum_{i=1}^{n}\sum_{j=1}^{n} C_{XX}^{-1}(i,j)\, x_i x_j}_{=x^T C_{XX}^{-1} x}} ,$$

wobei die Korrelationsmatrix C_{XX} berechnet wird durch

$$C_{XX}(i,j) = \langle x_i x_j \rangle = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\cdots\int_{-\infty}^{\infty} x_i x_j\, p(x_1, x_2, \ldots, x_n)\, dx_1 dx_2 \ldots dx_n .$$

Dann ist die Entropie von X gegeben durch

$$H(X) = \ln\sqrt{(2\pi e)^n \det C_{XX}} .$$

Beweis 2-3: Zunächst ergibt sich für den Teilausdruck ($\ln p(x_1, x_2, \ldots, x_n)$)

$$\ln p(x_1, x_2, \ldots, x_n) = \ln \frac{1}{\sqrt{(2\pi)^n \det C_{XX}}} e^{-\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} C_{XX}^{-1}(i,j)\, x_i x_j}$$

$$= \ln \frac{1}{\sqrt{(2\pi)^n \det C_{XX}}} + \ln e^{-\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} C_{XX}^{-1}(i,j)\, x_i x_j}$$

$$= \ln \frac{1}{\sqrt{(2\pi)^n \det C_{XX}}} - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} C_{XX}^{-1}(i,j)\, x_i x_j$$

Die Entropie H(X) läßt sich nun berechnen durch

$$H(X) = -\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\cdots\int_{-\infty}^{\infty} p(x_1, x_2, \ldots, x_n) \ln p(x_1, x_2, \ldots, x_n)\, dx_1 dx_2 \ldots dx_n$$

und damit folgt dann

$$= -\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\cdots\int_{-\infty}^{\infty} p(x_1, x_2, \ldots, x_n) \ln \frac{1}{\sqrt{(2\pi)^n \det C_{XX}}}$$

$$+ \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} C_{XX}^{-1}(i,j)\, x_i x_j \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\cdots\int_{-\infty}^{\infty} p(x_1, x_2, \ldots, x_n)\, dx_1 dx_2 \ldots dx_n$$

$$= -\ln \frac{1}{\sqrt{(2\pi)^n \det C_{XX}}} \underbrace{\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\cdots\int_{-\infty}^{\infty} p(x_1, x_2, \ldots, x_n)\, dx_1 dx_2 \ldots dx_n}_{1}$$

$$+ \frac{1}{2}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\cdots\int_{-\infty}^{\infty} p(x_1, x_2, \ldots, x_n) \sum_{i=1}^{n}\sum_{j=1}^{n} C_{XX}^{-1}(i,j)\, x_i x_j\, dx_1 dx_2 \ldots dx_n$$

$$= \ln \sqrt{(2\pi)^n \det C_{XX}} + \frac{1}{2}\left\langle \sum_{i=1}^{n}\sum_{j=1}^{n} C_{XX}^{-1}(i,j)\, x_i x_j \right\rangle$$

$$= \ln \sqrt{(2\pi)^n \det C_{XX}} + \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} C_{XX}^{-1}(i,j) \left\langle x_i x_j \right\rangle$$

und weiterhin ergibt sich

$$= \ln\sqrt{(2\pi e)^n \det C_{XX}} + \frac{1}{2}\sum_{i=1}^{n}\underbrace{\sum_{j=1}^{n} C_{XX}^{-1}(i,j)\, C_{XX}(i,j)}_{1}$$

$$= \ln\sqrt{(2\pi e)^n \det C_{XX}} + \frac{1}{2}n = \ln\sqrt{(2\pi e)^n \det C_{XX}} + \frac{1}{2}\ln e^n$$

$$= \ln\sqrt{(2\pi e)^n \det C_{XX}} + \ln\sqrt{e^n} = \ln\sqrt{(2\pi e)^n e^n \det C_{XX}}$$

$$= \ln\sqrt{(2\pi e)^n \det C_{XX}} \qquad \text{q.e.d.}$$

Korollar 2-4: Es seien die Verbundquellen Y, Z und Φ wie im Modell in Abbildung 2-7 gegeben. Weiterhin seien alle geforderten Voraussetzungen für das Modell erfüllt. Dann gilt die Beziehung

$$H(Z) = H(\Phi) .$$

Beweis 2-4: Nach [PAP91] gilt für eine lineare Transformation von stetigen Zufallsgrößen und deren Entropie der folgende Sachverhalt.

Falls stetige Zufallsgrößen V^1, V^2, …, V^n und W^1, W^2, …, W^n durch eine Linearkombination der Art

$$V^i = a_{i1}W^1 + a_{i2}W^2 + \ldots + a_{in}W^n \qquad\qquad i = 1, 2, \ldots, n$$

miteinander in Beziehung stehen, dann gilt für deren Entropie

$$H(V^1, V^2, \ldots, V^n) = H(W^1, W^2, \ldots, W^n) + \ln|\det A|,$$

falls $\det A \neq 0$. Die $n \times n$ Matrix A ist gegeben durch $A = [a_{ij}]_{i,\, j\, =\, 1,\, 2,\, \ldots,\, n}$.

Für den Fall der Verbundquellen Y, Z und Φ dieses Korollars ergibt sich dann

$$Y = a_{11}Y + a_{12}\Phi \qquad\qquad a_{11} = 1 \;\wedge\; a_{12} = 0$$

$$Z = a_{21}Y + a_{22}\Phi \qquad\qquad a_{21} = 1 \;\wedge\; a_{22} = 1.$$

und

$$H(Z) = H(\Phi) + \ln|\det A| = H(\Phi) + \ln 1$$

$$= H(\Phi) \qquad \text{q.e.d.}$$

Satz 2-2: Es seien die Verbundquellen Y, Z und Φ wie im Modell in Abbildung 2-7 gegeben und alle geforderten Voraussetzungen für dieses Modell seien erfüllt. Dann ist die Transinformation H(Y;Z) gegeben durch

$$H(Y;Z) = \frac{1}{2}\ln\det\left(\frac{1}{P_\Phi}C_{YY} + I\right).$$

Beweis 2-2: Die Transinformation ist nach [MIL90] auch im kontinuierlichen Fall gegeben durch

$$\begin{aligned}
H(Y;Z) &= H(Y+\Phi;Z+\Phi;Z) \\
&= H(Y+\Phi;Z+\Phi;\Phi) \\
&= H(Y+\Phi;Z+\Phi) - H(Y+\Phi;\Phi) \\
&= H(Z+\Phi;\Phi)
\end{aligned}$$

Da die Verbundquellen Z und Φ normalverteilt sind, kann deren Entropie mit Hilfe von Korollar 2-3 bestimmt werden. Dann erhält man

$$\begin{aligned}
H(Y;Z) &= \ln\sqrt{(2\pi e)^n \det C_{ZZ}} - \ln\sqrt{(2\pi e)^n \det C_{\Phi\Phi}} \\
&= \ln\sqrt{(2\pi e)^n}\ln\sqrt{\det C_{ZZ}} - \left(\sqrt{(2\pi e)^n}\ln\sqrt{\det C_{\Phi\Phi}}\right) \\
&= \ln\sqrt{\det C_{ZZ}} - \ln\sqrt{\det C_{\Phi\Phi}} \\
&= \frac{1}{2}\ln\det C_{ZZ} - \ln\det C_{\Phi\Phi} = \frac{1}{2}\left(\det C_{ZZ} + \ln\frac{1}{\det C_{\Phi\Phi}}\right) \\
&= \frac{1}{2}\ln\det C_{ZZ} + \ln\det C_{\Phi\Phi}^{-1} = \frac{1}{2}\ln\det C_{ZZ}\det C_{\Phi\Phi}^{-1} \\
&= \frac{1}{2}\ln\det C_{ZZ}C_{\Phi\Phi}^{-1} = \left(\frac{1}{2}\ln\det C_{YY} + C_{\Phi\Phi}\ C_{\Phi\Phi}^{-1}\right) \\
&= \frac{1}{2}\ln\det C_{YY}C_{\Phi\Phi}^{-1} + I = \left(\frac{1}{2}\ln\det\left(\frac{1}{C_{YY}}C_\Phi\right) + I\right) \\
&= \frac{1}{2}\ln\det\left(\frac{1}{P_\Phi}C_{YY} + I\right)
\end{aligned}$$

q.e.d.

Satz 2-3: Es sei durch C_{YY} eine n × n Diagonalmatrix gegeben, deren Diagonalelemente in der Form

$$\frac{1}{P_\Phi} C_{YY}\ i,i\ +1 \qquad\qquad i = 1, 2, \ldots, n$$

darstellbar sind. Dann besitzt die Funktion

$$f(c_{11}, c_{22}, \ldots, c_{nn}) = \prod_{i=1}^{n} \left(\frac{1}{P_\Phi} c_{ii} + 1 \right) \qquad\qquad c_{ii} = C_{YY}\ i,i$$

unter der Nebenbedingung

$$\sum_{i=1}^{n} c_{ii} = P_Y$$

genau dann ein relatives Extremum, wenn die Matrix C_{YY} gegeben ist durch

$$C_{YY} = \frac{1}{n} P_Y \left(E \right).$$

Beweis 2-3: Für den Beweis wird die Methode der Lagrange-Multiplikatoren verwendet. Dazu wird mit einer Funktion

$$g(c_{11}, c_{22}, \ldots, c_{nn}) = P_Y - \sum_{i=1}^{n} c_{ii}$$

sowie der Funktion

$$f(c_{11}, c_{22}, \ldots, c_{nn}) = \prod_{i=1}^{n} \left(\frac{1}{P_\Phi} c_{ii} + 1 \right)$$

und einer vorläufig unbestimmten Konstanten λ die Lagrange-Funktion L mit dem Multiplikator λ

$$L(c_{11}, c_{22}, \ldots, c_{nn}, \lambda) = f(c_{11}, c_{22}, \ldots, c_{nn}) + \lambda g(c_{11}, c_{22}, \ldots, c_{nn})$$

gebildet. Für die zu bestimmenden n + 1 Größen c_{11}, c_{22}, …, c_{nn}, λ wird das Gleichungssystem

$$\begin{aligned} &f(c_{11}, c_{22}, \ldots, c_{nn}) = 0 \\ &\frac{L}{c_{ii}}(c_{11}, c_{22}, \ldots, c_{nn}, \lambda) = 0 \end{aligned} \qquad\qquad i = 1, 2, \ldots, n$$

mit n + 1 Gleichungen untersucht. Die erste Gleichung ist genau dann erfüllt, wenn die geforderte Nebenbedingung gilt. Die verbleibenden n Gleichungen werden im folgenden untersucht.

$$\frac{\partial L}{\partial c_{jj}}\left(c_{11}, c_{22}, \ldots, c_{nn}, \lambda\right) = \frac{\partial}{\partial c_{jj}}\left(\prod_{i=1}^{n}\frac{1}{P_\Phi}c_{ii} + 1\right)\lambda P_Y - \sum_{i=1}^{n} c_{ii}$$

$$= \frac{\partial}{\partial c_{jj}}\prod_{i=1}^{n}\frac{1}{P_\Phi}c_{ii} + 1\frac{\partial}{\partial c_{jj}}\lambda P_Y - \lambda\sum_{i=1}^{n} c_{ii}$$

$$= \frac{\partial}{\partial c_{jj}}\frac{1}{P_\Phi}c_{jj} + 1\prod_{i=1,i\neq j}^{n}\frac{1}{P_\Phi}c_{ii} + 1\frac{\partial}{\partial c_{jj}}\lambda P_Y - \lambda c_{jj} - \lambda\sum_{i=1,i\neq j}^{n} c_{ii}$$

$$= \frac{\partial}{\partial c_{jj}}\frac{1}{P_\Phi}c_{jj}\prod_{i=1,i\neq j}^{n}\frac{1}{P_\Phi}c_{ii} + 1\underbrace{\frac{\partial}{\partial c_{jj}}\prod_{i=1,i\neq j}^{n}\frac{1}{P_\Phi}c_{ii} + 1}_{0}\underbrace{\frac{\partial}{\partial c_{jj}}\lambda P_Y}_{0}$$

$$+ \frac{\partial}{\partial c_{jj}}\left(-\lambda c_{jj}\right) + \underbrace{\frac{\partial}{\partial c_{jj}}\lambda\sum_{i=1,i\neq j}^{n} c_{ii}}_{0}$$

$$= \frac{1}{P_\Phi}\prod_{i=1,i\neq j}^{n}\frac{1}{P_\Phi}c_{ii} + 1\lambda$$

Unter Berücksichtigung des obigen Gleichungssystems folgt

$$\frac{1}{P_\Phi}\prod_{i=1,i\neq j}^{n}\frac{1}{P_\Phi}c_{ii} + 1\lambda \qquad j = 1, 2, \ldots, n$$

und für beliebige j, k mit $j \neq k$ gilt dann

$$\prod_{i=1,i\neq j}^{n}\frac{1}{P_\Phi}c_{ii} + 1 = \prod_{i=1,i\neq k}^{n}\frac{1}{P_\Phi}c_{ii} + 1 \qquad j, k = 1, 2, \ldots, n;\ j \neq k$$

$$\Rightarrow \frac{1}{P_\Phi}c_{kk} + 1\prod_{\substack{i=1\\ i\neq j, i\neq k}}^{n}\frac{1}{P_\Phi}c_{ii} + 1 = \frac{1}{P_\Phi}c_{jj} + 1\prod_{\substack{i=1\\ i\neq k, i\neq j}}^{n}\frac{1}{P_\Phi}c_{ii} + 1$$

$$\Rightarrow \frac{1}{P_\Phi}c_{kk} + 1 = \frac{1}{P_\Phi}c_{jj} + 1$$

$$\Rightarrow c_{kk} = c_{jj} \qquad j, k = 1, 2, \ldots, n;\ j \neq k.$$

Aus der Lösung des Gleichungssystems ergibt sich also, daß alle Diagonalelemente der Matrix C_{YY} den gleichen Wert besitzen müssen, um ein relatives Extremum für die Funktion f(.) zu erhalten. Mit der Nebenbedingung

$$\sum_{i=1}^{n} c_{ii} = P_Y$$

folgt

$$c_{ii} = \frac{1}{n} P_Y \qquad i = 1, 2, \ldots, n$$

und damit auch

$$C_{YY} = \frac{1}{n} P_Y \mathbf{I}_n \qquad \text{q.e.d.}$$

Anhang 2: Beweise zum Kapitel 2.2.

