Analyse und Auswertung von Zeitreihenprognosen mit dem Stuttgarter Neuronale Netze Simulator (SNNS)

Karsten Moranz

Universität Osnabrück

19. Mai 1998

Vorwort

Die vorliegende Diplomarbeit entstand im Anschluß an ein von Prof. Dr. Volker Sperschneider angebotenes Praktikum „Neuronale Netze” im Fachbereich Mathematik/Informatik der Universität Osnabrück. Diese Arbeit ist in der Arbeitsgruppe Praktische Informatik von Prof. Dr. Oliver Vornberger angesiedelt und Teil eines Projektes zur Prognose von Zeitreihen unter Leitung von Dipl.-Math. Frank M. Thiesing. Mein Dank gilt an dieser Stelle allen, die zum Gelingen dieser Arbeit beigetragen haben:

- Herrn Prof. Dr. Oliver Vornberger, für die Betreuung der Arbeit und für die Gespräche, durch die weitere Ideen und Arbeitsansätze erlangt wurden.
- Herrn Dipl.-Math. Frank M. Thiesing für die Betreuung der Arbeit und für rege Kommunikation, um Ideen und Ziele des Projektes zu verwirklichen, sowohl Kommunikation verbaler Art (per direktem Gespräch, Telefon) als auch non-verbaler Art (E-mail, Snail-mail),
- Herrn Prof. Dr. Volker Sperschneider, der das Neuropraktikum anbot, durch das ich gut vorbereitet in das Thema der Diplomarbeit einsteigen konnte,
- Herrn Cand.-Math Carsten Wilhelm und Herrn Cand.-Math Stephan Krone, die mir als weitere am Projekt beteiligte Personen gerade in der Anfangsphase mit Antworten auf meine Fragen zu helfen wußten,
- und meiner Frau Kirsten, die mit mir nicht nur diesen steinigen Weg gegangen ist.

Erklärung

Ich versichere, die vorliegende Arbeit selbständig angefertigt und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet zu haben.

Osnabrück, den 19. Mai 1998

Karsten Moranz

Inhaltsverzeichnis

1 Einleitung **1**

2 Das Prognosesystem **5**

2.1 Funktionsgruppen 5

2.2 Das Prognosesystem auf Dateiebene 6

2.3 Allgemeiner Ablauf einer Prognose 7

3 Der Stuttgarter Neuronale Netze Simulator **11**

3.1 Relevante Komponenten im Prognosesystem 11

3.2 Der Interpreter Batchman 12

4 SNNS im Prognosesystem **17**

4.1 Die ASCII-Netzdatei und der Netzgenerator 17

4.2 Batchman Templates 20

4.3 Die Standardprognose 22

5 Dateien im Detail **25**

5.1 Das BP-File 25

5.1.1 Die Parameter 27

5.2 Das Log-File 32

6 Kreuzvalidierung **37**

6.1 Parameter im Steuerungsfile 38

6.2 Die Kreuzvalidierung im Prognosesystem 38

6.3 Ausgabedateien 39

7 Sensitivitätsanalyse **41**

7.1 Praktischer Einsatz 41

7.2 Parameter im Steuerungsfile 42

7.3 Ablauf einer Sensitivitätsanalyse 44

7.4 Ausgabedateien 44

8 Pruning-Verfahren **49**

8.1 Parameter im Steuerungsfile 50

8.2 Ablauf des Prunings im Prognosesystem 51

8.3 Ausgabedateien 52

9 Fehler- oder Zielfunktionen **53**

9.1 Beispiele von Zielfunktionen 54

9.2 Das Lernverfahren BPM_ln_cosh 55

9.3 Implementation von Zielfunktionen in SNNS 56

10 Selbstorganisierende Karten **59**

10.1 Parameter im Steuerungsfile 60

10.2 Ablauf einer Klassenzerlegung mit anschließendem Prognoselauf 62

10.3 Ausgabedateien 62

11 Netztopologien **65**

11.1 Shortcutverbindungen 65

11.2 Verwendung einer Vorschicht 66

11.3 Jordan- und Elman-Netze 67

12 Auswertungen **71**

12.1 `zr_error.pl` 71

12.1.1 Leistungsmerkmale 72

12.1.2 Aufrufkonventionen 75

12.1.3 Eingabedateien 77

12.1.4 Ausgabedateien 79

12.1.5 Anwendungsbeispiele 80

13 Ergebnisse der Tegut-Abverkaufsprognosen **87**

13.1 Beschreibung der Daten 87

13.1.1 Artikeldaten 87

13.1.2 Globale Daten 88

13.1.3 Gruppierung der Artikel 88

13.2 Prognoseläufe 89

13.2.1 Zeiträume 89

13.2.2 Parametereinstellungen der Prognoseläufe 89

13.3 Ergebnisse 91

14 Zusammenfassung **99**

Literaturverzeichnis **101**

Abbildungsverzeichnis

2.1 Funktionsgruppen des Prognosesystems . . . 6

2.2 Dateistruktur des Prognosesystems . . . 9

2.3 Struktur einer Prognose . . . 10

3.1 Hauptkomponenten von SNNS . . . 12

3.2 Relevante Komponenten im Prognosesystem . . . 13

4.1 Darstellung eines Netzes in der graphischen Oberfläche von *SNNS* . . . 19

4.2 Prognose mit *SNNS* . . . 23

4.3 Ablauf eines Standard-Batchman-Skripts . . . 24

5.1 Parameter START und END . . . 27

5.2 Parameter ADVANCE . . . 27

5.3 Trainings-, Testfehlerverlauf (solid, gestrichelt) `ERR`: -8 . . . 28

5.4 Parameter *MAXERROR* . . . 29

5.5 Parameter *HIDDEN* . . . 30

5.6 Beispiel für `.X:3:2` . . . 31

5.7 Parameter zur Aufteilung der Patterngesamtheit . . . 31

5.8 Die Parameter *USE* und *UNKNOWN* . . . 33

5.9 Zusammenhänge der verschiedenen Patternmengen auf Zeitraum *START* bis *END* . . . 33

6.1 Die Parameter *RANDOM* und *USETST* . . . 39

6.2 Ablauf der Kreuzvalidierung . . . 40

7.1 Die zwei Ansätze zur Sensitivitätsanalyse (Quadrate repräsentieren Input-/Outputwerte und ihre Änderungen) . . . 43

7.2 Ablauf einer Sensitivitätsanalyse 46

7.3 Beispiel zur Sensitivitätsanalyse 47

8.1 Ablauf des Prunings 51

9.1 Zielfunktionen im Vergleich 54

9.2 Ableitung der Zielfunktion $\frac{1}{10} \cdot \ln(\cosh(10x))$ 55

10.1 Zerlegung der Patterngesamtheit in Klassen 60

10.2 Kohonen-Netz mit Gewichtsvektoren eines Neurons j 60

10.3 Ablauf der Entfaltung eines Kohonen-Netzes 62

10.4 Ablauf des Perl-Skripts `come_get_SOMe.pl` 63

11.1 3-schichtiges Feed-Forward-Netz 65

11.2 3-schichtiges Feed-Forward-Netz mit Shortcut-Verbindungen 66

11.3 3-schichtiges Feed-Forward-Netz mit Vorschicht 67

11.4 Jordan-Netz mit Kontextzelle 68

11.5 Elman-Netz mit Kontextzellen 69

12.1 Die vier Dimensionen 72

12.2 Zeitreihe: Tegut/AAA/215732, Prognoselauf: diplbsp 74

12.3 Zusammenhang zwischen *zr-List* und *pic-List* 79

12.4 Zeitreihe: CO2, Prognoselauf: lauf1 81

12.5 Zeitreihe: CO2, Prognoselauf: lauf1 81

12.6 Zeitreihe: CO2, Prognoselauf: lauf1 81

12.7 Verfahren: SNNS, Zeitreihe: SUNSPOTS 83

12.8 Verfahren: SNNS, Zeitreihe: SUNSPOTS 84

Tabellenverzeichnis

3.1 SNNS-Funktionsaufrufe in Batchman-Skripten 15

12.1 Fehlermaß: mape Prognoselauf: Test3 73

12.2 Zeitreihe: 215732 Prognoselauf: Test3 73

12.3 Fehlermaß: mape Prognoselauf: Test3 Sortierung: comp_snns_tegut 77

13.1 Artikelanzahlen in den Gruppen 88

13.2 Prognosezeiträume 89

13.3 Art und Anzahl eingehender Zeitreihen 90

13.4 Verwendete Lernverfahren 90

13.5 Beste Prognosegüten (MAPE) in den Zeiträumen T1 – T6 92

13.6 Die besten Netze im Mittel über die Zeiträume T1 – T6 im Vergleich zur Tegut-Prognose (MAPE) 93

13.7 Die besten Netze im Mittel über die Zeiträume T1 – T6 im Vergleich zur Tegut-Prognose (RMSE) 94

13.8 Über die Zeiträume T1 – T6 gemittelte Fehler getesteter Topologien der Gruppe S . . . 94

13.9 Über die Zeiträume T1 – T6 gemittelte Fehler getesteter Topologien der Gruppe SP . . 95

13.10 Über die Zeiträume T1 – T6 gemittelte Fehler getesteter Topologien der Gruppe SAP . 96

13.11 Über die Zeiträume T1 – T6 gemittelte Fehler getesteter Topologien der Gruppe LP . . 97

13.12 Über die Zeiträume T1 – T6 gemittelte Fehler getesteter Topologien der Gruppe LAP . 98

Kapitel 1

Einleitung

Viele in unserer Umwelt anzutreffenden dynamischen Systeme sind chaotischer oder sogar zufälliger Natur. Sie genügen keinen offensichtlichen Gesetzmäßigkeiten. Noch vor nicht allzu langer Zeit bemerkte der französische Mathematiker Pierre Simon de Laplace im Jahre 1776:

> „ Der momentane Zustand des Systems Natur ist offensichtlich eine Folge dessen, was er im vorherigen Moment war, und wenn wir uns eine Intelligenz vorstellen, die zu einem gegebenen Zeitpunkt alle Beziehungen zwischen den Teilen des Universums verarbeiten kann, so könnte sie Orte, Bewegungen und allgemeine Beziehungen zwischen all diesen Teilen für alle Zeitpunkte in Vergangenheit und Zukunft vorhersagen."