Satz 2-7: Es sei y = A x eine lineare Transformation mit einer orthogonalen $n^2 \times n^2$ Matrix A, die als Zeilenvektoren die orthonormierten Eigenvektoren $\{e^1, e^2, ..., e^{n^2}\}$ der Korrelationsmatrix C_{XX} enthält. Diese Eigenvektoren seien gemäß der Größe der zugehörigen Eigenwerte $\lambda_1, \lambda_2, \ldots, \lambda_{n^2}$ absteigend geordnet. Dann wird der durch

$$\varepsilon_s^2 = \left\langle \left| x - x' \right|^2 \right\rangle$$

definierte mittlere quadratische Sampling-Fehler minimal.

Beweis 2-7: Annahme: Es sei A eine beliebige orthogonale Transformationsmatrix und es gelte $y_i = a^{iT} x$ für i = 1, 2, …, n^2.

Nach Satz 2-6 ist dann der mittlere quadratische Sampling-Fehler gegeben durch

$$\varepsilon_s^2 = \sum_{i=m+1}^{n^2} \left\langle y_i^2 \right\rangle .$$

Da für jeden Term der Summe $\langle y_i^2 \rangle \geq 0$ gilt, wird diese Summe minimal, falls jeder einzelne Term minimal ist. Aus diesem Grund wird eine Kostenfunktion

$$R(a^i) = \left\langle y_i^2 \right\rangle$$

$$= \left\langle a^{iT} x x^T a^i \right\rangle = a^{iT} \left\langle x x^T \right\rangle a^i = a^{iT} C_{XX} a^i \qquad m + 1 \leq i \leq n^2$$

definiert, für die ein globales Minimum unter der Nebenbedingung $|a^i|^2 = 1$ gesucht wird. Zur Bestimmung der relativen Extrema von R(.) wird die Methode der Lagrange-Multiplikatoren verwendet. Dazu wird mit einer Funktion

$$g(a^i) = \left| a^i \right|^2 - 1$$

sowie der Funktion

$$f(a^i) = a^{iT} C_{XX} a^i$$

und einer vorläufig unbestimmten Konstanten μ_i die Lagrange-Funktion L mit dem Multiplikator μ_i

$$L(a^i, \mu_i) = f(a^i) + \mu_i \, g(a^i)$$

gebildet.

Für die zu bestimmenden Größen a^i und μ_i wird das Gleichungssystem

$$\begin{pmatrix} \nabla_{a^i} L(a^i, \mu_i) \\ L(a^i, \mu_i) \end{pmatrix} = 0$$

untersucht. Die erste Gleichung ist genau dann erfüllt, wenn die geforderte Nebenbedingung gilt. Für die zweite Gleichung ergibt sich

$$\nabla_{a^i} L(a^i, \mu_i) = 2 C_{XX} a^i + 2\mu_i a^i.$$

Unter Berücksichtigung des obigen Gleichungssystems folgt

$$2 C_{XX} a^i = -2\mu_i a^i$$

$$\Rightarrow C_{XX} a^i = -\mu_i a^i.$$

Das Gleichungssystem besitzt für $\mu_i = -\lambda_i$ die normierten Eigenvektoren $\{e^1, e^2, ..., e^{n^2}\}$ der Korrelationsmatrix C_{XX} als Lösung. Unter Berücksichtigung der Nebenbedingung besitzt die Kostenfunktion R(.) genau bei jedem dieser Eigenvektoren ein relatives Extremum, welches gegeben ist durch

$$R(e^i) = e^{iT} C_{XX} e^i$$

$$= e^{iT} \lambda_i e^i = \lambda_i e^{iT} e^i = \lambda_i \left| e^i \right|^2 = \lambda_i.$$

Ein lokales Minimum der Kostenfunktion R(.) ist also durch den Eigenvektor e^i mit dem kleinsten Eigenwert λ_i gegeben.

Damit der mittlere quadratische Sampling-Fehler

$$\varepsilon_s^2 = \sum_{i=m+1}^{n^2} \left\langle y_i^2 \right\rangle = \sum_{i=m+1}^{n^2} R(e^i) = \sum_{i=m+1}^{n^2} \lambda_i$$

minimal wird, müssen für die Berechnung der vernachlässigten Koeffizienten

$$y_i = e^{iT} x \qquad\qquad m + 1 \le i \le n^2$$

gerade die Eigenvektoren mit den kleinsten Eigenwerten verwendet werden. Sind die Eigenvektoren in den Zeilen der Transformationsmatrix A abgelegt und gemäß der Größe ihrer zugehörigen Eigenwerte absteigend geordnet, dann ist diese Bedingung erfüllt. q.e.d.

Anhang 3: Beweise zum Kapitel 3.

Das folgende Korollar wird für den Beweis des nachfolgenden Satzes benötigt.

Korollar 3-1: Es sei eine Verbundquelle X mit n Teilquellen gegeben. Dann gilt für die einzelnen Zeichen der Worte $x \in X$ folgende Beziehung

$$\sum_{i=1}^{n}\sum_{j=1,i\neq j}^{n}\left\langle x_i\, x_j\right\rangle^m = 2\sum_{i=1,i\neq k}^{n}\left\langle x_k\, x_i\right\rangle^m + \sum_{i=1,i\neq k}^{n}\sum_{\substack{j=1,j\neq k\\ i\neq j}}^{n}\left\langle x_i\, x_j\right\rangle^m \qquad k = 1, 2, \ldots, n;\ m \in \aleph.$$

Beweis 3-1: (durch vollständige Induktion nach n)

Induktionsanfang n = 2:

$$\sum_{i=1}^{2}\sum_{j=1,i\neq j}^{2}\left\langle x_i\, x_j\right\rangle^m = \left\langle x_1\, x_2\right\rangle^m + \left\langle x_2\, x_1\right\rangle^m$$

$$= 2\left\langle x_1\, x_2\right\rangle^m + 0 = 2\left\langle x_2\, x_1\right\rangle^m + 0 = 2\sum_{i=1,i\neq k}^{2}\left\langle x_k\, x_i\right\rangle^m + 0$$

$$= 2\sum_{i=1,i\neq k}^{2}\left\langle x_k\, x_i\right\rangle^m + \sum_{i=1,i\neq k}^{2}\sum_{\substack{j=1,j\neq k\\ i\neq j}}^{2}\left\langle x_i\, x_j\right\rangle^m \qquad k = 1, 2;\ m \in \aleph$$

Induktionsschritt $n \rightarrow n + 1$: Sei

$$\sum_{i=1}^{n}\sum_{j=1,i\neq j}^{n}\left\langle x_i\, x_j\right\rangle^m = 2\sum_{i=1,i\neq k}^{n}\left\langle x_k\, x_i\right\rangle^m + \sum_{i=1,i\neq k}^{n}\sum_{\substack{j=1,j\neq k\\ i\neq j}}^{n}\left\langle x_i\, x_j\right\rangle^m \qquad k = 1, 2, \ldots, n;\ m \in \aleph$$

bereits bewiesen (Induktionsvoraussetzung). Dann ist zu zeigen

$$\sum_{i=1}^{n+1}\sum_{j=1,i\neq j}^{n+1}\left\langle x_i\, x_j\right\rangle^m = 2\sum_{i=1,i\neq k}^{n+1}\left\langle x_k\, x_i\right\rangle^m + \sum_{i=1,i\neq k}^{n+1}\sum_{\substack{j=1,j\neq k\\ i\neq j}}^{n+1}\left\langle x_i\, x_j\right\rangle^m \qquad k = 1, 2, \ldots, n+1;\ m \in \aleph.$$

Dies sieht man so

$$\sum_{i=1}^{n+1}\sum_{j=1,i\neq j}^{n+1}\left\langle x_i\, x_j\right\rangle^m = \sum_{i=1}^{n}\sum_{j=1,i\neq j}^{n}\left\langle x_i\, x_j\right\rangle^m + \sum_{i=1}^{n}\left\langle x_i\, x_{n+1}\right\rangle^m + \sum_{j=1}^{n}\left\langle x_{n+1}\, x_j\right\rangle^m$$

$$= 2\sum_{i=1,i\neq k}^{n}\left\langle x_k\, x_i\right\rangle^m + \sum_{i=1,i\neq k}^{n}\sum_{\substack{j=1,j\neq k\\ i\neq j}}^{n}\left\langle x_i\, x_j\right\rangle^m + 2\sum_{i=1}^{n}\left\langle x_i\, x_{n+1}\right\rangle^m$$

und weiterhin folgt

$$= 2 \sum_{i=1,i\neq k}^{n} \left\langle x_k\, x_i \right\rangle^m + \sum_{i=1,i\neq k}^{n} \sum_{\substack{j=1,j\neq k \\ i\neq j}}^{n} \left\langle x_i\, x_j \right\rangle^m + 2 \left\langle x_k\, x_{n+1} \right\rangle_{|n+1\neq k}^m + 2 \sum_{i=1,i\neq k}^{n} \left\langle x_i\, x_{n+1} \right\rangle^m$$

$$= 2 \sum_{i=1,i\neq k}^{n+1} \left\langle x_k\, x_i \right\rangle^m + \sum_{i=1,i\neq k}^{n} \sum_{\substack{j=1,j\neq k \\ i\neq j}}^{n} \left\langle x_i\, x_j \right\rangle^m + 2 \sum_{i=1,i\neq k}^{n} \left\langle x_i\, x_{n+1} \right\rangle^m$$

$$= 2 \sum_{i=1,i\neq k}^{n+1} \left\langle x_k\, x_i \right\rangle^m + \sum_{i=1,i\neq k}^{n} \sum_{\substack{j=1,j\neq k \\ i\neq j}}^{n} \left\langle x_i\, x_j \right\rangle^m + \sum_{i=1,i\neq k}^{n} \left\langle x_i\, x_{n+1} \right\rangle^m + \sum_{j=1,j\neq k}^{n} \left\langle x_{n+1}\, x_j \right\rangle^m$$

$$= 2 \sum_{i=1,i\neq k}^{n+1} \left\langle x_k\, x_i \right\rangle^m + \sum_{i=1,i\neq k}^{n} \sum_{\substack{j=1,j\neq k \\ i\neq j}}^{n} \left\langle x_i\, x_j \right\rangle^m + \sum_{i=1,i\neq k}^{n} \sum_{\substack{j=n+1,j\neq k \\ i\neq j}}^{n+1} \left\langle x_i\, x_j \right\rangle^m + \sum_{\substack{j=1,j\neq k \\ i\neq j}}^{n+1} \sum_{i=n+1,i\neq k}^{n+1} \left\langle x_i\, x_j \right\rangle^m$$

$$= 2 \sum_{i=1,i\neq k}^{n+1} \left\langle x_k\, x_i \right\rangle^m + \sum_{i=1,i\neq k}^{n} \sum_{\substack{j=1,j\neq k \\ i\neq j}}^{n+1} \left\langle x_i\, x_j \right\rangle^m + \sum_{\substack{j=n+1,j\neq k \\ i\neq j}}^{n+1} \sum_{i=n+1,i\neq k}^{n+1} \left\langle x_i\, x_j \right\rangle^m$$

$$= 2 \sum_{i=1,i\neq k}^{n+1} \left\langle x_k\, x_i \right\rangle^m + \sum_{\substack{j=1,j\neq k \\ i\neq j}}^{n+1} \sum_{i=1,i\neq k}^{n} \left\langle x_i\, x_j \right\rangle^m + \sum_{\substack{j=1,j\neq k \\ i\neq j}}^{n+1} \sum_{i=n+1,i\neq k}^{n+1} \left\langle x_i\, x_j \right\rangle^m$$

$$= 2 \sum_{i=1,i\neq k}^{n+1} \left\langle x_k\, x_i \right\rangle^m + \sum_{\substack{j=1,j\neq k \\ i\neq j}}^{n+1} \sum_{i=1,i\neq k}^{n} \left\langle x_i\, x_j \right\rangle^m + \sum_{i=n+1,i\neq k}^{n+1} \left\langle x_i\, x_j \right\rangle^m$$

$$= 2 \sum_{i=1,i\neq k}^{n+1} \left\langle x_k\, x_i \right\rangle^m + \sum_{\substack{j=1,j\neq k \\ i\neq j}}^{n+1} \sum_{i=1,i\neq k}^{n+1} \left\langle x_i\, x_j \right\rangle^m$$

$$= 2 \sum_{i=1,i\neq k}^{n+1} \left\langle x_k\, x_i \right\rangle^m + \sum_{i=1,i\neq k}^{n+1} \sum_{\substack{j=1,j\neq k \\ i\neq j}}^{n+1} \left\langle x_i\, x_j \right\rangle^m \qquad k = 1, 2, \ldots, n+1;\ m \in \aleph.$$

q.e.d.

Satz 3-1: Es sei durch R eine Kostenfunktion

$$R\left(w^1, w^2, \ldots, w^m\right) = \frac{1}{4}\sum_{i=1}^{m}\sum_{j=1, j\neq i}^{m}\left\langle y_i\, y_j\right\rangle^2 + \frac{1}{4}\sum_{i=1}^{m}\left(\left\langle y_i^2\right\rangle\right)^2$$

gegeben. Für den Gradienten von R erhält man dann für k = 1, 2, …, m

$$\nabla_{w^k} R\left(w^1, w^2, \ldots, w^m\right) = \sum_{i=1, i\neq k}^{m}\left\langle y_k\, y_i\right\rangle\left\langle x\, y_i\right\rangle - P\left\langle x\, y_k\right\rangle + \left\langle y_k^2\right\rangle\left\langle x\, y_k\right\rangle.$$

Beweis 3-1: Zunächst wird mit Hilfe von Korollar 3-1 die Doppelsumme im ersten Teilausdruck von R umgeformt. Dann läßt R darstellen als

$$R\left(w^1, w^2, \ldots, w^m\right) = \frac{1}{4}\sum_{i=1, i\neq k}^{m}\left\langle y_k\, y_i\right\rangle^2 + \sum_{i=1, i\neq k}^{m}\sum_{\substack{j=1, j\neq k,\\ i\neq j}}^{m}\left\langle y_i\, y_j\right\rangle^2 + \frac{1}{4}\sum_{i=1}^{m}\left(\left\langle y_i^2\right\rangle\right)^2$$

und für den Gradienten von R folgt

$$\nabla_{w^k} R\left(w^1, w^2, \ldots, w^m\right) = \frac{1}{2}\nabla_{w^k}\sum_{i=1, i\neq k}^{m}\left\langle y_k\, y_i\right\rangle^2 + \underbrace{\frac{1}{4}\nabla_{w^k}\sum_{i=1, i\neq k}^{m}\sum_{\substack{j=1, j\neq k,\\ i\neq j}}^{m}\left\langle y_i\, y_j\right\rangle^2}_{0}$$

$$+ \frac{1}{4}\nabla_{w^k}\left(\left\langle y_k^2\right\rangle\right)^2 + \underbrace{\frac{1}{4}\nabla_{w^k}\sum_{i=1, i\neq k}^{m}\left(\left\langle y_i^2\right\rangle\right)^2}_{0}$$

$$= \frac{1}{2}\sum_{i=1, i\neq k}^{m}\nabla_{w^k}\left\langle y_k\, y_i\right\rangle^2 + \frac{1}{4}\nabla_{w^k}\left(\left\langle y_k^2\right\rangle\right)^2$$

$$= \sum_{i=1, i\neq k}^{m}\left\langle y_k\, y_i\right\rangle\nabla_{w^k}\left\langle y_i\right\rangle + \frac{1}{2}\left\langle y_k^2\right\rangle\nabla_{w^k}\left\langle y_k\right\rangle$$

$$= \sum_{i=1, i\neq k}^{m}\left\langle y_k\, y_i\right\rangle C_{XX}\, w^i + \frac{1}{2}\left\langle y_k^2\right\rangle C_{XX}\, w^k$$

$$= \sum_{i=1, i\neq k}^{m}\left\langle y_k\, y_i\right\rangle\left\langle x\, y_i\right\rangle - P\, C_{XX}\, w^k + \left\langle y_k^2\right\rangle C_{XX}\, w^k$$

$$= \sum_{i=1, i\neq k}^{m}\left\langle y_k\, y_i\right\rangle\left\langle x\, y_i\right\rangle - P\left\langle x\, y_k\right\rangle + \left\langle y_k^2\right\rangle\left\langle x\, y_k\right\rangle \qquad k = 1, 2, \ldots, m.$$

q.e.d.

Satz 3-2: Es sei der Gradientenabstieg

$$w^k(t+1) = w^k(t) - \gamma \frac{\partial R(w^1, w^2, \ldots, w^m)}{\partial w^k} \qquad k = 1, 2, \ldots, m$$

und die Kostenfunktion R

$$R(w^1, w^2, \ldots, w^m) = \frac{1}{4}\sum_{i=1}^{m}\sum_{j=1, j\neq i}^{m}\left\langle y_i\, y_j\right\rangle^2 + \frac{1}{4}\sum_{i=1}^{m}\left(1-\left\langle y_i^2\right\rangle\right)^2$$

gegeben. Wird bei dem Gradientenabstieg eine sequentielle Gewichtskorrektur durchgeführt, so daß in jedem Zeitschritt t ein einzelner Gewichtsvektor w^k für k = 1, 2, …, m adaptiert wird, dann bewirkt die Anwendung der obigen Iterationsgleichung eine Konvergenz der Gewichtsvektoren w^1, w^2, …, w^m derart, daß das globale Minimum 0 der Kostenfunktion R gefunden wird.

Beweis 3-2: Wenn man zeigen kann, daß die Kostenfunktion R ein globales Minimum hat und bei sequentieller Anwendung des Gradientenverfahrens in jedem Zeitschritt t monoton abnimmt, dann besitzt sie die Eigenschaften einer Ljapunov-Funktion. Nach [BRO93] wäre dann die Konvergenz der Kostenfunktion zu diesem globalen Minimum gesichert.

Für den zeitlichen Verlauf der Kostenfunktion gilt

$$\frac{dR}{dt} = \sum_{i=1}^{m}\frac{\partial R}{\partial w^i}\frac{\partial w^i}{\partial t}$$

und mit dem Gradientenverfahren gilt

$$\frac{\partial w^k}{\partial t} = -\gamma\frac{\partial R}{\partial w^k} \qquad k = 1, 2, \ldots, m.$$

Das Einsetzen der zweiten Gleichung in die erste Gleichung liefert

$$\frac{dR}{dt} = \sum_{i=1}^{m}\frac{\partial R}{\partial w^i}\left(-\gamma\frac{\partial R}{\partial w^i}\right) = -\gamma\left(\sum_{i=1}^{m}\frac{\partial R}{\partial w^i}\right)^2 \leq 0.$$

Somit ist gezeigt, daß die Kostenfunktion R bei sequentieller Anwendung des Gradientenverfahrens in jedem Zeitschritt t monoton abnimmt. Da R auch ein globales Minimum bei 0 besitzt (siehe Seite 70), sind alle Bedingungen für eine Ljapunov-Funktion erfüllt. Damit ist die Konvergenz der Kostenfunktion R zu ihrem globalen Minimum 0 bewiesen.

q.e.d.

Satz 3-3: Es sei durch R eine Kostenfunktion

$$R(w^1, w^2, \ldots, w^m) = \frac{1}{4}\sum_{i=1}^{m}\sum_{j=1, j\neq i}^{m}\langle y_i\, y_j\rangle^2 + \frac{1}{4}\sum_{i=1}^{m}\left(P - \langle y_i^2\rangle\right)^2$$

gegeben. Dann erhält man als Lösung für das Gleichungssystem

$$\begin{aligned} \nabla_{w^1} R(w^1, w^2, \ldots, w^m) &= 0 \\ \nabla_{w^2} R(w^1, w^2, \ldots, w^m) &= 0 \\ &\vdots \\ \nabla_{w^m} R(w^1, w^2, \ldots, w^m) &= 0 \end{aligned} \qquad m \leq n$$

mit den Gradienten

$$\nabla_{w^k} R(w^1, w^2, \ldots, w^m) =$$

$$\sum_{i=1, i\neq k}^{m}\left(w^{iT} C_{XX}\, w^i\right) C_{XX}\, w^i - P\, C_{XX}\, w^k + \left(w^{kT} C_{XX}\, w^k\right) C_{XX}\, w^k$$

die Gewichtsvektoren

$$w^i = \sum_{r=1}^{n} a_{ri}\, e^r \qquad i = 1, 2, \ldots, m$$

mit den Bedingungen für deren Koordinaten

$$\sum_{r=1}^{n} a_{ri}^2 = P \qquad i = 1, 2, \ldots, m$$

$$\sum_{r=1}^{n} a_{ri}\, a_{rj} = 0 \qquad i, j = 1, 2, \ldots, m;\ i \neq j$$

bezüglich der orthogonalen Eigenvektorbasis $\{e^1, e^2, \ldots, e^n\}$ zu verschiedenen Eigenwerten $\{\lambda_1, \lambda_2, \ldots, \lambda_n\}$. Bei der Eigenvektorbasis handle es sich um die Eigenvektoren der reellwertigen symmetrischen Korrelationsmatrix C_{XX}, wobei für die Norm jedes Eigenvektors die Bedingung gelte

$$\left|e^r\right|^2 = 1/\lambda_r \qquad r = 1, 2, \ldots, n.$$

Beweis 3-3: Es wird im folgenden gezeigt, daß durch das Einsetzen der Formel

$$w^i = \sum_{r=1}^{n} a_{ri}\, e^r \qquad i = 1, 2, \ldots, m$$

in den Gradienten

$$\nabla_{w^k} R\left(w^1, w^2, \ldots, w^m\right) \qquad k = 1, 2, \ldots, m$$

das oben beschriebene Gleichungssystem gelöst wird. Unter Berücksichtigung der angegebenen Bedingungen für die Koordinaten folgt dann

$$\nabla_{w^k} R\left(w^1, w^2, \ldots, w^m\right) = 0 \qquad k = 1, 2, \ldots, m.$$

Durch das Einsetzen der Formel wird der Gradient dann umgeformt in

$$\begin{aligned}
&\nabla_{w^k} R\left(w^1, w^2, \ldots, w^m\right) \\
&= \sum_{i=1, i\neq k}^{m} \left(w^{iT} C_{XX} w^i\right)\left(P C_{XX} w^k + w^{kT} C_{XX} w^k\right) \\
&= \sum_{i=1, i\neq k}^{m} \left(\sum_{r=1}^{n} a_{rk} e^{rT}\right) C_{XX} \left(\sum_{s=1}^{n} a_{si} e^s\right) \left(P C_{XX} \sum_{s=1}^{n} a_{si} e^s\right) \\
&\quad - P C_{XX} \left(\sum_{r=1}^{n} a_{rk} e^r\right) \left(\sum_{q=1}^{n} a_{qk} e^{qT}\right) C_{XX} \left(\sum_{r=1}^{n} a_{rk} e^r\right) C_{XX} \left(\sum_{r=1}^{n} a_{rk} e^r\right) \\
&= \sum_{i=1, i\neq k}^{m} \left(\sum_{r=1}^{n} a_{rk} e^{rT}\right) C_{XX} \left(\sum_{s=1}^{n} a_{si} e^s\right) \left(P C_{XX} \sum_{s=1}^{n} a_{si} e^s\right) \\
&\quad - P \left(\sum_{r=1}^{n} a_{rk} C_{XX} e^r\right) \left(\sum_{q=1}^{n} a_{qk} e^{qT}\right) \left(\sum_{r=1}^{n} a_{rk} C_{XX} e^r\right) \left(\sum_{r=1}^{n} a_{rk} C_{XX} e^r\right) \\
&= \sum_{i=1, i\neq k}^{m} \left(\sum_{r=1}^{n} a_{rk} e^{rT}\right) C_{XX} \left(\sum_{s=1}^{n} a_{si} \lambda_s e^s\right) \left(P C_{XX} \sum_{s=1}^{n} a_{si} \lambda_s e^s\right) \\
&\quad - P \left(\sum_{r=1}^{n} a_{rk} \lambda_r e^r\right) \left(\sum_{q=1}^{n} a_{qk} e^{qT}\right) \left(\sum_{r=1}^{n} a_{rk} \lambda_r e^r\right) \left(\sum_{r=1}^{n} a_{rk} \lambda_r e^r\right) \\
&= \sum_{i=1, i\neq k}^{m} \left(\sum_{r=1}^{n} a_{rk} e^{rT}\right) \sum_{r=1}^{n} \sum_{s=1}^{n} a_{rk} a_{si} \lambda_s \underbrace{e^{rT} e^r}_{=0\ r\neq s} \left(P \sum_{s=1}^{n} a_{si} \lambda_s e^s\right) \\
&\quad - P \left(\sum_{r=1}^{n} a_{rk} \lambda_r e^r\right) \left(\sum_{q=1}^{n} \sum_{r=1}^{n} a_{qk} a_{rk} \lambda_r \underbrace{e^{qT} e^r}_{=0\ q\neq r}\right) \left(\sum_{r=1}^{n} a_{rk} \lambda_r e^r\right)
\end{aligned}$$

und weiterhin folgt

$$= \sum_{i=1,i\neq k}^{m} \sum_{r=1}^{n} a_{rk}\, a_{ri}\, \lambda_r \left| e^r \right|^2_{=1/\lambda_r} \sum_{s=1}^{n} a_{si}\, \lambda_s\, e^s$$

$$- P \sum_{r=1}^{n} a_{rk}\, \lambda_r\, e^r \sum_{q=1}^{n} a_{qk}^2\, \lambda_q \left| e^q \right|^2_{=1/\lambda_q} \sum_{r=1}^{n} a_{rk}\, \lambda_r\, e^r$$

$$= \sum_{i=1,i\neq k}^{m} \underbrace{\sum_{r=1}^{n} a_{rk}\, a_{ri}}_{=0\ i\neq k} \sum_{s=1}^{n} a_{si}\, \lambda_s\, e^s - P \sum_{r=1}^{n} a_{rk}\, \lambda_r\, e^r \underbrace{\sum_{q=1}^{n} a_{qk}^2}_{=P} \sum_{r=1}^{n} a_{rk}\, \lambda_r\, e^r$$

$$= 0 \qquad k = 1, 2, \ldots, m. \qquad \text{q.e.d.}$$

Satz 3-4: Es sei y = W x eine lineare Transformation mit einer Matrix W derart, daß die Transformation eine Datenorthonormalisierung durchführt. Jeder Eingabevektor $x \in X$ sei durch die inverse Transformation

$$x = \sum_{i=1}^{n} y_i \left(\lambda_i^{-1} C_{XX} w^i \right)$$

gegeben. Zu jedem Eingabevektor x existiere ein approximativer Vektor x', der durch eine approximative inverse Transformation

$$x' = \sum_{i=1}^{m} y_i \left(\lambda_i^{-1} C_{XX} w^i \right) \qquad m \leq n$$

bestimmt wird, indem alle Summenterme für $m+1 \leq i \leq n$ vernachlässigt werden. Der hier definierte mittlere quadratische Sampling-Fehler

$$\varepsilon_s^2 = \left\langle \left| x - x' \right|^2 \right\rangle$$

ist dann gegeben durch

$$\varepsilon_s^2 = \left(\lambda_i^{-1} C_{XX} w^i \right) \sum_{r=1}^{n} \lambda_r \sum_{i=m+1}^{n} a_{ri}^2 .$$

Beweis 3-4: Nach dem Einsetzen des Eingabevektors x und des approximativen Vektors x' in die Formel

$$\varepsilon_s^2 = \left\langle \left| x - x' \right|^2 \right\rangle$$

ergibt sich dann

$$\varepsilon_s^2 = \left\langle \left| \sum_{i=1}^{n} y_i \, C_{XX} w^i - \sum_{i=1}^{m} y_i \, C_{XX} w^i \right|^2 \right\rangle$$

$$= \left\langle \left| \sum_{i=m+1}^{n} y_i \, C_{XX} w^i \right|^2 \right\rangle$$

$$= \left\langle \sum_{i=m+1}^{n} y_i \, C_{XX} w^i , \sum_{j=m+1}^{n} y_j \, C_{XX} w^j \right\rangle$$

$$= \left\langle \sum_{i=m+1}^{n} \sum_{j=m+1}^{n} y_i \, y_j \, C_{XX} w^i , C_{XX} w^j \right\rangle$$

$$= \sum_{i=m+1}^{n} \sum_{j=m+1}^{n} \underbrace{\langle y_i \, y_j \rangle}_{=0 \; i \neq j} C_{XX} w^i , C_{XX} w^j$$

$$= \sum_{i=m+1}^{n} \underbrace{\langle y_i^2 \rangle}_{=1/P} C_{XX} w^i , C_{XX} w^i$$

$$= \sum_{i=m+1}^{n} C_{XX} w^i , C_{XX} w^i = \sum_{i=m+1}^{n} C_{XX} \sum_{r=1}^{n} a_{ri} e^r , C_{XX} \sum_{s=1}^{n} a_{si} e^s$$

$$= \sum_{i=m+1}^{n} \sum_{r=1}^{n} a_{ri} \, C_{XX} e^r , \sum_{s=1}^{n} a_{si} \, C_{XX} e^s$$

$$= \sum_{i=m+1}^{n} \sum_{r=1}^{n} a_{ri} \, \lambda_r \, e^r , \sum_{s=1}^{n} a_{si} \, \lambda_s \, e^s$$

$$= \sum_{i=m+1}^{n} \sum_{r=1}^{n} \sum_{s=1}^{n} a_{ri} \, a_{si} \, \lambda_r \, \lambda_s \underbrace{e^r , e^s}_{=0 \; r \neq s}$$

$$= \sum_{i=m+1}^{n} \sum_{r=1}^{n} a_{ri}^2 \, \lambda_r^2 \underbrace{e^r , e^r}_{=1/\lambda_r} = \sum_{i=m+1}^{n} \sum_{r=1}^{n} a_{ri}^2 \, \lambda_r$$

$$= \sum_{r=1}^{n} \lambda_r \sum_{i=m+1}^{n} a_{ri}^2$$

q.e.d.