Den heutigen Standpunkt beschreibt der Mathematiker Henri Poincaré 1903:

> „Ein kleiner Fehler zu Anfang wird später einen großen Fehler zur Folge haben. Vorhersagen werden unmöglich, und wir haben ein zufälliges Ereignis."

Heutige rechnergestützte Prognosesysteme versuchen, ein exaktes Modell des Teils der Realität abzubilden, den sie prognostizieren wollen. Ein solches Prognosesystem kann auf verschiedener Basis aufbauen. Da gibt es Regelbasierte Systeme, Expertensysteme, Statistische Systeme oder Systeme mit Neuronalen Netzen. Sogar Hybridsysteme, die mehrere Ansätze integrieren, werden verwandt.

Ziel dieser Arbeit ist es, ein bestehendes Prognosesystem, welches statistische und Neuronale Ansätze enthält, in seiner Funktionalität zu erweitern und mit den damit gewonnenen Möglichkeiten Zeitreihen effektiver und effizienter prognostizieren und auswerten zu können. Diese Erweiterung geschieht mittels des *Stuttgarter Neuronale Netze Simulators*, kurz *SNNS*, der in das Prognosesystem eingebunden wird. Durch die Anwendung des Prognosesystems auf reale Daten — hier die Abverkaufszeitreihen der Produktgruppe der Cerealien einer Supermarkthandelskette — können die zur Verfügung stehenden Werkzeuge in ihrer Wirkung auf das Ergebnisverhalten untersucht werden.

In Erwartung guter Prognoseergebnisse bleibt trotz aller Zweifel von Poincaré zu sagen, daß es die große Stärke der Naturwissenschaften ist, Ursache und Wirkung in Zusammenhang zu bringen.

Es folgt eine Beschreibung der Kapitel:

In Kapitel 2 wird das in der *Arbeitsgruppe Praktische Informatik* an der Universität Osnabrück entwickelte Prognosesystem vorgestellt und erklärt.

In Kapitel 3 wird ein Simulationstool für Neuronale Netze vorgestellt, das in das Prognosesystem integriert wird. Es ist der an der Universität Stuttgart enwickelte *Stuttgarter Neuronale Netze Simulator*, kurz *SNNS*.

In Kapitel 4 wird dargestellt, wie *SNNS* in das Prognosesystem integriert wird.

In Kapitel 5 werden zwei wichtige Systemdateien beschrieben. Zum einen eine Datei, mit der Prognosen gesteuert werden können, zum anderen ein aus einem Prognoselauf resultierendes Log-File.

In Kapitel 6 geht es um das Verfahren der Kreuzvalidierung, mit dessen Hilfe günstige Netztopologien gefunden werden können, auch wenn nur wenige Trainingsdaten zur Verfügung stehen.

In Kapitel 7 wird eine Form der Sensitivitätsanalyse für Neuronale Netze beschrieben, die es erlaubt, die Signifikanz der Inputneuronen in bezug auf die Netzausgabe zu bestimmen.

In Kapitel 8 wird gezeigt, wie die in *SNNS* implementierten Pruning-Verfahren dem Prognosesystem zugänglich gemacht werden. Mit ihrer Hilfe können optimale Netztopologien durch Ausdünnung des Netzes gefunden werden.

In Kapitel 9 wird ein modifizierter Backpropagation-Algorithmus eingesetzt, um bessere Ergebnisse bei ökonometrischen Zeitreihen zu erhalten. Dazu werden Lernfunktionen innerhalb von *SNNS* geändert.

In Kapitel 10 wird ein Verfahren vorgestellt, mit dem die Inputpattern in Klassen zerlegt werden. An Hand der Klasseninformation wird eine Filterung der Pattern vorgenommen, damit störende Pattern entfernt werden.

In Kapitel 11 werden die in den Prognoseläufen verwendeten Netztopologien im einzelnen beschrieben.

In Kapitel 12 wird ein Tool zur Auswertung großer Datenmengen, wie sie insbesondere innerhalb der Abverkaufsprognose entstehen, beschrieben. Die Auswertung erfolgt auf Basis verschiedener Fehlermaße, die Aussagen über die Güte einer Prognose liefern.

In Kapitel 13 werden die durchgeführten Prognoseläufe mit ihren Parametereinstellungen beschrieben und die Ergebnisse diskutiert.

Kapitel 14 schließt die Diplomarbeit mit einer Zusammenfassung.

Kapitel 2

Das Prognosesystem

In der Arbeitsgruppe *Praktische Informatik* an der Universität Osnabrück wird unter der Leitung von Dipl.-Math. Frank M. Thiesing seit 1994 ein System zur Prognose von Zeitreihen entwickelt. Integriert und automatisiert sind verschiedene Vorverarbeitungs- sowie neuronale und statistische Prognosemethoden. Ein intelligentes Ablagesystem gestattet die Verwaltung sehr großer Zeitreihenmengen und deren Prognosen. Durch den nicht-spezialisierten Anwendungsbereich beschränkt sich die Arbeit zum Einbinden neuer Prognoseprojekte auf das Bereitstellen einer Konvertierungsprozedur von den Zeitreihenrohdaten zu den vom Prognosesystem lesbaren Zeitreihen. Eine leistungsstarke graphische Oberfläche läßt die Visualisierung der Zeitreihen und Prognosen zu. Prognosen können aus der Oberfläche oder auf Shell-Ebene gestartet werden. Eine Menge an Analyse- und Auswertungstools runden das Prognosesystem ab. Es ist auf verschiedenen Plattformen einsatzfähig, wie unter *UNIX* und seinen Derivaten (*SunOS*, *Solaris*, *HP-UX*, *Linux*).
Es sei nochmals angemerkt, daß sich das System durch Verwaltung sehr vieler Prognosen gut dazu eignet, projektspezifisch günstige Parametereinstellungen zu finden und von anderen abzugrenzen.
Ist im weiteren Verlauf der Arbeit von *Prognosesystem* die Rede, so ist damit das gesamte System gemeint, während *Simulations-* und *Prognosetool* den Teil des Prognosesystems meinen, der die Prognose durchführt, wie etwa *SNNS*, *BP* oder *GAUSS*.

2.1 Funktionsgruppen

Die Zeitreihenrohdaten werden in einem Vorverarbeitungsschritt in für das System lesbare Zeitreihen umgewandelt und auf Dateiebene abgelegt. Zusätzlich werden die Zeitreihen noch skaliert abgelegt. Dies geschieht noch statisch, wird aber im Rahmen der Diplomarbeit von Carsten Wilhelm dynamisch implementiert. Um Prognosen mit den verschiedenen Simulationstools ausführen zu können, bedarf es wegen der großen Menge an einstellbaren Parametern einer Datei, in der die Einstellungen für einen Prognoselauf zusammengefaßt werden. Dies ist das sogenannte *BP-File*. Wird eine Prognose mit einem der Prognosetools gestartet, so werden die benötigten

Zeitreihen und die Einstellungen aus dem BP-File benutzt. Neben der Prognosezeitreihe, die auf Dateiebene abgelegt wird, können noch Dateien mit weiterführenden Ergebnissen über die Prognose generiert worden sein, zumindest aber ein zusätzliches *Log-File*, welches Daten über den Prognoseverlauf beinhaltet. Alle generierten Dateien können Gegenstand einer Auswertung sein, die der Benutzer zu einem beliebigen Zeitpunkt nach dem Prognoselauf vornimmt. Solche Auswertungen können auch übergreifend sein und sich nicht nur auf eine Zeitreihe beziehen. Die Funktionsgruppen werden schematisch in Abbildung 2.1 dargestellt.