Satz 3-5: Es sei y = W x eine lineare Transformation mit einer Matrix W derart, daß die Transformation eine Datenorthonormalisierung durchführt. Weiterhin seien die Zeilen von W durch die Vektoren

$$w^{iT} = \sqrt{P}\, e^{iT} \qquad i = 1, 2, \ldots, n;\ P > 0$$

gegeben, wobei es sich bei $\{e^1, e^2, \ldots, e^n\}$ um die Eigenvektoren der Korrelationsmatrix C_{XX} zu verschiedenen Eigenwerten $\{\lambda_1, \lambda_2, \ldots, \lambda_n\}$ handelt. Die Eigenvektoren seien gemäß der Größe ihrer zugehörigen Eigenwerte absteigend geordnet und für deren Norm gelte die Bedingung

$$\left|e^i\right|^2 = \frac{1}{\lambda_i} \qquad i = 1, 2, \ldots, n.$$

Besitzt die Transformationsmatrix W die oben beschriebene Gestalt, dann wird der durch

$$\varepsilon_s^2 = \left\langle \left| x - x' \right|^2 \right\rangle$$

definierte mittlere quadratische Sampling-Fehler minimal und vereinfacht sich zu

$$\varepsilon_s^2 = \sum_{r=m+1}^{n} \lambda_r .$$

Beweis 3-5: Es sei W eine beliebige Transformationsmatrix, so daß mit einer linearen Transformation der Art y = W x eine Datenorthonormalisierung durchgeführt wird. Dann ist nach Satz 3-4 der mittlere quadratische Sampling-Fehler gegeben durch

$$\varepsilon_s^2 = \frac{1}{P} \sum_{r=1}^{m} \lambda_r \sum_{i=m+1}^{n} a_{ri}^2 .$$

Um diesen Fehler zu minimieren, werden die relativen Extrema seiner Funktion unter den Nebenbedingungen für die Koordinaten

$$\sum_{s=1}^{n} a_{si}^2 = P \qquad i = m+1, \ldots, n$$

mit Hilfe der Methode der Lagrange-Multiplikatoren untersucht. Hierzu wird mit einer Menge von Funktionen

$$g_i\left(a_{1i}, \ldots, a_{ni}\right) = \sum_{s=1}^{n} a_{si}^2 - P \qquad i = m+1, \ldots, n$$

sowie der Funktion

$$f(a_{1,m+1}, \ldots, a_{nn}) = \sum_{r=1}^{m} \alpha_r \sum_{i=m+1}^{n} a_{ri}^2$$

und vorläufig unbestimmten Konstanten $\mu_{m+1}, \ldots, \mu_n$ die Lagrange-Funktion L mit den Multiplikatoren $\mu_{m+1}, \ldots, \mu_n$

$$L(a_{1,m+1}, \ldots, a_{nn}, \mu_{m+1}, \ldots, \mu_n) = f(a_{1,m+1}, \ldots, a_{nn}) + \sum_{i=m+1}^{n} \mu_i g_i(a_{1i}, \ldots, a_{ni})$$

gebildet. Für die zu bestimmenden Größen

$$a_{ri} \qquad r = 1, 2, \ldots, n;\ i = m+1, \ldots, n$$

und

$$\mu_i \qquad i = m+1, \ldots, n$$

wird das Gleichungssystem

$$\begin{aligned} &g_i(a_{1i}, \ldots, a_{ni}) = 0 && i = m+1, \ldots, n \\ &\frac{\partial L}{\partial a_{jk}}(a_{1,m+1}, \ldots, a_{nn}, \mu_{m+1}, \ldots, \mu_n) = 0 && j = 1, 2, \ldots, n;\ k = m+1, \ldots n \end{aligned}$$

untersucht. Die Gleichungen $g_i(a_{1i}, \ldots, a_{ni}) = 0$ für i = m+1, …, n sind genau dann erfüllt, wenn die geforderten Nebenbedingungen gelten. Die verbleibenden Gleichungen werden im folgenden betrachtet.

$$\frac{\partial L}{\partial a_{jk}}(a_{1,m+1}, \ldots, a_{nn}, \mu_{m+1}, \ldots, \mu_n)$$

$$= \frac{\partial}{\partial a_{jk}} \left(\sum_{r=1}^{m} \alpha_r \sum_{i=m+1}^{n} a_{ri}^2 \sum_{i=m+1}^{n} \mu_i \sum_{s=1}^{n} a_{si}^2 - P \right)$$

$$= \frac{\partial}{\partial a_{jk}} \left(\sum_{r=1}^{m} \alpha_r \sum_{i=m+1}^{n} a_{ri}^2 \frac{\partial}{\partial a_{jk}} \sum_{i=m+1}^{n} \mu_i \sum_{s=1}^{n} a_{si}^2 - P \right)$$

$$= \sum_{r=1}^{m} \alpha_r \underbrace{\frac{\partial}{\partial a_{jk}} \sum_{i=m+1}^{n} a_{ri}^2}_{=0\ a_{ri} \neq a_{jk}} \sum_{i=m+1}^{n} \mu_i \underbrace{\frac{\partial}{\partial a_{jk}} \sum_{s=1}^{n} a_{si}^2}_{=0\ a_{si} \neq a_{jk}} - P$$

$$= \sum_{r=1}^{m} \lambda_j a_{jk} + \mu_k 2 a_{jk} = 2 a_{jk} \sum_{r=1}^{m} \lambda_j + \mu_k \qquad j = 1, 2, \ldots, n;\ k = m+1, \ldots, n$$

Unter Berücksichtigung des obigen Gleichungssystems folgt

$$2\,a_{jk} \cdot (\lambda_j + \mu_k) = 0$$

$$\Leftrightarrow a_{jk} \cdot (\lambda_j + \mu_k) = 0 \qquad j = 1, 2, \ldots, n;\ k = m+1, \ldots, n$$

Was läßt nun aus der obigen Gleichung folgern ? Zunächst einmal ist es einfach zu sehen, daß die Gleichung erfüllt ist, wenn die Bedingung

$$a_{jk} = 0 \quad \vee \quad (\lambda_j + \mu_k) = 0$$

gilt. Unter der Annahme, daß $a_{jk} \neq 0$ folgt dann

$$\mu_k = -\lambda_j .$$

Betrachtet man jetzt den Fall

$$a_{rk} \neq 0 \quad \wedge \quad a_{sk} \neq 0 \qquad r, s = 1, 2, \ldots, n;\ r \neq s$$

so ergibt sich ein Widerspruch

$$a_{rk} \neq 0 \quad \Rightarrow \quad \mu_k = -\lambda_r$$

$$a_{sk} \neq 0 \quad \Rightarrow \quad \mu_k = -\lambda_s$$

$$\Rightarrow \lambda_r = \lambda_s \qquad r, s = 1, 2, \ldots, n;\ r \neq s,$$

da nach Voraussetzung die Eigenvektoren der Korrelationsmatrix C_{XX} zu verschiedenen Eigenwerten betrachtet werden. Aus diesem Widerspruch läßt sich folgern, daß aus

$$a_{jk} \neq 0 \qquad j = 1, 2, \ldots, n$$

stets folgt

$$a_{rk} = 0 \qquad r = 1, 2, \ldots, n;\ r \neq j.$$

Aus den bisherigen Erkenntnissen läßt sich schließen, daß die Funktion des mittleren quadratischen Sampling-Fehlers genau dann relative Extrema besitzt, wenn die Komponenten a_{jk} jedes Koordinatenvektors

$$a^k = a_{1k}, a_{2k}, \ldots, a_{nk} \qquad k = m+1, \ldots, n$$

bis auf ein einziges Element den Wert 0 annehmen. Unter Berücksichtigung der Nebenbedingungen für die Koordinaten muß dieses Element den Wert $\sqrt{P}$ besitzen.

Da für die Normen der Koordinatenvektoren gilt

$$\left|a^k\right|^2 = P \qquad\qquad k = m+1, \ldots, n$$

und diese zusätzlich zueinander orthogonal sind (siehe Satz 3-3), nimmt die Funktion des mittleren quadratischen Sampling-Fehlers ihr Minimum an, wenn für beliebige $m \leq n$ die $n \times n$ Matrix A mit den Koordinatenvektoren in den Spalten die folgende Gestalt besitzt

$$A = \sqrt{P}\, I.$$

Denn nur in diesem Fall läuft für beliebige $m \leq n$ die Summe der Funktion nur noch über die kleinsten Eigenwerte $\lambda_{m+1}, \ldots, \lambda_n$ und vereinfacht sich zu

$$\varepsilon_s^2 = \frac{1}{P} \sum_{r=m+1}^{n} \lambda_r \underbrace{\sum_{i=m+1}^{n} a_{ri}^2}_{=P}$$

$$= \sum_{r=m+1}^{n} \lambda_r .$$

Da der i-te Koordinatenvektor a^i die Koordinaten des i-ten Zeilenvektors w^{iT} von W bezüglich der Eigenvektorbasis $\{e^1, e^2, \ldots, e^n\}$ beinhaltet, folgt aus der oben beschriebenen Gestalt der Matrix A unmittelbar, daß jeder Vektor der Transformationsmatrix W gegeben ist durch

$$w^{iT} = \sqrt{P}\, e^{iT} \qquad\qquad i = 1, 2, \ldots, n.$$

Damit ist der Satz bewiesen. q.e.d.

Anhang 4: Klassendeklarationen der Bibliothek MAC

```
/*=========================================================================*/
/* Klasse    | CMathArrayException                                         */
/* Inhalt    | Klasse zur Beschreibung einer Ausnahme, die auftreten kann, */
/*           | wenn mathematische Matrizen oder Vektoren verwendet werden  */
/*-------------------------------------------------------------------------*/
/* Bemerkung | Grund der Ausnahme kann mit 'GetCause' ermittelt werden     */
/*=========================================================================*/

class CMathArrayException : public CException
{
   DECLARE_DYNAMIC(CMathArrayException);

   private:

   // Daten der Klasse
      UINT m_nCause;                          // Grund für die Ausnahme

   public:

   // Konstanten der Klasse
      enum                                    // Beschreibung aller Fehler
      {
         None,                                // Kein Fehler wird angezeigt
         OutOfRange,                          // Zugriff außerhalb Feld
         WrongDimension,                      // Feldgrößen nicht passend
         IllegalObject,                       // Ungültiger Objektzeiger
      };

   // Konstruktoren und Destruktoren
      CMathArrayException(UINT nCause = None);

   // Mitgliedsfunktionen
      UINT GetCause();

#ifdef _DEBUG
      virtual void Dump(CDumpContext& dc) const;
#endif // _DEBUG
}; // CMathArrayException
```

```
/*=========================================================================*/
/* Klasse    | CMathArray                                                  */
/* Inhalt    | Die grundlegende Basisklasse für mathematische Matrizen und */
/*           | Vektoren mit reellwertigen Komponenten                      */
/*-------------------------------------------------------------------------*/
/* Bemerkung | Keine                                                       */
/*=========================================================================*/

class CMathArray : public CObject
{
   DECLARE_SERIAL(CMathArray);

   protected:

   // Daten der Klasse
      UINT   m_nNrComponents,                // Anzahl der Zahlenelemente
             m_nWidth;                       // Feldbreite für Textausgabe
      double *m_pdComponents;                // Zeiger auf das Zahlenfeld

   public:

   // Konstruktoren und Destruktoren
      CMathArray();
      CMathArray(const CMathArray &array);
      CMathArray(UINT nNrComp);
      virtual ~CMathArray();

   // Mitgliedsfunktionen
      virtual void Serialize(CArchive &archive);
      void Fill(double dValue);
      double Min() const;
      double Max() const;
      void SetWidth(UINT nWidth);
      UINT GetWidth() const;
      void Init(UINT nNrComp);
      void Empty();

#ifdef _DEBUG
      virtual void AssertValid() const;
      virtual void Dump(CDumpContext &dc) const;
#endif // _DEBUG

   // Operatorfunktionen
      CMathArray &operator = (const CMathArray &array);
      CMathArray &operator += (const CMathArray &array);
      CMathArray &operator -= (const CMathArray &array);
      CMathArray &operator *= (double dScalar);
      CMathArray &operator /= (double dScalar);
      CMathArray operator + () const;
      CMathArray operator - () const;
      CMathArray operator + (const CMathArray &array) const;
      CMathArray operator - (const CMathArray &array) const;
      CMathArray operator * (double dScalar) const;
      CMathArray operator / (double dScalar) const;
      BOOL operator == (const CMathArray &array) const;
      BOOL operator != (const CMathArray &array) const;

   protected:

   // Mitgliedsfunktionen
      double &Item(UINT nIndex);
      double Item(UINT nIndex) const;
      UINT GetSize() const;
}; // CMathArray
```

```
/*============================================================================*/
/* Klasse    | CMatrix                                                        */
/* Inhalt    | Klasse für mathematische Matrizen                              */
/*----------------------------------------------------------------------------*/
/* Bemerkung | Keine                                                          */
/*============================================================================*/

class CMatrix : public CMathArray
{
   DECLARE_SERIAL(CMatrix);

   private:

   // Operatorfunktionen
      CMatrix &operator = (const CMathArray &array);

   protected:

   // Daten der Klasse
      UINT m_nNrLines,                        // Anzahl der Matrixzeilen
           m_nNrColumns;                      // Anzahl der Matrixspalten

   public:

   // Konstanten der Klasse
      enum                                    // Angabe Zeile oder Spalte
      {
         Line   = 1,                          // Zeilenbasierte Operation
         Column = 2                           // Spaltenbasierte Operation
      };

   // Konstruktoren und Destruktoren
      CMatrix();
      CMatrix(UINT nNrLines, UINT nNrColumns);
      CMatrix(const CMatrix &matrix);

   // Mitgliedsfunktionen
      virtual void Serialize(CArchive &archive);
      void Transponate();
      CMatrix GetTransponate() const;
      void Insert(UINT nIndex, UINT nPart);
      void Delete(UINT nIndex, UINT nPart);
      void Expand(UINT nPart);
      void Reduce(UINT nPart);
      void Init(UINT nNrLines, UINT nNrColumns);
      void Empty();
      void Swap(UINT nIndex1, UINT nIndex2, UINT nPart);
      UINT GetNrLines() const;
      UINT GetNrColumns() const;

#ifdef _DEBUG
      virtual void AssertValid() const;
      virtual void Dump(CDumpContext &dc) const;
#endif // _DEBUG

   // Operatorfunktionen
      CMatrix &operator = (const CMatrix &matrix);
      double &operator [] (const CIndex &index);
      double operator [] (const CIndex &index) const;
      CMatrix operator + () const;
      CMatrix operator - () const;
      CMatrix operator + (const CMatrix &matrix) const;
      CMatrix operator - (const CMatrix &matrix) const;
      CMatrix operator * (double dScalar) const;
      CMatrix operator / (double dScalar) const;
      CMatrix &operator *= (const CMatrix &matrix);
      CMatrix operator * (const CMatrix &matrix) const;
      BOOL operator == (const CMatrix &matrix) const;
      CMatrix &operator += (const CMatrix &matrix);
      CMatrix &operator -= (const CMatrix &matrix);
   // ...
```

```
   // ...
      CMatrix &operator *= (double dScalar);
      CMatrix &operator /= (double dScalar);
      BOOL operator != (const CMatrix &matrix) const;

   // Freunde
      friend ostream &operator << (ostream &stream, const CMatrix &matrix);
      friend istream &operator >> (istream &stream, CMatrix &matrix);

   protected:

   // Mitgliedsfunktionen
      double &Item(UINT nLine, UINT nColumn);
      double Item(UINT nLine, UINT nColumn) const;
}; // CMatrix


/*=========================================================================*/
/* Klasse    | CIndex                                                      */
/* Inhalt    | Klasse zur Darstellung eines Index für eine Matrix          */
/*-------------------------------------------------------------------------*/
/* Bemerkung | Binäre Zuweisung und binärer Kopier-Konstruktor erlaubt     */
/*=========================================================================*/

struct CIndex
{
   // Daten der Klasse
      UINT m_nLine,                          // Zeilennummer in Matrix
           m_nColumn;                        // Spaltennummer in Matrix

   // Konstruktoren und Destruktoren
      CIndex();
      CIndex(UINT nLine, UINT nColumn);

   // Operatorfunktionen
      CIndex &operator () (UINT nLine, UINT nColumn);
}; // CIndex


/*=========================================================================*/
/* Klasse    | CPartVector                                                 */
/* Inhalt    | Klasse zur Beschreibung eines Teils einer Matrix vom Daten- */
/*           | typ 'CMatrix', der sich als Vektor darstellen läßt          */
/*-------------------------------------------------------------------------*/
/* Bemerkung | Binäre Zuweisung und binärer Kopier-Konstruktor erlaubt     */
/*=========================================================================*/

class CPartVector : public CObject
{
   DECLARE_DYNAMIC(CPartVector);

   public:

   // Daten der Klasse
      UINT    m_nPart;                       // Festlegung des Matrixteils
      CVector *m_pVector;                    // Vektor ist Teil der Matrix

   // Konstruktoren und Destruktoren
      CPartVector();
      CPartVector(CVector &vector, UINT nPart);

   // Mitgliedsfunktionen
#ifdef _DEBUG
      virtual void AssertValid() const;
      virtual void Dump(CDumpContext &dc) const;
#endif // _DEBUG
}; // CPartVector
```

```
/*============================================================================*/
/* Klasse    | CLine                                                          */
/* Inhalt    | Klasse zur Beschreibung eines Vektors, welcher als Zeile in */
/*           | einer Matrix vom Typ 'CMatrix' betrachtet werden soll       */
/*----------------------------------------------------------------------------*/
/* Bemerkung | Binäre Zuweisung und binärer Kopier-Konstruktor erlaubt     */
/*============================================================================*/

class CLine : public CPartVector
{
   DECLARE_DYNAMIC(CLine);

   public:

   // Konstruktoren und Destruktoren
      CLine();
      CLine(CVector &vector, UINT nLine);

   // Operatorfunktionen
      CLine &operator () (CVector &vector, UINT nLine);
}; // CLine


/*============================================================================*/
/* Klasse    | CColumn                                                        */
/* Inhalt    | Klasse zur Beschreibung eines Vektors, der als Spalte einer */
/*           | Matrix vom Typ 'CMatrix' betrachtet werden soll             */
/*----------------------------------------------------------------------------*/
/* Bemerkung | Binäre Zuweisung und binärer Kopier-Konstruktor erlaubt     */
/*============================================================================*/

class CColumn : public CPartVector
{
   DECLARE_DYNAMIC(CColumn);

   // Konstruktoren und Destruktoren
      CColumn();
      CColumn(CVector &vector, UINT nColumn);

   // Operatorfunktionen
      CColumn &operator () (CVector &vector, UINT nColumn);
}; // CColumn


/*============================================================================*/
/* Klasse    | CDiagonal                                                      */
/* Inhalt    | Klasse zur Beschreibung eines Vektors, welcher als Diagona- */
/*           | le oder Gegen-Diagonale einer quadratischen Matrix vom Typ  */
/*           | 'CSquareMatrix' betrachtet werden soll                      */
/*----------------------------------------------------------------------------*/
/* Bemerkung | Binäre Zuweisung und binärer Kopier-Konstruktor erlaubt     */
/*============================================================================*/

class CDiagonal : public CPartVector
{
   DECLARE_DYNAMIC(CDiagonal);

   public:

   // Konstruktoren und Destruktoren
      CDiagonal();
      CDiagonal(CVector &vector, BOOL bContra = FALSE);

   // Operatorfunktionen
      CDiagonal &operator () (CVector &vector, BOOL bContra = FALSE);
}; // CDiagonal
```

```
/*============================================================================*/
/* Klasse    | CVector                                                        */
/* Inhalt    | Klasse für mathematische Vektoren                              */
/*----------------------------------------------------------------------------*/
/* Bemerkung | Keine                                                          */
/*============================================================================*/

class CVector : public CMathArray
{
   DECLARE_SERIAL(CVector);

   private:

   // Konstruktoren und Destruktoren
      CVector(const CMathArray &array);

   public:

   // Konstruktoren und Destruktoren
      CVector();
      CVector(UINT nNrComp);
      CVector(const CVector &vector);

   // Mitgliedsfunktionen
      void Insert(UINT nIndex);
      void Delete(UINT nIndex);
      void Expand();
      void Reduce();
      void Normalize(double dLength = 1.0);
      void Swap(UINT nIndex1, UINT nIndex2);
      UINT GetNrComponents() const;
      virtual double Norm() const;

   // Operatorfunktionen
      double operator * (const CVector &vector) const;
      double &operator [] (UINT nIndex);
      double operator [] (UINT nIndex) const;
      CVector operator + () const;
      CVector operator - () const;
      CVector &operator = (const CVector &vector);
      CVector &operator += (const CVector &vector);
      CVector &operator -= (const CVector &vector);
      CVector &operator *= (double dScalar);
      CVector &operator /= (double dScalar);
      CVector operator + (const CVector &vector) const;
      CVector operator - (const CVector &vector) const;
      CVector operator * (double dScalar) const;
      CVector operator / (double dScalar) const;

   // Freunde
      friend ostream &operator << (ostream &stream, const CVector &vector);
      friend istream &operator >> (istream &stream, CVector &vector);
}; // CVector
```

```
/*============================================================================*/
/* Klasse    | CSquareMatrix                                                  */
/* Inhalt    | Klasse für mathematische quadratische Matrizen                 */
/*----------------------------------------------------------------------------*/
/* Bemerkung | Keine                                                          */
/*============================================================================*/

class CSquareMatrix : public CMatrix
{
   DECLARE_SERIAL(CSquareMatrix);

   public:

   // Konstruktoren und Destruktoren
      CSquareMatrix();
      CSquareMatrix(UINT nDim);
      CSquareMatrix(const CSquareMatrix &matrix);

   // Mitgliedsfunktionen
      BOOL IsSymmetrical() const;
      void FillDiag(double dScalar, BOOL bContra = FALSE);
      void Init(UINT nDim);
      void Expand();
      void Reduce();
      void Insert(UINT nIndex);
      void Delete(UINT nIndex);

   // Operatorfunktionen
      CSquareMatrix &operator = (const CMatrix &matrix);
      CSquareMatrix &operator *= (const CMatrix &matrix);
      CSquareMatrix &operator << (const CDiagonal &diag);
      const CSquareMatrix &operator >> (CDiagonal &diag) const;

   // Freunde
      friend class CEigenMatrix;
}; // CSquareMatrix


/*============================================================================*/
/* Klasse    | CStandardMatrix                                                */
/* Inhalt    | Klasse für mathematische quadratische Matrizen, welche als     */
/*           | Einheitsmatrix initialisiert werden                            */
/*----------------------------------------------------------------------------*/
/* Bemerkung | Keine                                                          */
/*============================================================================*/

class CStandardMatrix : public CSquareMatrix
{
   DECLARE_SERIAL(CStandardMatrix);

   public:

   // Konstruktoren und Destruktoren
      CStandardMatrix();
      CStandardMatrix(UINT nDim);
      CStandardMatrix(const CStandardMatrix &matrix);

   // Mitgliedsfunktionen
      void Insert(UINT nIndex, BOOL bUpdateDiag = TRUE);
      void Expand(BOOL bUpdateDiag = TRUE);
      void Init(UINT nDim);

   // Operatorfunktionen
      CStandardMatrix &operator = (const CMatrix &matrix);