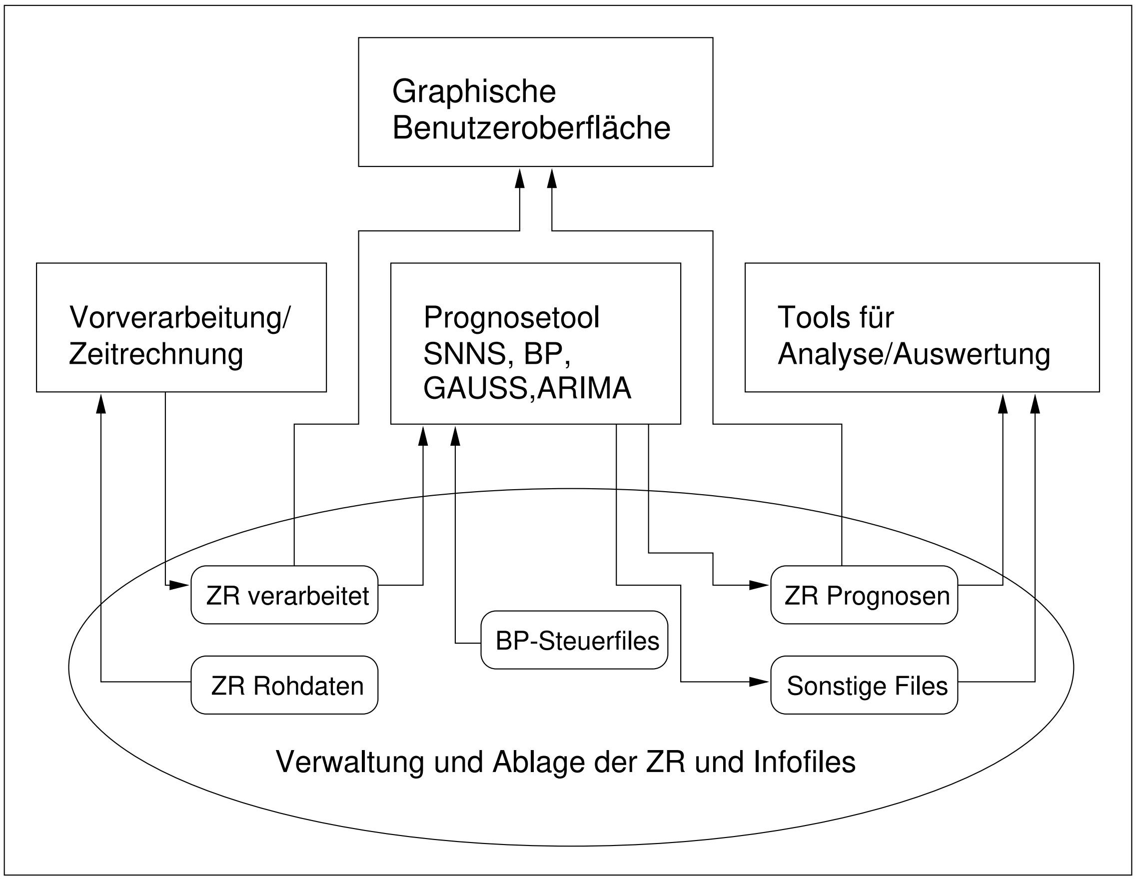


Abbildung 2.1: Funktionsgruppen des Prognosesystems

2.2 Das Prognosesystem auf Dateiebene

Die Abbildung 2.2 zeigt das Prognosesystem auf Dateiebene. Um den im Abschnitt 2.3 erläuterten Verlauf einer Prognose verstehen zu können, müssen erst einige Strukturen auf Dateiebene, vornehmlich das Ablagesystem, erklärt werden.
Zum Systemstart wird ein Shell-Skript namens `.projektrc` abgearbeitet, das im wesentlichen

Pfade und Environmentvariablen setzt. Im Verzeichnis `Tcl` befindet sich das Programm der graphischen Benutzeroberfläche [Kro97], welches mit dem Kommando `draw_cmd` aufgerufen wird. Im Verzeichnis `SNNS` liegt eines der verwendeten Prognosetools, hier ist es der *Stuttgarter Neuronale Netze Simulator*. Das übrige System mitsamt allen projektspezifischen Zeitreihen beginnt unter dem Wurzelverzeichnis `Projekte`. Der größte Teil des Programmcodes, der in den Sprachen *C*, *Perl* und durch einige *Shell-Skripte* realisiert wird, liegt dort in den `Bin`- und `Include`-Verzeichnissen. Auf derselben Ebene befindet sich das Verzeichnis, in dem globale, also projektunabhängige Zeitreihen zu finden sind wie etwa Feiertage in Abhängigkeit der Bundesländer oder Hilfszeitreihen, die eine Sinus- oder Kosinusschwingung mit Jahressaisonalität beinhalten. Die Ablagestruktur der Zeitreihendaten entspricht der, die im folgenden exemplarisch an einem bestimmten Projekt erläutert wird.
Ein einzelnes Projekt kann mehrere Ebenen von Unterprojekten enthalten. Als Beispiel möge eine Supermarktkette (erste Projektkomponente) dienen, die mehrere Filialen (jeweils eine Filiale entspricht der zweiten Projektkomponente) besitzt. Innerhalb einer Filiale sollen Artikelabverkäufe prognostiziert werden (jeweils ein Artikel entspricht der dritten Projektkomponente). In der Abbildung 2.2 findet sich diese Aufteilung für den allgemeinen Fall unter `Proj1_komp1` bis hin zu `Proj1_kompn` wieder. Im Beispiel ist `n` gleich 3, da es drei Projektebenen gibt.
Spezieller projektbezogener Programmcode findet sich in den `Bin`- und `Include`-Verzeichnissen unter einem Hauptprojektverzeichnis wieder. Dort ist auf jeden Fall die Schnittstelle von den Zeitreihenrohdaten zu den vom System lesbaren Zeitreihen eingebunden. In einem sogenannten `MODELL`-File werden mögliche Skalierungen angegeben, deren zugehörige Zeitreihen ebenfalls erzeugt und abgelegt werden. Die Ablage der Zeitreihen erfolgt unterhalb des letzten Projektkomponentenverzeichnisses in den Verzeichnissen `ZR/Head` und `ZR/Body`. Dort sind auch die Prognosezeitreihen zu finden. Andere zu einer Prognosezeitreihe gehörigen Dateien, wie etwa Log-File und ASCII-Netzdatei, finden sich unter dem Verzeichnis `Prognosen`, dessen weitere Unterebenen analog zu den Projektkomponenten angelegt sind. Um nun Prognosezeitreihen und die dazugehörigen Dateien in Einklang zu bringen, gibt es einen sogenannten, vom Benutzer zu vergebenden, eindeutigen `comment`. Einmal kommt dieser `comment` als Verzeichnis vor, einmal dient er als Eintrag innerhalb eines `INDEX`-Files zur Identifizierung der entsprechenden Prognosezeitreihe. Um also eine eindeutige Ablage aller in einem Prognoselauf anfallenden Dateien zu gewährleisten, ist die Angabe aller Projektkomponenten sowie die Angabe des Comments nötig.

2.3 Allgemeiner Ablauf einer Prognose

Ein Benutzer des Systems muß von der Dateistruktur nichts wissen, will er Prognosen und Auswertungen durchführen. Zum tiefergehenden Verständnis des Prognoseablaufs ist dies aber sicherlich nötig. Im folgenden wird nicht auf Verzeichnisse eingegangen, von denen oder in die Daten geholt oder geschrieben werden, da im vorherigen Abschnitt die betreffenden Verzeichnisnamen beschrieben wurden. Es wird davon ausgegangen, daß die Zeitreihen eines Projektes bereits skaliert abgelegt und alle Parameter in einem BP-File eingestellt worden sind.

Der Kommandozeilenaufruf[1] eines Prognoselaufes lautet

```
training.pl BP|SNNS <BP-File> <Comment> <Anzahl Projektebenen
            <Komponente 1> ... <Komponente n>.
```

Mit `BP` oder `SNNS` kann eines der zwei im System verwendeten neuronalen Simulationstools zum Einsatz kommen. Des weiteren werden das `BP-File` mit allen relevanten Einstellungen angegeben sowie der `Comment` und die `Projektkomponenten`. Der weitere Verlauf wird auch in Abbildung 2.3 wiedergegeben. Der Aufruf kann aus einem beliebigen Verzeichnis erfolgen, da alle weiteren Dateien nach Wechsel in das Zielverzeichnis in eben dieses abgelegt werden. Über ein weiteres Skript (`fe2bp.pl`) werden die für das Simulationstool wichtigen Dateien generiert. Für neuronale Verfahren relevant sind Pattern- und Netzdateien. Dann wird das Simulationstool aufgerufen, das neben der Prognosezeitreihe noch weitere Dateien (z.B. Log-File) schreibt. In einem letzten Schritt (`create_prognose.pl`) wird noch die Prognosezeitreihe aus dem Log-File in ihr Ablageverzeichnis geschrieben. Werden mehrere Trainingsaufrufe in einem Shell-Skript zusammengefaßt, so können auf einfache Art viele Prognosen durchgeführt werden.

[1] Dies ist der Aufruf für die Prognose mit Neuronalen Netzen. Entsprechend gibt es z.B. für die Regression ein Skript namens `regression.pl`.