}; // CStandardMatrix
```

```
/*============================================================================*/
/* Klasse    | CDiagMatrix                                                    */
/* Inhalt    | Klasse für mathematische quadratische Matrizen, welche ihre */
/*           | Quadratsummennorm außerhalb der Hauptdiagonalen berechnen   */
/*           | können. Dieser Wert bildet ein Maß dafür, wie stark die Ma- */
/*           | trix von der Diagonalgestalt abweicht. Zusätzlich kann jede */
/*           | Matrix ihr betragsgrößtes Element außerhalb der Hauptdiago- */
/*           | nalen ermitteln.                                             */
/*----------------------------------------------------------------------------*/
/* Bemerkung | Keine                                                          */
/*============================================================================*/

class CDiagMatrix : public CSquareMatrix
{
   DECLARE_SERIAL(CDiagMatrix);

   public:

   // Konstruktoren und Destruktoren
      CDiagMatrix();
      CDiagMatrix(UINT nDim);
      CDiagMatrix(const CDiagMatrix &matrix);

   // Mitgliedsfunktionen
      double SquareSumNorm() const;
      void FindMaxItem(UINT &nLine, UINT &nColumn) const;

   // Operatorfunktionen
      CDiagMatrix &operator = (const CMatrix &matrix);

   // Freunde
      friend class CEigenMatrix;
}; // CDiagMatrix
```

```
/*============================================================================*/
/* Klasse    | CEigenMatrix                                                   */
/* Inhalt    | Klasse zur Beschreibung von symmetrischen Matrizen, welche     */
/*           | ihre Eigenwerte und Eigenvektoren berechnen können. Es wird    */
/*           | das klassische Jacobische Verfahren verwendet, nachzulesen     */
/*           | in Stummel/Hainer Numerische Mathematik, Teubner Verlag, ab    */
/*           | Seite 212ff.                                                   */
/*----------------------------------------------------------------------------*/
/* Bemerkung | Keine                                                          */
/*============================================================================*/

class CEigenMatrix : public CSquareMatrix
{
   DECLARE_SERIAL(CEigenMatrix);

   protected:

   // Daten der Klasse
      BOOL m_bContinue;                        // Berechnungen fortsetzen

   private:

   // Mitgliedsfunktionen
      double CalcAngle(const CDiagMatrix &A, UINT nSigma, UINT nTau) const;

      void CalcTransformation(CSquareMatrix &U,
                              UINT nSigma, UINT nTau, double dPhi) const;

   public:

   // Konstruktoren und Destruktoren
      CEigenMatrix();
      CEigenMatrix(UINT nDim);
      CEigenMatrix(const CEigenMatrix &matrix);

   // Mitgliedsfunktionen
      virtual void Serialize(CArchive &archive);
      void Abort();
      BOOL GetAbortFlag() const;
      void Init(UINT nDim);
      void Empty();

      virtual void Calculate(CVector &eigenvalues,
                             CSquareMatrix &eigenvectors,
                             double dAposteriori,
                             UINT nPeriod = (UINT)FALSE);

#ifdef _DEBUG
      virtual void Dump(CDumpContext &dc) const;
#endif // _DEBUG

   // Operatorfunktionen
      CEigenMatrix &operator = (const CEigenMatrix &matrix);
      CEigenMatrix &operator = (const CMatrix &matrix);
      BOOL operator == (const CEigenMatrix &matrix) const;
      BOOL operator != (const CEigenMatrix &matrix) const;

   protected:

   // Mitgliedsfunktionen
      virtual BOOL WakeUp(double dSqrSumNorm, UINT nStep);
}; // CEigenMatrix
```

```
/*============================================================================*/
/* Klasse    | CInputVector                                                   */
/* Inhalt    | Klasse zur Beschreibung eines Eingabevektors für ein neuro- */
/*           | nales Netz                                                     */
/*----------------------------------------------------------------------------*/
/* Bemerkung | Keine                                                          */
/*============================================================================*/

class CInputVector : public CVector
{
   DECLARE_SERIAL(CInputVector);

   private:

   // Daten der Klasse
      double m_dProb;                          // Wahrscheinlichkeit

   public:

   // Konstruktoren und Destruktoren
      CInputVector();
      CInputVector(const CInputVector &vector);
      CInputVector(UINT nNrComp, double dProb = 0.0);

   // Mitgliedsfunktionen
      virtual void Serialize(CArchive &archive);
      void Empty();
      double GetProb() const;
      void SetProb(double dProb);
      void Init(UINT nNrComp, double dProb = 0.0);

#ifdef _DEBUG
      virtual void Dump(CDumpContext &dc) const;
#endif // _DEBUG

   // Operatorfunktionen
      CInputVector &operator = (const CInputVector &vector);
      BOOL operator == (const CInputVector &vector) const;
      CInputVector &operator = (const CVector &vector);
      BOOL operator != (const CInputVector &vector) const;

   // Freunde
      friend ostream &operator << (ostream &stream, const CInputVector &vec);
      friend istream &operator >> (istream &stream, CInputVector &vec);
}; // CInputVector
```

```
/*============================================================================*/
/* Klasse    | CInputMatrix                                                   */
/* Inhalt    | Klasse zur Beschreibung einer Eingabemenge für ein neurona-    */
/*           | les Netz                                                       */
/*----------------------------------------------------------------------------*/
/* Bemerkung | Jede Zeile der Eingabematrix beschreibt einen Eingabevektor    */
/*           | für das neuronale Netz. Die Menge aller Matrixzeilen stellt    */
/*           | dann die gesamte Eingabemenge für das neuronale Netz dar.      */
/*============================================================================*/

class CInputMatrix : public CMatrix
{
   DECLARE_SERIAL(CInputMatrix);

   private:

   // Daten der Klasse
      CVector m_Probs;                        // Alle Wahrscheinlichkeiten

   public:

   // Konstruktoren und Destruktoren
      CInputMatrix();
      CInputMatrix(const CInputMatrix &matrix);
      CInputMatrix(UINT nNrLines, UINT nNrColumns, double dProb = 0.0);

   // Mitgliedsfunktionen
      virtual void Serialize(CArchive &archive);
      CSquareMatrix CalcCorrelation() const;
      CSquareMatrix CalcCovariance() const;
      void Insert(UINT nIndex, UINT nPart);
      void Delete(UINT nIndex, UINT nPart);
      void Expand(UINT nPart);
      void Reduce(UINT nPart);
      void Swap(UINT nIndex1, UINT nIndex2, UINT nPart);
      void Init(UINT nNrLines, UINT nNrColumns, double dProb = 0.0);
      void Empty();
      void SetProb(UINT nLine, double dProb);
      double GetProb(UINT nLine) const;
      void FillProbs(double dProb);

#ifdef _DEBUG
      virtual void AssertValid() const;
      virtual void Dump(CDumpContext &dc) const;
#endif // _DEBUG

   // Operatorfunktionen
      CInputMatrix &operator = (const CInputMatrix &matrix);
      CInputMatrix &operator = (const CMatrix &matrix);
      BOOL operator == (const CInputMatrix &matrix) const;
      CInputMatrix &operator << (const CLine &line);
      const CInputMatrix &operator >> (CLine &line) const;
      BOOL operator != (const CInputMatrix &matrix) const;

   // Freunde
      friend ostream &operator << (ostream &stream, const CInputMatrix &mat);
      friend istream &operator >> (istream &stream, CInputMatrix &mat);
}; // CInputMatrix
```

```
/*============================================================================*/
/* Klasse    | CWeightVector                                                  */
/* Inhalt    | Klasse zur Beschreibung eines Gewichtsvektors eines einzel- */
/*           | nen Neurons aus einem neuronalen Netz                          */
/*----------------------------------------------------------------------------*/
/* Bemerkung | Keine                                                          */
/*============================================================================*/

class CWeightVector : public CVector
{
   DECLARE_SERIAL(CWeightVector);

   public:

   // Konstruktoren und Destruktoren
      CWeightVector();
      CWeightVector(UINT nNrComp, double dBorder = 1.0);
      CWeightVector(const CWeightVector &vector);

   // Mitgliedsfunktionen
      void Insert(UINT nIndex, double dBorder = 1.0);
      void Expand(double dBorder = 1.0);
      void Init(UINT nNrComp, double dBorder = 1.0);
      static void Seed();
      static double Rand(double dBorder);

   // Operatorfunktionen
      CWeightVector &operator = (const CVector &vector);
}; // CWeightVector


/*============================================================================*/
/* Klasse    | CWeightMatrix                                                  */
/* Inhalt    | Klasse zur Beschreibung mehrerer Gewichtsvektoren einzelner */
/*           | Neuronen aus einem neuronalen Netz                             */
/*----------------------------------------------------------------------------*/
/* Bemerkung | Keine                                                          */
/*============================================================================*/

class CWeightMatrix : public CMatrix
{
   DECLARE_SERIAL(CWeightMatrix);

   public:

   // Konstruktoren und Destruktoren
      CWeightMatrix();
      CWeightMatrix(UINT nNrLines, UINT nNrColumns, double dBorder = 1.0);
      CWeightMatrix(const CWeightMatrix &matrix);

   // Mitgliedsfunktionen
      void Init(UINT nNrLines, UINT nNrColumns, double dBorder = 1.0);
      void Insert(UINT nIndex, UINT nPart, double dBorder = 1.0);
      void Expand(UINT nPart, double dBorder = 1.0);

   // Operatorfunktionen
      CWeightMatrix &operator = (const CMatrix &matrix);
}; // CWeightMatrix
```

```
/*============================================================================*/
/* Klasse    | CPosition                                                      */
/* Inhalt    | Klasse zur Beschreibung der Position eines Punktes in einem */
/*           | zweidimensionalen Eingabefeld der Größe n x n, dessen Koor- */
/*           | dinaten nur durch die Indizes 0,1,...,(n^2 - 1) eines ent-  */
/*           | sprechenden eindimensionalen Vektors gegeben sind.          */
/*----------------------------------------------------------------------------*/
/* Bemerkung | Binäre Zuweisung und binärer Kopier-Konstruktor erlaubt     */
/*============================================================================*/

struct CPosition
{
   // Daten der Klasse
      int  m_x,                              // X-Koordinate des Punktes
           m_y;                              // Y-Koordinate des Punktes
      UINT m_nNrColumns;                     // Anzahl der Spalten im Feld

   // Konstruktoren und Destruktoren
      CPosition(UINT nNrColumns);

   // Mitgliedsfunktionen
      void Set(UINT nIndex);
}; // CPosition


/*============================================================================*/
/* Klasse    | CHabibiMatrix                                                  */
/* Inhalt    | Klasse für quadratische Korrelationsmatrizen, die mit Hilfe */
/*           | der Korrelationsfunktion von A. Habibi und P. A. Wintz be-  */
/*           | rechnet werden; nachzulesen in IEEE Transactions on Commu-  */
/*           | nication Technology, Vol. COM-19, No. 1, Februar 1971.      */
/*----------------------------------------------------------------------------*/
/* Bemerkung | Die Korrelationsfunktion von Habibi und Wintz berechnet in  */
/*           | ihrer ursprünglichen Form einen Punkttensor vierter Ordnung */
/*           | und keine quadratische Matrix. Hier soll aber eine Korrela- */
/*           | tionsmatrix einer eindimensionalen linearen Transformation  */
/*           | bestimmt werden. Daher wird angenommen, daß ein n x n Feld  */
/*           | von Eingabedaten in einem Vektor mit n * n Komponenten ab-  */
/*           | gelegt wird, indem jede der n Zeilen des Feldes hinterein-  */
/*           | ander in den Vektor kopiert wird. Die quadratische Korrela- */
/*           | tionsmatrix stellt dann eine (n * n) x (n * n) Matrix dar.  */
/*           | Für die Berechnungen werden auf Basis der Größe n die Koor- */
/*           | dinaten (i,j) i,j = 1,2,...,n des Feldes der Eingabedaten   */
/*           | verwendet, und nicht die Koordinaten des konstruierten Vek- */
/*           | tors. Dabei wird n als 'n = sqrt(CMatrix::m_nNrLines)' oder */
/*           | als 'n = sqrt(CMatrix::m_nNrColumns)' angenommen.           */
/*============================================================================*/

class CHabibiMatrix : public CSquareMatrix
{
   DECLARE_SERIAL(CHabibiMatrix);

   public:

   // Konstruktoren und Destruktoren
      CHabibiMatrix();
      CHabibiMatrix(UINT nDim);
      CHabibiMatrix(const CHabibiMatrix &matrix);

   // Mitgliedsfunktionen
      void Init(UINT nDim);
      void CalcCorrelation(double dAlpha, double dBeta);

   // Operatorfunktionen
      CHabibiMatrix &operator = (const CMatrix &matrix);

   protected:

   // Mitgliedsfunktionen
      BOOL IsDimOk(UINT nDim);
}; // CHabibiMatrix
```

```
/*============================================================================*/
/* Klasse    | CSortVector                                                    */
/* Inhalt    | Klasse für Vektoren, die ihre Komponenten auf- oder abstei- */
/*           | gend sortieren können. Zusätzlich kann nach einem Sortier-  */
/*           | vorgang für jede Komponente ermittelt werden, welchen Index */
/*           | sie vor der Sortierung hatte.                                  */
/*----------------------------------------------------------------------------*/
/* Bemerkung | Keine                                                          */
/*============================================================================*/

class CSortVector : public CVector
{
   DECLARE_SERIAL(CSortVector);

   protected:

   // Daten der Klasse
      BOOL       m_bOrder,                    // Ordnung einer Sortierung
                 m_bStreamIO;                 // Stream I/O mit den Indizes
      CUIntArray m_OriginalIndices;           // Indizes vor der Sortierung

   public:

   // Konstruktoren und Destruktoren
      CSortVector();
      CSortVector(UINT nNrComp);
      CSortVector(const CSortVector &vector);

   // Mitgliedsfunktionen
      virtual void Serialize(CArchive &archive);
      void Init(UINT nNrComp);
      void Empty();
      UINT GetOriginalIndex(UINT nIndex) const;
      void Sort();
      void Swap(UINT nIndex1, UINT nIndex2);
      BOOL GetOrder() const;
      BOOL GetStreamIO() const;
      void Expand();
      void Reduce();
      void SetOrder(BOOL bOrder);
      void SetStreamIO(BOOL bStreamIO);
      void Insert(UINT nIndex);
      void Delete(UINT nIndex);

#ifdef _DEBUG
      virtual void AssertValid() const;
      virtual void Dump(CDumpContext &dc) const;
#endif // _DEBUG

   // Operatorfunktionen
      CSortVector &operator = (const CSortVector &vector);
      CSortVector &operator = (const CVector &vector);

   protected:

   // Mitgliedsfunktionen
      void InitIndices(UINT nNrComp);
      virtual void QuickSort(UINT nLeft, UINT nRight);

   public:

   // Freunde
      friend ostream &operator << (ostream &stream, const CSortVector &vec);
      friend istream &operator >> (istream &stream, CSortVector &vec);
}; // CSortVector
```

Anhang 5: Klassendeklarationen der Bibliothek SIC

```
/*=========================================================================*/
/* Klasse    | CSimulation                                                 */
/* Inhalt    | Klasse zur Beschreibung von Simulationen, welche M Schritte */
/*           | lang laufen werden, aber zwischendurch pausieren können.    */
/*-------------------------------------------------------------------------*/
/* Bemerkung | Bei dieser Klasse handelt es sich um eine abstrakte Klasse. */
/*=========================================================================*/

class CSimulation : public CObject
{
   DECLARE_DYNAMIC(CSimulation);

   protected:

   // Daten der Klasse
      UINT m_nStep,                          // Aktueller Durchlauf
           m_nNrSteps;                       // Anzahl aller Durchläufe

   public:

   // Konstruktoren und Destruktoren
      CSimulation();
      CSimulation(UINT nNrSteps);

   // Mitgliedsfunktionen
      virtual void Serialize(CArchive &archive);
      virtual void Run(UINT nPeriod) = 0;
      void Init(UINT nNrSteps);
      void Empty();
      UINT GetStep() const;
      UINT GetNrSteps() const;

#ifdef _DEBUG
      virtual void Dump(CDumpContext &dc) const;
#endif // _DEBUG

   // Operatorfunktionen
      CSimulation &operator = (const CSimulation &simulation);
}; // CSimulation
```

```
/*============================================================================*/
/* Klasse    | CNeuronLayer                                                   */
/* Inhalt    | Klasse zur Beschreibung von Simulationen einer Schicht von  */
/*           | Neuronen, wobei jede Simulation insgesamt M Schritte läuft. */
/*           | Die Eingabemenge für das neuronale Netz liegt dabei zeilen- */
/*           | weise in der Matrix 'm_X', die Ausgabemenge zeilenweise in  */
/*           | der Matrix 'm_Y'. Alle Gewichtsvektoren der Neuronen liegen */
/*           | wieder zeilenweise in der Matrix 'm_W'. Alle Auftrittswahr- */
/*           | scheinlichkeiten in 'm_X' und 'm_Y' werden mit dem konstan- */
/*           | ten Wert '1.0 / nNrInput' initialisiert.                    */
/*-----------+----------------------------------------------------------------*/
/* Bemerkung | Es werden nur eindimensionale Schichten berücksichtigt. Bei */
/*           | dieser Klasse handelt es sich um eine abstrakte Klasse.     */
/*============================================================================*/

class CNeuronLayer : public CSimulation
{
   DECLARE_DYNAMIC(CNeuronLayer);

   protected:

   // Daten der Klasse
      UINT m_nNrNeuron,                       // Anzahl der Neuronen
           m_nNrInput,                        // Anzahl der Eingaben
           m_nDimInput;                       // Komponenten pro Eingabe

   public:

   // Daten der Klasse
      CInputMatrix  m_X,                      // Menge aller Eingaben
                    m_Y;                      // Menge aller Ausgaben
      CWeightMatrix m_W;                      // Gewichte aller Neuronen

   // Konstruktoren und Destruktoren
      CNeuronLayer();

      CNeuronLayer(UINT nNrSteps, UINT nNrNeuron,
                   UINT nNrInput, UINT nDimInput);

   // Mitgliedsfunktionen
      virtual void Serialize(CArchive &archive);
      void Init(UINT nNrSteps, UINT nNrNeuron, UINT nNrInput, UINT nDimInp);
      void Empty();

#ifdef _DEBUG
      virtual void AssertValid() const;
      virtual void Dump(CDumpContext &dc) const;
#endif // _DEBUG

   // Operatorfunktionen
      CNeuronLayer &operator = (const CNeuronLayer &layer);
}; // CNeuronLayer
```

```
/*============================================================================*/
/* Klasse    | CLinearLayer                                                   */
/* Inhalt    | Klasse zur Beschreibung von Simulationen einer Schicht von     */
/*           | Neuronen mit linearer Ausgabefunktion 'y = w*x', wobei jede    */
/*           | Simulation insgesamt M Schritte läuft. Simulationen können     */
/*           | im Online- oder Offline-Verfahren durchgeführt werden.         */
/*----------------------------------------------------------------------------*/
/* Bemerkung | Es werden nur eindimensionale Schichten berücksichtigt. Bei    */
/*           | dieser Klasse handelt es sich um eine abstrakte Klasse.        */
/*============================================================================*/

class CLinearLayer : public CNeuronLayer
{
   DECLARE_DYNAMIC(CLinearLayer);

   public:

   // Konstruktoren und Destruktoren
      CLinearLayer();

      CLinearLayer(UINT nNrSteps, UINT nNrNeuron,
                   UINT nNrInput, UINT nDimInput);

   protected:

   // Mitgliedsfunktionen
      void OnlineTransformation(UINT nStep);
      void OfflineTransformation();

   public:

   // Operatorfunktionen
      CLinearLayer &operator = (const CLinearLayer &layer);
}; // CLinearLayer

/*============================================================================*/
/* Klasse    | CLinearVoid                                                    */
/* Inhalt    | Klasse zur Beschreibung von Simulationen einer Schicht von     */
/*           | Neuronen mit linearer Ausgabefunktion, bei denen Zwischen-     */
/*           | ergebnisse in beliebigen Strukturen gespeichert werden, die    */
/*           | erst in abgeleiteten Klassen zur Verfügung gestellt werden.    */
/*           | Jede Simulation wird M Schritte lang laufen, kann aber zwi-    */
/*           | schendurch in bestimmten Perioden pausieren.                   */
/*----------------------------------------------------------------------------*/
/* Bemerkung | Es werden nur eindimensionale Schichten berücksichtigt. Bei    */
/*           | dieser Klasse handelt es sich um eine abstrakte Klasse.        */
/*============================================================================*/

class CLinearVoid : public CLinearLayer
{
   DECLARE_DYNAMIC(CLinearVoid);

   public:

   // Konstruktoren und Destruktoren
      CLinearVoid();

      CLinearVoid(UINT nNrSteps, UINT nNrNeuron,
                  UINT nNrInput, UINT nDimInput);

   // Mitgliedsfunktionen
      virtual void Run(UINT nPeriod);

   // Operatorfunktionen
      CLinearVoid &operator = (const CLinearVoid &layer);

   protected:

   // Mitgliedsfunktionen
      virtual void OneStep() = 0;
}; // CLinearVoid
```

```
/*============================================================================*/
/* Klasse    | CLinearDouble                                                  */
/* Inhalt    | Klasse zur Beschreibung von Simulationen einer Schicht von     */
/*           | Neuronen mit linearer Ausgabefunktion, bei denen Zwischen-     */
/*           | ergebnisse vom Datentyp 'double' anfallen. Jede Simulation     */
/*           | wird M Schritte lang laufen, kann aber zwischendurch in be-    */
/*           | stimmten Perioden pausieren.                                   */
/*----------------------------------------------------------------------------*/
/* Bemerkung | Es werden nur eindimensionale Schichten berücksichtigt. Bei    */
/*           | dieser Klasse handelt es sich um eine abstrakte Klasse.        */
/*============================================================================*/

class CLinearDouble : public CLinearLayer
{
   DECLARE_DYNAMIC(CLinearDouble);

   protected:

   // Daten der Klasse
      CVector m_Results;                      // Alle Zwischenergebnisse

   public:

   // Konstruktoren und Destruktoren
      CLinearDouble();

      CLinearDouble(UINT nNrSteps, UINT nNrNeuron,
                    UINT nNrInput, UINT nDimInput);

   // Mitgliedsfunktionen
      virtual void Serialize(CArchive &archive);
      virtual void Run(UINT nPeriod);
      CVector &GetResults();
      void Init(UINT nNrSteps, UINT nNrNeuron, UINT nNrInput, UINT nDimInp);
      void Empty();

#ifdef _DEBUG
      virtual void AssertValid() const;
      virtual void Dump(CDumpContext &dc) const;
#endif // _DEBUG

   // Operatorfunktionen
      CLinearDouble &operator = (const CLinearDouble &layer);

   protected:

   // Mitgliedsfunktionen
      virtual double OneStep() = 0;
}; // CLinearDouble
```

```
/*============================================================================*/
/* Klasse    | COjaNeuron                                                     */
/* Inhalt    | Klasse für die Beschreibung der Simulation eines einzelnen  */
/*           | Neurons mit der Lernregel von E. Oja.                          */
/*----------------------------------------------------------------------------*/
/* Bemerkung | Es wird auf den Matrizen 'm_X', 'm_Y' und 'm_W' gerechnet.  */
/*============================================================================*/

class COjaNeuron : public CLinearVoid
{
   DECLARE_SERIAL(COjaNeuron);

   public:

   // Konstruktoren und Destruktoren
      COjaNeuron();
      COjaNeuron(const COjaNeuron &neuron);
      COjaNeuron(UINT nNrSteps, UINT nNrInput, UINT nDimInput);

   // Mitgliedsfunktionen
      void Init(UINT nNrSteps, UINT nNrInput, UINT nDimInput);

   // Operatorfunktionen
      COjaNeuron &operator = (const COjaNeuron &neuron);

   protected:

   // Mitgliedsfunktionen
      virtual void OneStep();
      virtual double Gamma();
}; // COjaNeuron


/*============================================================================*/
/* Funktion | COjaNeuron::OneStep                                            */
/* Inhalt   | Simulation eines Schritts von insgesamt M Schritten          */
/*----------------------------------------------------------------------------*/
/* Eingabe  | Keine                                                          */
/* Ausgabe  | Keine                                                          */
/* Ausnahmen| CMemoryException, CMathArrayException                          */
/*----------------------------------------------------------------------------*/
/* Bemerkung| Es wird auf den Matrizen 'm_X', 'm_Y' und 'm_W' gerechnet.   */
/*============================================================================*/

void COjaNeuron::OneStep()
{
   ASSERT_VALID(this);                          // Vorbedingungen überprüfen

   CVector x(m_nDimInput),                      // Eingabe in einem Schritt
           w(m_nDimInput);                      // Gewichtsvektor des Neurons
   double  y;                                   // Ausgabe in einem Schritt
   UINT    p = m_nStep % m_nNrInput;            // Nummer des Eingabemusters

   OnlineTransformation(p);                     // Lineare Transformation

   y = m_Y[CIndex(p, 0)];                       // Daten für Lernregel holen
   m_X >> CLine(x, p);
   m_W >> CLine(w, 0);

   w += Gamma() * y * (x - w * y);              // Lernregel von E. Oja

   m_W << CLine(w, 0);                          // Neues Gewicht schreiben
} // COjaNeuron::OneStep
```

```
/*============================================================================*/
/* Klasse    | CSubspace                                                      */
/* Inhalt    | Klasse für die Beschreibung der Simulation eines neuronalen */
/*           | Netzes mit dem Subspace-Algorithmus von E. Oja.              */
/*----------------------------------------------------------------------------*/
/* Bemerkung | Es wird auf den Matrizen 'm_X', 'm_Y' und 'm_W' gerechnet.  */
/*============================================================================*/

class CSubspace : public CLinearVoid
{
   DECLARE_SERIAL(CSubspace);

   public:

   // Konstruktoren und Destruktoren
      CSubspace();
      CSubspace(const CSubspace &subspace);
      CSubspace(UINT nNrSteps, UINT nNrNeuron, UINT nNrInput, UINT nDimInp);

   // Operatorfunktionen
      CSubspace &operator = (const CSubspace &subspace);

   protected:

   // Mitgliedsfunktionen
      virtual void OneStep();
      virtual double Gamma();
}; // CSubspace


/*============================================================================*/
/* Funktion | CSubspace::OneStep                                              */
/* Inhalt   | Simulation eines Schritts von insgesamt M Schritten          */
/*----------------------------------------------------------------------------*/
/* Eingabe  | Keine                                                          */
/* Ausgabe  | Keine                                                          */
/* Ausnahmen| CMemoryException, CMathArrayException                          */
/*----------------------------------------------------------------------------*/
/* Bemerkung| Es wird auf den Matrizen 'm_X', 'm_Y' und 'm_W' gerechnet.   */
/*============================================================================*/

void CSubspace::OneStep()
{
   ASSERT_VALID(this);                          // Vorbedingungen überprüfen

   CInputVector  x(m_nDimInput),                // Muster der Eingabemenge
                 y(m_nNrNeuron);                // Muster der Ausgabemenge
   CWeightVector w(m_nDimInput);                // Gewichtsvektor k oder i
   CVector       z(m_nDimInput);                // Feedback Vektor
   UINT          p = m_nStep % m_nNrInput;      // Nummer des Eingabemusters

   OnlineTransformation(p);                     // Lineare Transformation

   m_X >> CLine(x, p);                          // Forward Pass
   m_Y >> CLine(y, p);

   for (UINT k = 0; k < m_nNrNeuron; k++)       // Backward Pass
   {
      m_W >> CLine(w, k);
      z += y[k] * w;
   }

   for (UINT i = 0; i < m_nNrNeuron; i++)       // Weight Update Pass
   {
      m_W >> CLine(w, i);
      w += Gamma() * y[i] * (x - z);
      m_W << CLine(w, i);
   }
} // CSubspace::OneStep
```

```
/*============================================================================*/
/* Klasse    | CSilvaVoid                                                     */
/* Inhalt    | Gemeinsame Basisklasse für Simulationen zur Daten-Orthonor- */
/*           | malisierung von M. Silva und Luis B. Almeida, wo keinerlei  */
/*           | Zwischenergebnisse anfallen, die gesondert gespeichert wer- */
/*           | den müssen.                                                    */
/*----------------------------------------------------------------------------*/
/* Bemerkung | Bei dieser Klasse handelt es sich um eine abstrakte Klasse. */
/*============================================================================*/

class CSilvaVoid : public CLinearVoid
{
   DECLARE_DYNAMIC(CSilvaVoid);

   public:

   // Konstruktoren und Destruktoren
      CSilvaVoid();
      CSilvaVoid(UINT nNrSteps, UINT nNrNeuron, UINT nNrInput);

   // Mitgliedsfunktionen
      void Init(UINT nNrSteps, UINT nNrNeuron, UINT nNrInput);

   // Operatorfunktionen
      CSilvaVoid &operator = (const CSilvaVoid &silva);
}; // CSilvaVoid


/*============================================================================*/
/* Klasse    | CSilvaVoidBatch                                                */
/* Inhalt    | Klasse für eine Batch-Version des Algorithmus zur Daten-Or- */
/*           | thonormalisierung von M. Silva und Luis B. Almeida          */
/*----------------------------------------------------------------------------*/
/* Bemerkung | Es wird auf den Matrizen 'm_X', 'm_Y' und 'm_W' gerechnet.  */
/*============================================================================*/

class CSilvaVoidBatch : public CSilvaVoid
{
   DECLARE_SERIAL(CSilvaVoidBatch);

   public:

   // Konstruktoren und Destruktoren
      CSilvaVoidBatch();
      CSilvaVoidBatch(const CSilvaVoidBatch &silva);
      CSilvaVoidBatch(UINT nNrSteps, UINT nNrNeuron, UINT nNrInput);

   // Operatorfunktionen
      CSilvaVoidBatch &operator = (const CSilvaVoidBatch &silva);

   protected:

   // Mitgliedsfunktionen
      virtual void OneStep();
      virtual double Gamma();
}; // CSilvaVoidBatch
```

```
/*============================================================================*/
/* Klasse    | CSilvaVoidDistributed                                          */
/* Inhalt    | Klasse für eine Distributed-Version des Algorithmus zur Da- */
/*           | ten-Orthonormalisierung von M. Silva und Luis B. Almeida    */
/*----------------------------------------------------------------------------*/
/* Bemerkung | Es wird auf den Matrizen 'm_X', 'm_Y' und 'm_W' gerechnet.  */
/*============================================================================*/

class CSilvaVoidDistributed : public CSilvaVoid
{
   DECLARE_SERIAL(CSilvaVoidDistributed);

   public:

   // Konstruktoren und Destruktoren
      CSilvaVoidDistributed();
      CSilvaVoidDistributed(const CSilvaVoidDistributed &silva);
      CSilvaVoidDistributed(UINT nNrSteps, UINT nNrNeuron, UINT nNrInput);

   // Operatorfunktionen
      CSilvaVoidDistributed &operator = (const CSilvaVoidDistributed &silva);

   protected:

   // Mitgliedsfunktionen
      virtual void OneStep();
      virtual double Gamma();
}; // CSilvaVoidDistributed


/*============================================================================*/
/* Funktion | CSilvaVoidDistributed::OneStep                                  */
/* Inhalt   | Simulation eines Schritts von insgesamt M Schritten, bei dem */
/*          | alle Gewichtsvektoren des Netzes korrigiert werden           */
/*----------------------------------------------------------------------------*/
/* Eingabe  | Keine                                                          */
/* Ausgabe  | Keine                                                          */
/* Ausnahmen| CMemoryException, CMathArrayException                          */
/*----------------------------------------------------------------------------*/
/* Bemerkung| Es wird auf den Matrizen 'm_X', 'm_Y' und 'm_W' gerechnet.   */
/*============================================================================*/

void CSilvaVoidDistributed::OneStep()
{
   ASSERT_VALID(this);                          // Vorbedingungen überprüfen

   CInputVector  y(m_nNrNeuron);                // Muster der Ausgabemenge
   CWeightVector w(m_nDimInput);                // Gewichtsvektor k oder j
   CVector       z(m_nDimInput);                // Verteilte Implementierung
   UINT          p = m_nStep % m_nNrInput;      // Nummer des Eingabemusters

   OnlineTransformation(p);                     // Lineare Transformation
   m_Y >> CLine(y, p);                          // Forward Pass

   for (UINT k = 0; k < m_nNrNeuron; k++)       // Backward Pass
   {
      m_W >> CLine(w, k);
      z += y[k] * w;
   }

   for (UINT j = 0; j < m_nNrNeuron; j++)       // Weight Update Pass
   {
      m_W >> CLine(w, j);
      w += Gamma() * w - Gamma() * y[j] * z;
      m_W << CLine(w, j);
   }
} // CSilvaVoidDistributed::OneStep
```

```
/*============================================================================*/
/* Klasse    | CRipNetVoidBatch                                               */
/* Inhalt    | Klasse zur Beschreibung einer 'RipNet'-Simulation, bei der  */
/*           | eine Batch-Version des Lernverfahrens verwendet wird. Diese */
/*           | Simulation läuft ab, ohne das Zwischenergebnisse gesondert  */
/*           | abgespeichert werden müssen.                                   */
/*----------------------------------------------------------------------------*/
/* Bemerkung | Es wird auf den Matrizen 'm_X', 'm_Y' und 'm_W' gerechnet.  */
/*============================================================================*/

class CRipNetVoidBatch : public CLinearVoid
{
   DECLARE_SERIAL(CRipNetVoidBatch);

   public:

   // Konstruktoren und Destruktoren
      CRipNetVoidBatch();
      CRipNetVoidBatch(const CRipNetVoidBatch &ripnet);
      CRipNetVoidBatch(UINT nNrSteps, UINT nNrNeuron, UINT nNrInput);

   // Mitgliedsfunktionen
      void Init(UINT nNrSteps, UINT nNrNeuron, UINT nNrInput);

   // Operatorfunktionen
      CRipNetVoidBatch &operator = (const CRipNetVoidBatch &ripnet);

   protected:

   // Mitgliedsfunktionen
      virtual void OneStep();
      virtual double Gamma();
}; // CRipNetVoidBatch
```

```
/*============================================================================*/
/* Funktion | CRipNetVoidBatch::OneStep                                       */
/* Inhalt   | Simulation eines Schritts von insgesamt M Schritten, bei dem */
/*          | alle Gewichtsvektoren des Netzes korrigiert werden            */
/*----------------------------------------------------------------------------*/
/* Eingabe  | Keine                                                          */
/* Ausgabe  | Keine                                                          */
/* Ausnahmen| CMemoryException, CMathArrayException                          */
/*----------------------------------------------------------------------------*/
/* Bemerkung| Es wird auf den Matrizen 'm_X', 'm_Y' und 'm_W' gerechnet.   */
/*============================================================================*/

void CRipNetVoidBatch::OneStep()
{
   ASSERT_VALID(this);                          // Vorbedingungen überprüfen

   CWeightVector w(m_nDimInput);                // Gewichtsvektor Nummer k
   CInputVector  x(m_nDimInput),                // Einzelnes Eingabemuster
                 y(m_nNrNeuron);                // Einzelnes Ausgabemuster
   CVector       qi(m_nDimInput),               // Berechne <x * yi> für i
                 qk(m_nDimInput),               // Berechne <x * yk> für k
                 si(m_nDimInput);               // Summe (uki * qi) über i
   double        uki;                           // Rechne <yk * yi> für k, i

   OfflineTransformation();                     // Lineare Transformation

   for (UINT k = 0; k < m_nNrNeuron; k++)       // Für alle Gewichtsvektoren
   {
      si.Fill(0.0);

      for (UINT i = 0; i < m_nNrNeuron; i++)  // Summe (uki * qi) über i
      {
         uki = 0.0;
         qi.Fill(0.0);

         for (UINT p = 0; p < m_nNrInput; p++) // Über alle Eingabemuster
         {
            m_X >> CLine(x, p);
            m_Y >> CLine(y, p);

            uki += y[k] * y[i] * y.GetProb(); // Rechne <yk * yi> für k, i
            qi += x * y[i] * x.GetProb();     // Berechne <x * yi> für i
         }

         si += uki * qi;
      }

      qk.Fill(0.0);

      for (UINT p = 0; p < m_nNrInput; p++)   // Über alle Eingabemuster
      {
         m_X >> CLine(x, p);
         m_Y >> CLine(y, p);

         qk += x * y[k] * x.GetProb();        // Berechne <x * yk> für k
      }

      m_W >> CLine(w, k);
      w -= Gamma() * (si - qk);               // 'RipNet' Lernverfahren
      m_W << CLine(w, k);
   }
} // CRipNetVoidBatch::OneStep
```