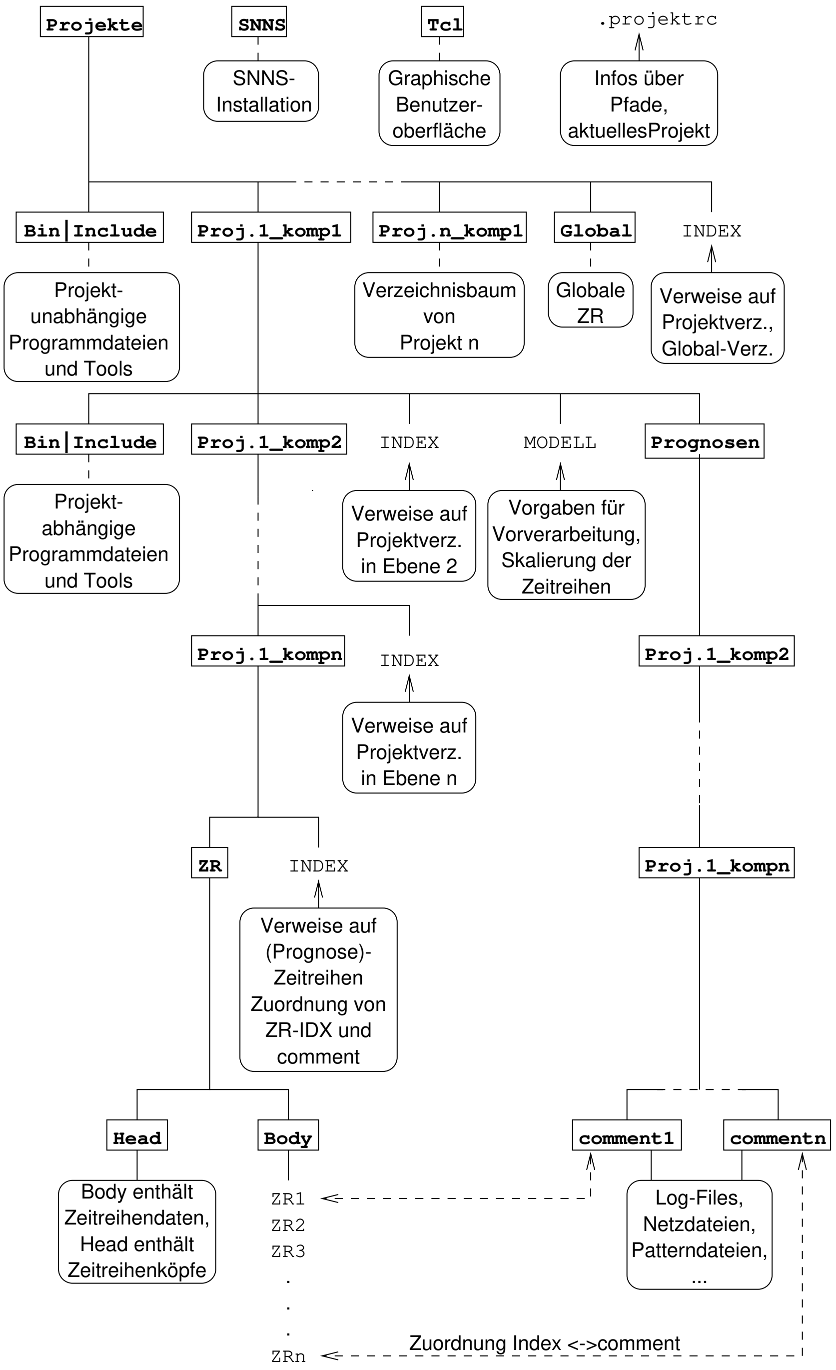


Abbildung 2.2: Dateistruktur des Prognosesystems

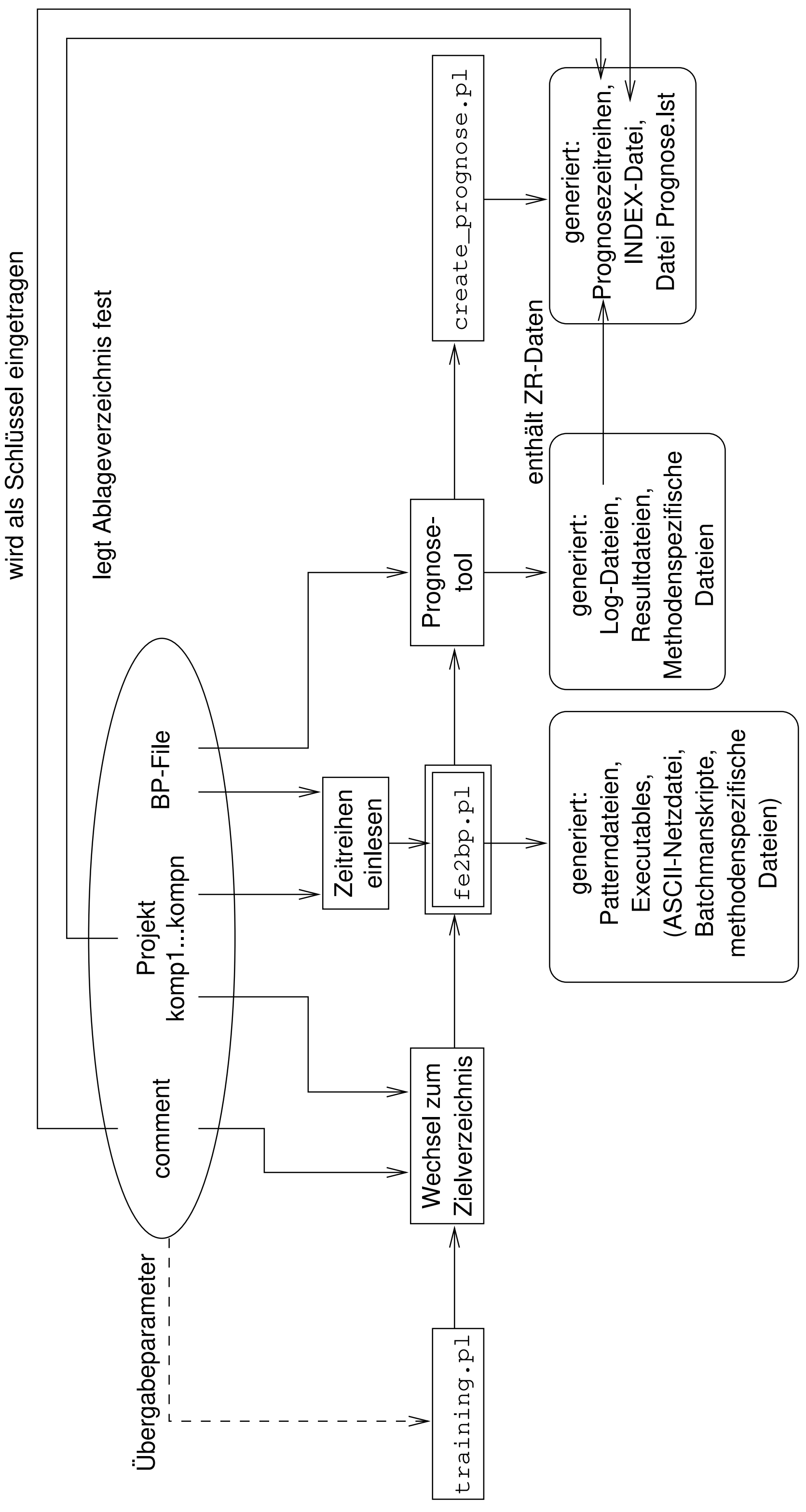


Abbildung 2.3: Struktur einer Prognose

Kapitel 3

Der Stuttgarter Neuronale Netze Simulator

SNNS (Stuttgarter Neuronale Netze Simulator) ist ein Simulator für Neuronale Netze, der am Institut für Parallele und Verteilte Höchstleistungsrechner (IPVR) an der Universität Stuttgart in der Arbeitsgruppe von Andreas Zell seit 1989 entwickelt und gepflegt wird. Das Ziel des Projektes ist es, eine effiziente und flexible Entwicklungsumgebung zur Untersuchung Neuronaler Netze zu schaffen [Zel94].
SNNS besteht aus vier Hauptkomponenten (siehe Abbildung 3.1), dem Simulatorkern, der graphischen Benutzerschnittstelle *XGUI*, dem Interpreter *Batchman* zum Trainieren von Netzen im Batch-Betrieb und dem Netzwerkcompiler *snns2c*.

Der Simulatorkern arbeitet auf einer internen Repräsentation der Neuronalen Netze und führt mit ihnen verschiedenste Aktionen aus. Er ist durch eine Netzwerkbeschreibungsdatei mit dem Netzwerkcompiler verbunden und durch eine Funktionsschnittstelle mit der graphischen Benutzeroberfläche. Diese Oberfläche, auch *XGUI* genannt, die auf dem Simulatorkern aufsetzt, stellt eine graphische Repräsentation der Neuronalen Netze zur Verfügung und läßt Kontrolle über den Simulatorkern während eines Simulationslaufes zu. Zusätzlich kann die Schnittstelle dazu benutzt werden, Netzwerke direkt zu erstellen, zu manipulieren und auf viele Arten zu visualisieren. SNNS wurde vollständig in ANSI-C implementiert und auf verschiedenen Plattformen getestet. XGUI basiert auf X11 Release 5 von MIT und läuft auch unter X11R6.
Zu dem vollständigen SNNS-Paket gehören noch einige Tools, die unter anderem der Ergebnisanalyse, der Netz- und Patterngenerierung dienen, um den Umgang mit SNNS zu vereinfachen. Auf den Interpreter *Batchman* wird später noch detailliert eingegangen.

3.1 Relevante Komponenten im Prognosesystem

Die innerhalb des Prognosesystems relevanten Komponenten von SNNS zeigt Abbildung 3.2.

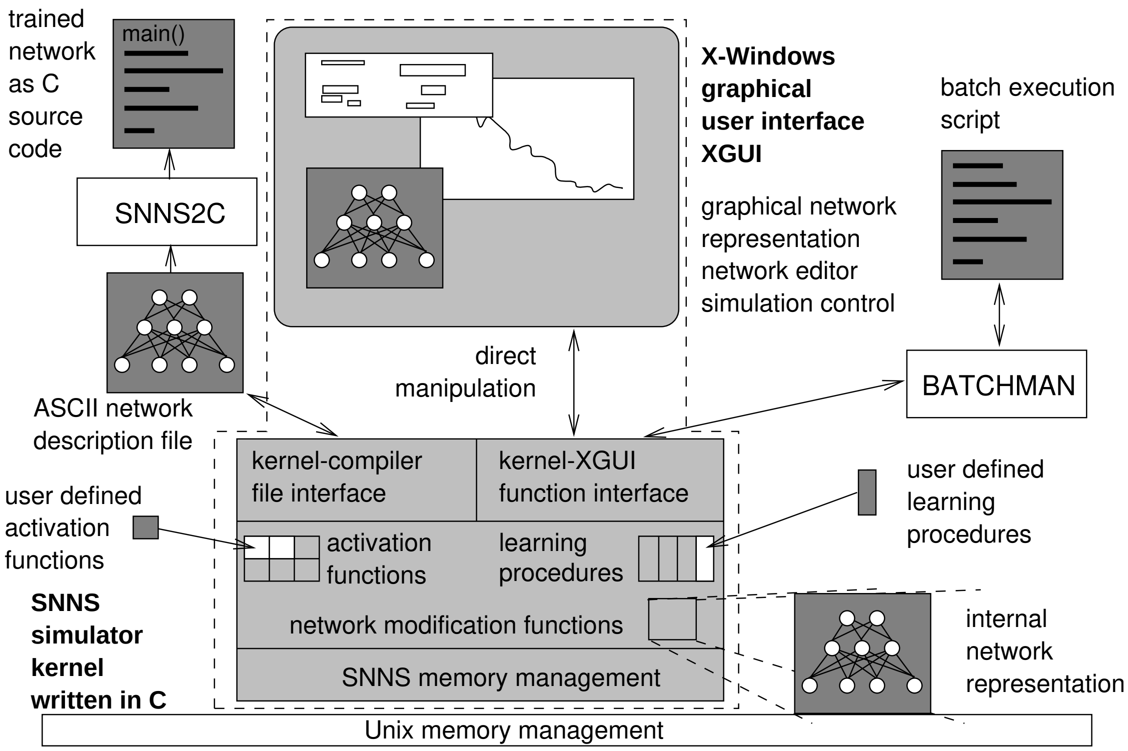


Abbildung 3.1: Hauptkomponenten von SNNS

Diese Komponenten gehören zur sogenannten *Batch*-Version von SNNS, die nur aus dem Simulatorkern und dem Interpreter Batchman besteht. Dabei werden im Batch-Betrieb die Netzdatei geladen, Patterndateien übergeben und Ergebnisdateien geschrieben. Der Batch-Betrieb wird günstigerweise dazu verwendet, um empirisch optimale Netzwerkgrößen und Parameterwerte bei Lernverfahren für Zeitreihen innerhalb des Prognosesystems zu finden. Das noch in der Abbildung 3.2 dargestellte zusätzliche Tool *mkhead* ist für die Generierung eines Dateikopfes für Patterndateien zuständig. Der größte Aufwand bei Verwendung der zu SNNS gehörigen Komponenten liegt in der Ablaufsteuerung der Batchdateien für den Interpreter Batchman[1]. Zusätzlich wird von der Möglichkeit Gebrauch gemacht, eigene Lernverfahren in den Simulatorkern einzubinden (siehe Kapitel 9).

3.2 Der Interpreter Batchman

Um Aktionen wie das Training eines Neuronalen Netzes im Hintergrund ausführen zu können, gibt es in SNNS den Interpreter Batchman, der eine Schnittstelle zum Simulatorkern darstellt. Er wird in einer eigenen Batch-Sprache programmiert, die neben Training und Testen eines Netzes

[1]Das Einbinden von SNNS in das Prognosesystem und die dazu gehörigen Programme sind in einem eigenen Kapitel beschrieben.

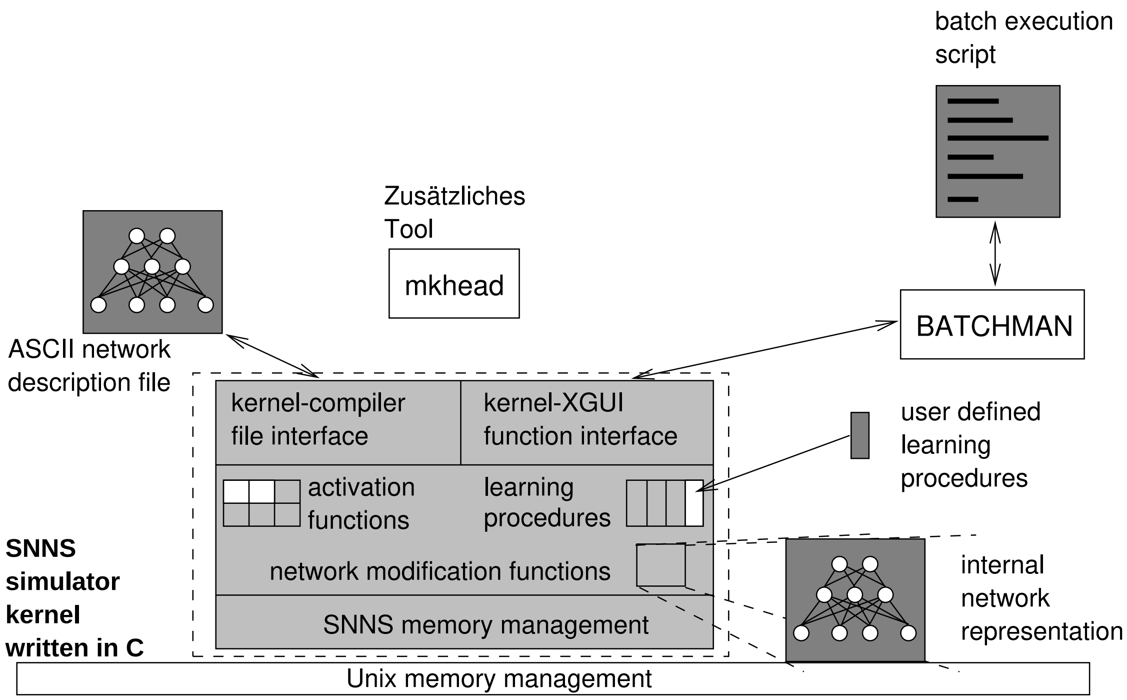


Abbildung 3.2: Relevante Komponenten im Prognosesystem

auch alle nicht-graphischen Funktionen, die im XGUI erreichbar sind, zur Verfügung stellt. Die Batch-Sprache wurde Sprachen wie *AWK*, *Pascal*, *Modula2* und *C* nachempfunden. Die Sprache kennt vier Datentypen,

- Integerwerte,
- Floatwerte,
- die bool'schen Werte `TRUE` und `FALSE`,
- Strings.

Variablen, die einen dieser Datentypen enthalten, werden nicht deklariert, sondern stehen sofort zur Verfügung. Es gibt einige Systemvariablen, von denen zwei in den meisten Skripten vorkommen. Zum einen die Variable `MSE`, die den Mean Square Error über alle Netzoutputs enthält, zum anderen `CYCLES`, die die Anzahl der schon trainierten Zyklen enthält. Obligatorisch ist das Verwenden von Ausdrücken mittels verschiedener Operatoren, wie es auch bei anderen Hochsprachen üblich ist. Folgende Kontrollstrukturen sind in der Sprache möglich:

- `if AUSDRUCK then BLOCK endif`,
- `if AUSDRUCK then BLOCK else BLOCK endif`,

- `for ZUWEISUNG to AUSDRUCK do BLOCK endfor`,
- `while AUSDRUCK do BLOCK endwhile`,
- `repeat BLOCK until AUSDRUCK`.

Um Ausgaben auf dem Bildschirm zu generieren oder in eine Datei zu schreiben, existiert eine `print`-Anweisung.
Einen Großteil der Sprache machen SNNS-Funktionsaufrufe aus, die sich direkt auf Funktionen des Simulatorkerns beziehen. Sie lassen sich in vier Gruppen aufteilen. Diese sind Funktionen, die SNNS-Parameter setzen, wie Lern- und Updatefunktionen, Funktionen, die sich auf Neuronale Netze beziehen, wie das Laden einer ASCII-Netzdatei oder das Trainieren eines Netzes, Funktionen, die sich auf Pattern beziehen, wie das Laden oder das Zuweisen von Patterndateien zu einem Netz und Funktionen, die spezielle Aufgaben übernehmen, wie Pruning von Netzen oder das Ausführen von Shellkommandos.
Die im einzelnen innerhalb des Prognosesystems verwendeten Funktionen sind in Tabelle 3.1 aufgelistet. Es sind leider undokumentierte Funktionen für das Pruning von Netzen nötig, da die zur Gruppe der speziellen Funktionen gehörige Funktion `pruneNet()` nicht die kleinste Funktionseinheit darstellt, sondern einen vollständigen Ablauf beinhaltet, wie er eigentlich in einem Batch-Skript hätte implementiert werden sollen und damit frei konfigurierbar wäre. Die undokumentierten Funktionen beinhalten solche kleinsten Funktionseinheiten. Zu finden sind sie in der Datei `ic_snns.c` im Verzeichnis `tools/sources` des SNNS-Installationsverzeichnisses, in der im übrigen alle vom Simulatorkern für Batchman bereitgestellten Funktionen zu finden sind.
Es sei noch erwähnt, daß Batchman in einem interaktiven Modus aufgerufen werden kann, in dem er Benutzereingaben auf einer eigenen Kommandozeile erwartet, oder mit einem Batch-Skript, das ein vollständiges Programm enthält. Letzteres wird in dem Prognosesystem verwendet.

Tabelle 3.1: SNNS-Funktionsaufrufe in Batchman-Skripten

Funktionen, die SNNS-Parameter setzen		
Funktion	**Erläuterung**	**Beispiel**
setInitFunc()	Setzt Initialisierungsfunktion des Netzes	setInitFunc ("Randomize_Weights", 1.0, -1.0)
setLearnFunc()	Setzt Lernverfahren des Netzes	setLearnFunc ("Std_Backpropagation", 0.1, 0.0)
setUpdateFunc()	Setzt Updatefunktion des Netzes	setUpdateFunc ("Topological_Order")
setPruningFunc()	Setzt Pruning-Algorithmus für das Netzwerk (max. 9 Parameter)	setPruningFunc ("Skeletonization", "Rprop", ...)
setShuffle()	Mischen von Pattern erlauben	setShuffle(TRUE)
Funktionen, die sich auf Neuronale Netze beziehen		
Funktion	**Erläuterung**	**Beispiel**
loadNet()	Laden einer ASCII-Netzdatei	loadNet("filename.net")
saveNet()	Speichern einer ASCII-Netzdatei	saveNet("filename.net")
saveResult()	Speichern einer Ergebnisdatei	saveResult("ergebnis.res")
initNet()	Initialisiert das Netz mit Initialisierungsfunktion	initNet()
trainNet()	Durchführen eines Trainingszyklus	trainNet()
testNet()	Abfragen des Netzes	testNet()
Funktionen, die sich auf Pattern beziehen		
Funktion	**Erläuterung**	**Beispiel**
loadPattern()	Laden einer Patterndatei	loadPattern("pattern.pat")
setPattern()	Setzen einer Patterndatei für aktuelles Netz	setPattern("pattern.pat")
Funktionen, die spezielle Aufgaben übernehmen		
Funktion	**Erläuterung**	**Beispiel**
execute()	Führt Shellkommando aus	execute("echo hallo")
Undokumentierte Funktionen		
Funktion	**Erläuterung**	**Beispiel**
pruneNetNow()	Führt einen Pruning-Schritt aus	pruneNetNow()
pruneTrainNet()	Führt normales Training aus (anstelle von trainNet() zu verwenden, wenn setPruningFunc() aufgerufen wurde)	pruneTrainNet()

Kapitel 4

SNNS im Prognosesystem

Wegen seines großen Funktionsumfangs und seiner flexiblen Steuerung über Batchman-Skripte, wurde *SNNS* als weiteres neuronales Simulationstool neben dem *BP*-Tool in das Prognosesystem integriert. Dabei mußten Eigenheiten wie die ASCII-Netzdatei und Batchman-Skripte berücksichtigt werden im Gegensatz zum BP-Tool, das seine Netze und Ablaufsteuerung intern zur Laufzeit aufbaut. *SNNS* liefert eine große Menge an verschiedenen Lernfunktionen, unter anderem auch Pruning-Algorithmen (siehe Kapitel 8). Durch eine einfache Netzbeschreibungsdatei können zudem verschiedenartige Topologien verwirklicht werden, unter anderem auch rekurrente Netztopologien. Außerdem sind die in *SNNS* implementierten Lernverfahren leicht den jeweiligen Ansprüchen anpaßbar, wie der Einbau eigener Fehlerfunktionen zeigt (siehe Kapitel 9).
SNNS verlangt zum automatischen Ablauf eine Netzdatei und ein Batchman-Skript, welche vor dem eigentlichen Prognoselauf erst generiert werden müssen. Das Generieren dieser Dateien und den speziellen Ablauf einer *SNNS*-Prognose wird im folgenden abgehandelt.