Anhang 6: Verwendete Definitionen und Begriffe

X, Y	Nachrichtenquelle X bzw. Nachrichtenziel Y
I(x)	Information eines Zeichens x
H(X)	Entropie einer Nachrichtenquelle X
H_0	Entscheidungsgehalt einer Nachrichtenquelle
C	Kanalkapazität eines Übertragungskanals
e	Eulersche Zahl 2,718281828
lg	Briggscher Logarithmus zur Basis 10
ln	Natürlicher Logarithmus zur Basis e
ld	Dualer Logarithmus zur Basis 2
exp	Exponentialfunktion
P(x)	Wahrscheinlichkeitsfunktion einer Zufallsgröße X
p(x)	Dichtefunktion einer kontinuierlichen Zufallsgröße X
W(X)	Wertemenge bzw. Wertebereich einer Zufallsgröße X
$\sigma(X)$	Standardabweichung einer Zufallsgröße X
$\sigma^2(X)$	Varianz einer Zufallsgröße X
A	Matrix A
a	Spaltenvektor a
A^T	Transponierte der Matrix A
a^T	Zeilenvektor a
V	Endlich-dimensionaler Vektorraum mit Skalarprodukt
(…, …)	Allgemeines reelles Skalarprodukt im Vektorraum V
|…|	Norm
$\Re$	Körper der reellen Zahlen einschließlich Nullelement
$\Re_+^*$	Menge der positiven reellen Zahlen ohne Nullelement
$\aleph$	Menge der natürlichen Zahlen
$\overline{z}$	Zu z konjugiert-komplexe Zahl
E^n	Euklidischer n-dimensionaler Vektorraum
x	Spaltenvektor von Eingabewerten
y	Spaltenvektor von Ausgabewerten
z	Aktivität eines Neurons
w, w^T	Spaltenvektor bzw. Zeilenvektor von Gewichten
w^i	Gewichtsvektor i
w_i	Komponente i eines Gewichtsvektors
w_i^k	Komponente i eines Gewichtsvektors k
W	Gewichtsmatrix mit Gewichtsvektoren in den Matrixzeilen
T	Schwellwert eines Neurons
z(.)	Aktivitätsfunktion eines Neurons
S(.)	Ausgabefunktion eines Neurons

e^i	Eigenvektor i
λ_i	Eigenwert zum Eigenvektor e^i
C_{XX}	Korrelationsmatrix einer Eingabemenge X
C_{YY}	Korrelationsmatrix einer Ausgabemenge Y
C_X	Kovarianzmatrix einer Eingabemenge X
C_Y	Kovarianzmatrix einer Ausgabemenge Y
I	Einheitsmatrix
Tr(C)	Spur einer quadratischen Matrix C
t	Zeitschritt
$\gamma(t)$, $\mu(t)$	Schrittabhängige Proportionalitätskonstante, Lernrate
$\langle f(t) \rangle_t$	Erwartungswert von f(t) bezüglich aller Werte von t
Γ	Kontinuierliche Punktmenge
Δ	Größe einer kontinuierlichen Teilmenge von Γ
g, h, …, n	Indizes für Summen, Komponenten von Vektoren

Anhang 7: Literaturverzeichnis

[BRA91] R. Brause: *Neuronale Netze*; B. G. Teubner Stuttgart 1991

[BRO93] I. N. Bronstein, K. A. Semendjajew: *Taschenbuch der Mathematik*; Verlag Harri Deutsch, Thun und Frankfurt/M 1993

[DUS77] J. Duske, H. Jürgensen: *Codierungstheorie*; Bibliographisches Institut Mannheim/Wien/Zürich, B. I. Wissenschaftsverlag 1977

[FIH73] G. M. Fichtenholz: *Differential- und Integralrechnung I*; VEB Deutscher Verlag der Wissenschaften, Herausgegeben von H. Grell, K. Maruhn sowie W. Rinow, Band 61, Berlin 1973

[FOR83] O. Forster: *Anaysis I - Differential- und Integralrechnung einer Veränderlichen*; Vieweg & Sohn Verlagsgesellschaft mbH, Braunschweig 1983

[FOR84] O. Forster: *Analysis II - Differentialrechnung im* R^n; Vieweg & Sohn Verlagsgesellschaft mbH, Braunschweig 1984

[HAB71] A. Habibi, P. A. Wintz: *Image Coding by Linear Transformation and Block Quantization*; IEEE Transactions on Communication Technology, Vol. COM-19, No. 1, pp. 50-62, February 1971

[HOF91] N. Hoffmann: *Simulation Neuronaler Netze*; Vieweg & Sohn Verlagsgesellschaft mbH, Braunschweig 1991

[HUA63] J. J. Y. Huang, P. M. Schultheiss: *Block Quantization of correlated Gaussian random variables*; IEEE Transactions on Communication Systems, Vol. CS-11, pp. 289-296, September 1963

[JAY84] N. S. Jayant, P. Noll: *Digital Coding Of Waveforms*; Prentice-Hall Inc. Englewood Cliffs, New Jersey 1984

[KAM73] T. Kameda, K. Weihrauch: *Einführung in die Codierungstheorie I*; Bibliographisches Institut Mannheim/Wien/Zürich, B. I. Wissenschaftsverlag 1973

[KRO90] A. Krogh, J. A. Hertz: *Hebbian Learning of Principal Components*; Parallel Processing in Neural Systems and Computers, R. Eckmiller, G. Hartmann and G. Hauske (Editors), © Elsevier Science Publishers B.V. North-Holland, pp. 183-186, 1990

[KÜH90] H. Kühnel, P. Tavan: *The Anti-Hebb Rule derived from information theory*; Parallel Processing in Neural Systems and Computers, R. Eckmiller, G. Hartmann and G. Hauske (Editors), © Elsevier Science Publishers B.V. North-Holland, pp. 187-190, 1990

[LAU79] J. Lauter, M. Breger: *Wahrscheinlichkeitsrechnung und Statistik*; Pädagogischer Verlag Schwann-Bagel GmbH, Düsseldorf 1979

[MAX60] J. Max: *Quantizing for minimum distortion*; IRE Transactions on Information Theory, Vol. IT-6, pp. 7-12, March 1960

[MIL90] O. Mildenberger: *Informationstheorie und Codierung*; Vieweg & Sohn Verlagsgesellschaft mbH, Braunschweig 1990

[OJA82] E. Oja: *A simplified neuron model as a principal component analyzer*; J. Math. Biology, Vol. 15, pp. 267-273, 1982

[OJA89] E. Oja: *Neural Networks, Principal Components, and Subspaces*; International Journal of Neural Systems, Vol. 1, No. 1, pp. 61-68, April 1989

[OJA92] E. Oja: *Principal Components, Minor Components, and Linear Neural Networks*; Neural Networks, Vol. 5, pp. 927-935, Pergamon Press Ltd. 1992

[PAP91] A. Papoulis: *Probability, Random Variables, and Stochastic Processes*; McGraw-Hill, International Edition 1991

[PLU93] M. D. Plumbley: *Efficient Information Transfer and Anti-Hebbian Neural Networks*; Neural Networks, Vol. 6, pp. 823-833, Pergamon Press Ltd. 1993

[REI77] H. Reichardt: *Kleine Enzyklopädie Mathematik*; Verlag Harri Deutsch, Thun und Frankfurt/M 1977

[RUB90] J. Rubner, K. Schulten, P. Tavan: *A self-organizing network for complete feature extraction*; Parallel Processing in Neural Systems and Computers, R. Eckmiller, G. Hartmann and G. Hauske (Editors), © Elsevier Science Publishers B.V. North-Holland, pp. 365-368, 1990

[SAN89] T. D. Sanger: *Optimal unsupervised learning in a single-layer linear feedforward network*; Neural Networks, Vol. 2, pp. 459-473, 1989

[SCE90] E. Schöneburg, N. Hansen, A. Gawelczyk: *Neuronale Netzwerke*; Markt & Technik Verlag AG 1990

[SHA49] C. E. Shannon: *Communication in the Presence of Noise*; Proceedings of the IRE, Vol. 37, pp. 10-21, January 1949

[SHA76] C. E. Shannon, W. Weaver: *Mathematische Grundlagen der Informationstheorie*; R. Oldenbourg Verlag München Wien 1976

[SIL91] F. M. Silva, L. Almeida: *A distributed solution for data orthonormalization*; Artificial Neural Networks, T. Kohonen, K. Mäkisara, O. Simula, J. Kangas (Editors), © Elsevier Science Publishers B.V. (North-Holland), pp. 943-948, 1991

[STA83] U. Stammbach: *Lineare Algebra*; B. G. Teubner Stuttgart 1983

[STR92] B. Stroustrup: *Die C++ Programmiersprache*; Addison-Wesley Publishing Company 1992

[STU82] F. Stummel, K. Hainer: *Praktische Mathematik*; B. G. Teubner Stuttgart '82

[TOP74] F. Topsøe: *Informationstheorie*; B. G. Teubner Stuttgart 1974

[WIN68] P.A. Wintz, A. J. Kurtenbach: *Waveform error control in PCM telemetry*; IEEE Transactions on Information Theory, Vol. IT-14, pp 650-661, September 1968

[WIN72] P.A.Wintz: *Transform Picture Coding*; Proceedings of the IEEE, Vol. 60, No. 7, pp. 809-820, July 1972

[WOD69] R. C. Wood: *On Optimum Quantization*; IEEE Transactions on Information Theory, Vol. IT-15, No. 2, pp. 248-252, March 1969

Index

T

U

Ü

V

W

X

Z