4.1 Die ASCII-Netzdatei und der Netzgenerator

Ein zentraler Bestandteil des Tools *SNNS* ist seine ASCII-Beschreibungsdatei eines Neuronalen Netzes. Die über den *Simulator-Kernel* erstellte interne Repräsentation einer beliebigen Netztopologie hat auf Dateiebene eben diese ASCII-Datei als Entsprechung. Im *File-Interface*-Modul des Kernels findet durch einen Compiler die Umwandlung in beide Richtungen statt[1].
Um dem Tool *SNNS* eine beliebige Netztopologie zugänglich zu machen, gibt es zwei Möglichkeiten. Zum einen kann ein Netz manuell mit Hilfe der graphischen Benutzeroberfläche *XGUI* erstellt werden, zum anderen kann eine bestehende Netzdatei in den Kernel geladen werden. Da das Prognosesystem ein automatisches System ist und viele verschiedene Netztopologien einen hohen Zeitaufwand bei der manuellen Erstellung bedeuten, ist nur die letztgenannte Möglichkeit sinnvoll. Die Netzdatei, die zur Verfügung stehen muß, wird über ein *Perl*-Skript

[1] Über ein mitgeliefertes Tool (*snns2c*) kann darüber hinaus eine Umwandlung der ASCII-Netzdatei in C-Source Code erfolgen.

namens `snns_create_net.pl` generiert. Die Netztopologie kann durch Übergabeparameter beliebig gewählt werden im Rahmen der für Zeitreihenprognosen sinnvollen Topologien. Folgende Anforderungen werden an den Netzgenerator gestellt:

- Erzeugung von Feed-Forward-, Jordan-, Elman- oder Kohonen-Netzen,
- Neuronenzahl für jede Schicht frei wählbar,
- Vollvernetzung oder Einfachvernetzung zwischen je zwei benachbarten Schichten,
- optionale Shortcut-Verbindungen zwischen je zwei nicht benachbarten Schichten,
- Aktivierungs- und Ausgabefunktion für jede Schicht frei wählbar,
- übersichtliche Darstellung des Netzes in der graphischen Benutzeroberfläche von *SNNS*.

Der Netzgenerator besitzt folgenden Aufruf, wobei die Flags mit den einzelnen Parametern in beliebiger Reihenfolge erscheinen können:

```
snns_create_net.pl -f|j|e|k <Topologie> -n <Filename>
[-s <Shortcuts> -a <Aktivierungsfkt.> -o <Outputfkt.> -c <Out_Context>]
```

Welcher Art die erzeugte Topologie ist, wird durch das erste Flag festgelegt. So steht `f` für eine Feed-Forward-, `j` für eine Jordan-, `e` für eine Elman- und `k` für eine Kohonen-Topologie. Die Neuronenzahlen der einzelnen Schichten und die Vernetzung der Neuronen werden im Parameter *Topologie* übergeben. Die Anzahlen für jede Schicht sind Integerwerte, die Art der Vernetzung zwischen zwei Schichten entweder ein „`.`“ für eine Einfachvernetzung oder ein „`:`“ für eine Vollvernetzung. Natürlich können nur solche Schichten einfach miteinander verbunden werden, die die gleiche Anzahl Neuronen aufweisen. Beispielsweise wird ein dreischichtiges Feed-Forward-Netz mit 5 Input-, 3 Hidden und 1 Outputneuron durch `-f 5:3:1` identifiziert.
Bei Jordan- und Elman-Netzen werden die *Kontextschichten* und deren Verbindungen automatisch eingerichtet. Im speziellen kann bei Elman-Netzen entschieden werden, ob es eine Kontextschicht für die Outputschicht gibt (`-c 1`) oder nicht (`-c 0`). Bei Kohonen-Netzen ist zu beachten, daß die Neuronenzahlen in jeder Schicht identisch sein müssen, damit sich eine rechteckige Topologie ergibt. Der Dateiname, unter dem die Netzbeschreibung abgelegt wird, enthält der Parameter *Filename*. Die folgenden Parameter sind optional. Werden sie nicht angegeben, gibt es keine Shortcuts, als Aktivierungsfunktion wird die logistische Funktion und als Ausgabefunktion wird die Identität angenommen.
Als Kürzel für die Schichten werden im weiteren `I` für die Input-, `H`*n* für die *n*-te Hiddenschicht und `O` für die Outputschicht verwendet. Der Parameter *Shortcuts* enthält jeweils durch ein Leerzeichen getrennt die Angabe zweier nicht benachbarter Schichten, die durch ein „-“-Zeichen verbunden sind. So können mehrere Shortcut-Verbindungen in einem Netz angegeben werden (z.B. `-s I-O H1-H3`). Die Syntax der Parameter *Aktivierungsfkt.* und *Outputfkt.* ist identisch. Jeweils durch ein Leerzeichen getrennt erfolgt die Angabe einer Schicht, dem „`:`“-Zeichen und dem Namen einer in *SNNS* implementierten Aktivierungs- oder Ausgabefunktion

(z.B. `-a H1:Act_TanH O:Act_Identity`).
Eine generierte Netzdatei und die Darstellung in der graphischen Oberfläche (siehe Abbildung 4.1) von *SNNS* wird am Aufrufbeispiel

```
snns_create_net.pl -f 3.3:2:1 -n ff -s H1-O -a O:Act_Identity
```

dargestellt. Durch diesen Aufruf wird ein vierschichtiges Netz mit einer einfachen Verbindung von der Input- zur ersten Hiddenschicht erstellt. Es besitzt Shortcut-Verbindungen und die Identität als Aktivierungsfunktion in der Outputschicht. Die Darstellung der Shortcutverbindungen von Neuron 4 nach Neuron 9 und von Neuron 6 nach Neuron 9 ist durch die Anordnung der Neuronen verdeckt. Das Netz wird unter dem Dateinamen `ff.net`, deren Inhalt im folgenden erläutert wird, abgespeichert.

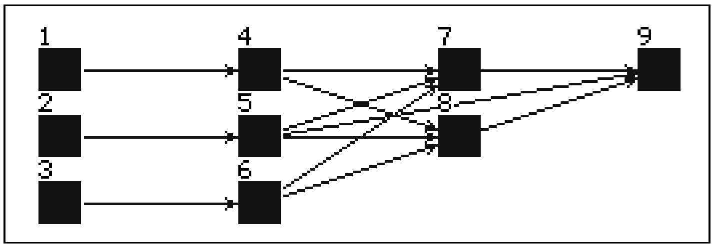


Abbildung 4.1: Darstellung eines Netzes in der graphischen Oberfläche von *SNNS*

Die Netzdatei teilt sich in vier Abschnitte. Im *Header* sind unter anderem der Dateiname, die Gesamtneuronenzahl und die Anzahl der Verbindungen zu finden. Der Inhalt des Abschnitts *unit default section* ist fest im Netzgenerator enthalten und wird nie geändert. Der Abschnitt *unit definition section* enthält eine Beschreibung jedes Neurons, unter anderem bestehend aus der aktuellen Aktivierung und dem aktuellen Schwellwert, die aber für die Netzgenerierung keine Rolle spielen, dem Neuronentyp (zu finden in der Spalte *st*), seiner Position in der graphischen Oberfläche und Angaben über Aktivierungs- und Ausgabefunktion. Im Abschnitt *connection definition section* wird in der Spalte *target* jedes Neuron aufgeführt, zu dem eine Verbindung von den in der Spalte *source* angegebenen Neuronen besteht.

```
SNNS network definition file V1.4-3D
generated at Tue Mar 24 18:35:56 1998

network name : ff
source files :
no. of units : 9
no. of connections : 14
no. of unit types : 0
no. of site types : 0


learning function : Std_Backpropagation
```

```
update function   : Topological_Order


unit default section :

act      | bias     | st | subnet | layer | act func     | out func
---------|----------|----|--------|-------|--------------|-------------
 0.00000 |  0.00000 | h  |      0 |     1 | Act_Logistic | Out_Identity
---------|----------|----|--------|-------|--------------|-------------


unit definition section :

no. | typeName | unitName | act      | bias     | st | position   | act func | out func |
----|----------|----------|----------|----------|----|------------|----------|----------|
  1 |          |          |  0.00000 |  0.00000 | i  |  2,  2, 19 |  |  |
  2 |          |          |  0.00000 |  0.00000 | i  |  2,  3, 19 |  |  |
  3 |          |          |  0.00000 |  0.00000 | i  |  2,  4, 19 |  |  |
  4 |          |          |  0.00000 |  0.00000 | h  |  5,  2, 19 |  |  |
  5 |          |          |  0.00000 |  0.00000 | h  |  5,  3, 19 |  |  |
  6 |          |          |  0.00000 |  0.00000 | h  |  5,  4, 19 |  |  |
  7 |          |          |  0.00000 |  0.00000 | h  |  8,  2, 19 |  |  |
  8 |          |          |  0.00000 |  0.00000 | h  |  8,  3, 19 |  |  |
  9 |          |          |  0.00000 |  0.00000 | o  | 11,  2, 19 | Act_Identity |  |
----|----------|----------|----------|----------|----|------------|----------|----------|


connection definition section:

target | site | source:weight
-------|------|---------------------------------------------------------------------
     4 |      |  1: 0.00000
     5 |      |  2: 0.00000
     6 |      |  3: 0.00000
     7 |      |  6: 0.00000,  5: 0.00000,  4: 0.00000
     8 |      |  6: 0.00000,  5: 0.00000,  4: 0.00000
     9 |      |  8: 0.00000,  7: 0.00000,  6: 0.00000,  5: 0.00000,  4: 0.00000
-------|------|---------------------------------------------------------------------
```

Datei `ff.net`

4.2 Batchman Templates

Zum automatischen Ablauf einer Prognose mit *SNNS* wird ein Batchman-Skript benötigt. Da dieses Skript unter anderem mit unterschiedlichen Zeitreihennamen, Lernfunktionen und Parametern zurechtkommen muß, wurde ein Template geschrieben, aus dem das ausführbare

Skript erzeugt wird. Es liegt im projektunabhängigen `Include`-Verzeichnis mit dem Dateinamen `bman.tmpl`. Einen Ausschnitt aus diesem Template zeigt der folgende Source-Code.

```
loadNet("./$FNCOMPONENTS.net")
$$INCL_VARSET
setLearnFunc("$LEARNFUNC", $LEARNPARAM)
setInitFunc("$INITFUNC", $INITPARAM)
setUpdateFunc("$UPDATEFUNC")
MIN_CYCLES := $MINEPOCHEN
MAX_CYCLES := $EPOCHEN
ERR_CYCLES := $ERR
MAX_ERROR  := $MAXERROR

trn_pattern := "$FNCOMPONENTS.trn.pat"
tst_pattern := "$FNCOMPONENTS.tst.pat"
val_pattern := "$FNCOMPONENTS.val.pat"
```

Datei `bman.tmpl`: Ausschnitt

Die Strings, die mit einem „$" beginnen, sind Schlüsselwörter, die durch Parameterwerte aus einem BP-File ersetzt werden. Wörter, die mit einem doppelten „$"-Zeichen beginnen, werden durch kleine Teile ausführbaren Programmcodes ersetzt. Der oben gezeigte Ausschnitt kann beispielsweise nach dem Textersatz so aussehen:

```
loadNet("./Tegut.AAA.229104.net")
P1 := 0.2
P2 :=  0.5
P3 :=  0.0
P4 :=  0.0
P5 :=  10.0
P6 :=  0.12
P7 :=  0.12
setLearnFunc("BPM_ln_cosh", 0.2, 0.5, 0.0, 0.0, 10.0, 0.12, 0.12)
setInitFunc("Randomize_Weights", 0.01, -0.01)
setUpdateFunc("Topological_Order")
MIN_CYCLES := 40
MAX_CYCLES := 100
ERR_CYCLES := -4
MAX_ERROR  := 10

trn_pattern := "Tegut.AAA.229104.trn.pat"
tst_pattern := "Tegut.AAA.229104.tst.pat"
val_pattern := "Tegut.AAA.229104.val.pat"
```

Datei `bman.Tegut.AAA.229104.bat`: Ausschnitt

Solche Templates gibt es für mehrere im BP-File einstellbare Methoden, die stark vom Standardtemplate `bman.tmpl` abweichende Abläufe besitzen, so etwa für Pruning-Verfahren (`bmanPRUNE.tmpl`), Kreuzvalidierung (`bmanCROSS.tmpl`), Kohonen-Netze (`bmanSOM.tmpl`) und für die Sensitivitätsanalyse (`bmanSENSE.tmpl`). Die Abläufe der einzelnen Skripte werden in den entsprechenden Kapiteln abgehandelt. Im weiteren wird das Standardskript im Rahmen eines Standardprognoselaufes mit *SNNS* behandelt.

4.3 Die Standardprognose

Nachdem ein Prognoselauf mit `training.pl` gestartet und die Programmkontrolle an `fe2bp.pl` abgegeben worden ist, beginnt ein *SNNS*-spezifischer Teil (siehe dazu Abbildung 4.2). Es werden drei Patternsets generiert, eine Trainingsmenge, mit der das Neuronale Netz trainiert wird, eine Testmenge, mit der das Netz nach jeder Epoche innerhalb seines Trainings abgefragt wird, und eine Validierungsmenge, mit der die eigentliche Prognose durchgeführt wird. Außerdem steht die ASCII-Netzdatei und das von *SNNS* ausführbare Batchman-Skript zur Verfügung.
In einem weiteren Shell-Skript sind drei Kommandoaufrufe zusammengefaßt. Dieses wird von `fe2bp.pl` aufgerufen. Es beginnt mit der Ausführung des Batchman-Skripts (siehe Abbildung 4.3), welches ein vollständiges Training durchführt, das Netz am Ende mit den drei Patternmengen abfragt und diese Ergebnisse als Resultfiles speichert. Weitere Eckdaten werden zusätzlich in ein Log-File geschrieben (siehe Kapitel 5).
Da das Simulationstool *SNNS* zu einem Zeitpunkt in das Prognosesystem integriert wurde, in dem Strukturen an einigen Schnittstellen, wie sie z.B. das Log-file darstellt, fest vorgegeben waren, mußten aus Gründen der Kompatibilitätswahrung die Ausgabedateien von *SNNS* in ein dem System bekanntes Format umgewandelt werden. Dies geschieht in einem zweiten Schritt, in dem das Log-File und die Resultfiles in ein endgültiges Format gebracht werden, um dann von schon bestehenden Skripten im dritten Schritt dazu benutzt werden, die Zeitreihen korrekt abzulegen.

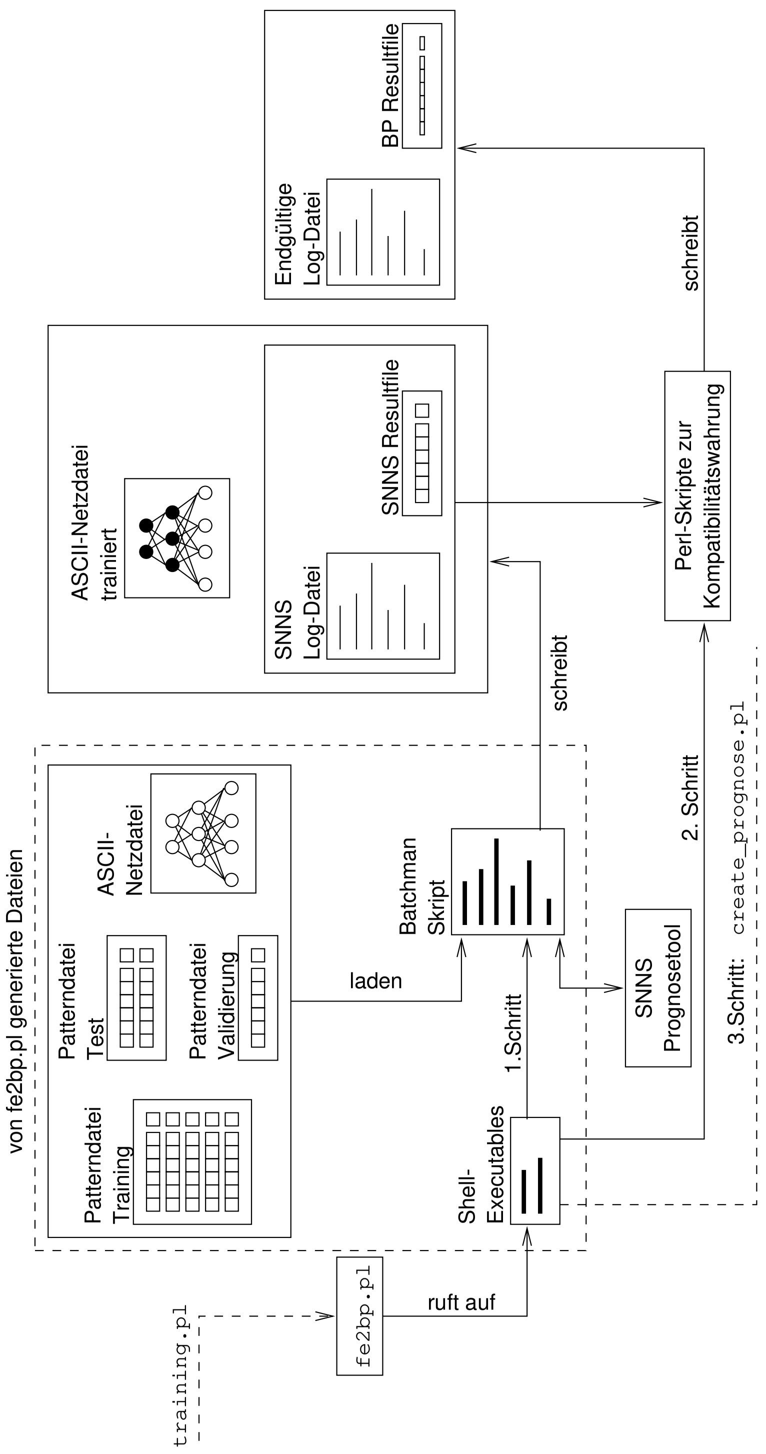


Abbildung 4.2: Prognose mit *SNNS*

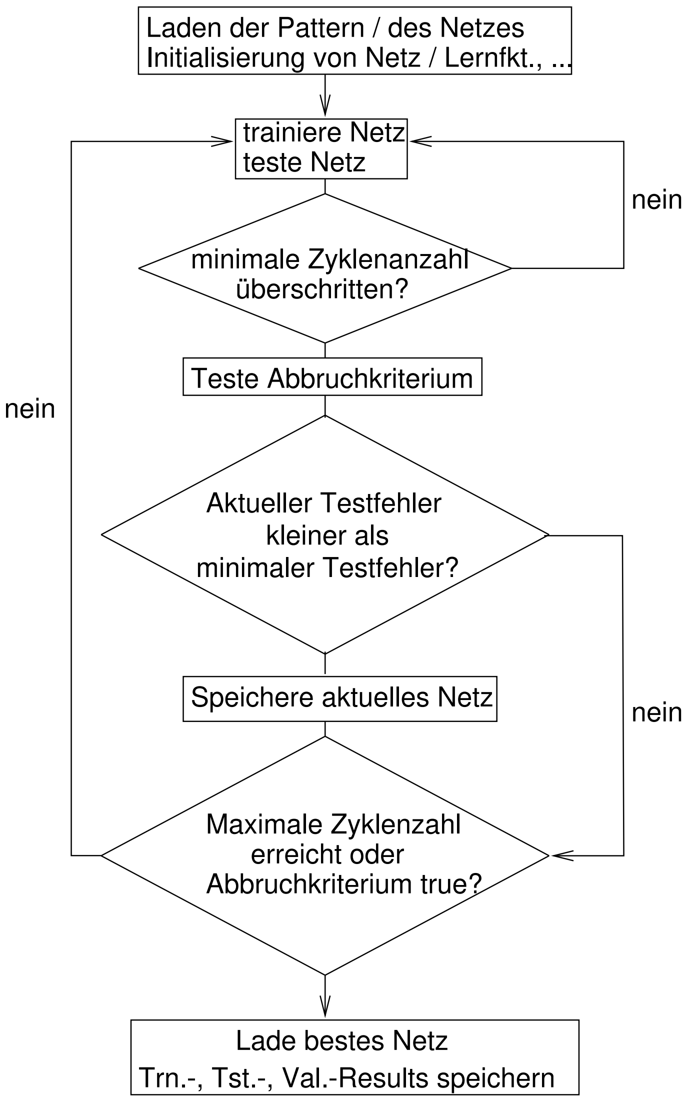


Abbildung 4.3: Ablauf eines Standard-Batchman-Skripts

Kapitel 5

Dateien im Detail

Es werden hier zwei elementare Dateien vorgestellt, die sich bei allen Prognoseläufen finden lassen und hohen Informationsgehalt hinsichtlich der Parametereinstellungen der Prognose und deren Ergebnis besitzen. Zum einen ist dies das *BP-File*, welches alle für die Prognose notwendigen Einstellungen besitzt, also vor dem eigentlichen Lauf vorliegen muß, zum anderen das *Log-File*[1], welches während des Laufes geschrieben wird. Auf das Log-File greifen Auswertungstools und Analysetools zu, um Daten von ihm zu verwenden oder in es zu schreiben.

5.1 Das BP-File

Das BP-File enthält alle wichtigen Einstellungen, um eine Zeitreihenprognose starten zu können. Die Eintragungen finden zeilenweise statt und werden, wenn es sich um einen variablen Parameter handelt, mit dem Stringwert `PARAM` eingeleitet. Danach folgt der Name des Parameters und seine Einstellung. Mir dieser Konvention brechen nur die zeitraumabhängigen Einstellungen (im Beispiel in den ersten vier Zeilen zu finden). Kommentare beginnen mit einem „#" und werden überlesen. Die Reihenfolge der Parameter in einem BP-File ist willkürlich, jedoch sollte sie zwecks Übersichtlichkeit in logische Gruppen aufgeteilt sein. Werden Parameter ausgelassen, so treten Standardeinstellungen dieser Parameter in Kraft. Es folgt ein BP-File in üblichem Umfang als Beispiel mit der Erläuterung seiner Parameter. Weitere Parameter, die Sonderfunktionen besitzen, werden in den folgenden Kapiteln behandelt.

```
MEASURE                 Wochen          # Zeitraum der Prognose festlegen
START                   36/94
END                     37/97
ADVANCE                 2               # Luecke zwischen letztem Wert und Prognose
```

[1] Es gibt noch ein weiteres Log-File oder Protokolldatei, welches den Ablauf einer Prognose überwacht und Erfolg oder Mißerfolg von Teilschritten protokolliert.

```
#--------------------allgemeine Daten----------------------
PARAM EPOCHEN           1000          # Maximale Trainingszyklen
PARAM ERR               -4
PARAM MINEPOCHEN        50            # Minimale Trainingszyklen
PARAM SHUFFLE           TRUE
PARAM MAXERROR          0.1
PARAM TRNTST            tst
#----------------------------------------------------------

#---------------------Net_Func-----------------------------
PARAM LEARNREM          Verfahren, ETA, IMP, FLATSPOT, DMAX, GLAETTUNG, NEG_AMP, POS_AMP
PARAM LEARNFUNC         BPM_ln_cosh
PARAM LEARNPARAM        0.2, 0.5, 0.0, 0.0, 10.0, 0.12, 0.12
PARAM PARAMFAKTOR       1.0, 1.0, 1.0, 1.0, 1.0,  1.0,  1.0
PARAM UPDATEFUNC        Topological_Order
PARAM INITREM           Initialisierung, a, b
PARAM INITFUNC          Randomize_Weights
PARAM INITPARAM         0.0, 1.0
#----------------------------------------------------------

#-----------------Netztopologie----------------------------
PARAM BIGNET            FeedForward
PARAM HIDDEN            .X:6
PARAM OUTPUT            1
PARAM SHORTCUT          I-O
PARAM ACT               O:Act_Identity
PARAM OUT               O:Out_Clip01
#----------------------------------------------------------

PARAM RANDOMIZE         90%   # Selektion der Testpattern
PARAM TEST_PAIRS        30%   # Anzahl der Testpaare
PARAM VAL_PAIRS         10    # Anzahl der Validierungspaare

PARAM MACHINES   1

# diese Zeitreihe wird prognostiziert
OUTPUT $KOMP1 $KOMP2 $KOMP3 ABV.LIN

# diese Zeitreihen gehen ein
UNKNOWN 4 $KOMP1 $KOMP2 $KOMP3 ABV.LIN  # Abverkauf
UNKNOWN 2 $KOMP1 $KOMP2 Regal RUS.LIN   # Regalumsatz
USE     3 $KOMP1 $KOMP2 $KOMP3 PRS.TRD0 # Preis
USE     1 $KOMP1 $KOMP2 $KOMP3 AKT.E_1T # Aktion
USE     2 Global Feiertage Hessen.BIN   # Feiertage
```

Datei `bp.bsp`: Beispiel einer BP-Steuerungsdatei

5.1.1 Die Parameter

MEASURE: gibt die Zeiteinheit oder den Zeittick an, in der die Zeitreihendaten vorliegen und in der die Prognosezeitreihe geschrieben wird. Mögliche Werte sind `Quartale`, `Monate`, `Wochen`, `Tage` und der unbestimmte Einheitentyp `Ticks`.
START, END: sind Zeitpunkte in der Genauigkeit von *MEASURE*. Sie umfassen einen Zeitraum wie er in Abbildung 5.1 dargestellt wird. Er beinhaltet sowohl Trainings-, Test- als auch Prognosezeitraum. Die gestrichelten Linien bezeichnen das Intervall, in dem sich die Parameter bewegen können.
Liegt *END* innerhalb des Zeitraums der vorhandenen Zeitreihendaten, so handelt es sich um eine *ex-post* anderenfalls um eine *ex-ante* Prognose.
Das Eingabeformat in Abhängigkeit von *MEASURE* ist ein Datumsformat mit Slash als Trenner.

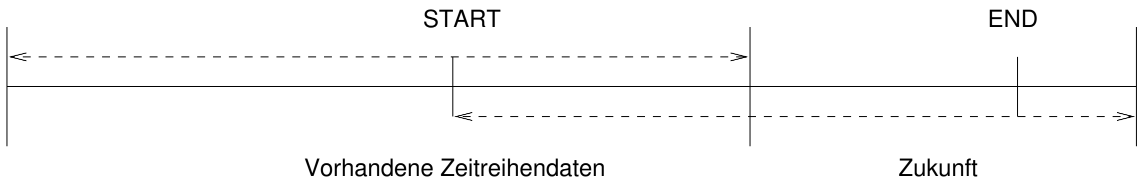


Abbildung 5.1: Parameter START und END

ADVANCE: ist ein positiver Offset in Zeiteinheiten von *MEASURE*, der zum Zeitpunkt $t-1$ addiert wird, wenn t den gegenwärtigen Zeitpunkt bezeichnet. Der errechnete Zeitpunkt ist dann der Anfang des Prognosezeitraums (siehe Abbildung 5.2). Bei Werten von *ADVANCE* größer 1 entsteht demnach eine Lücke zwischen dem letzten bekannten Zeitreihenwert am Zeitpunkt $t-1$ und dem Zeitpunkt des ersten Prognosewertes.

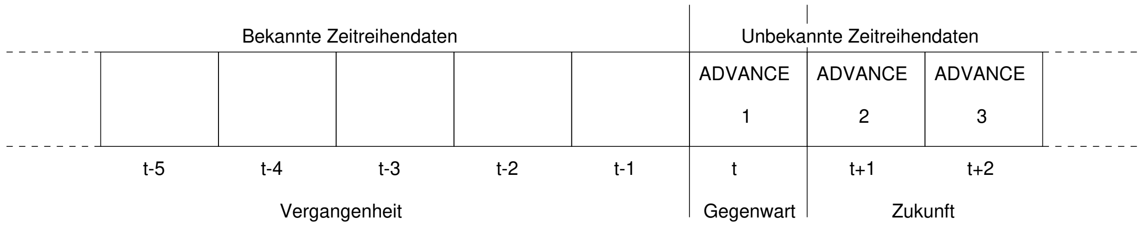


Abbildung 5.2: Parameter ADVANCE

EPOCHEN: legt die maximale Anzahl der Trainingszyklen fest, in denen das Neuronale Netz versucht zu generalisieren.
ERR: Ist dieser Parameter positiv, so werden alle `ERR` Trainingszyklen der Trainings- und der Testfehler des Netzes in eine LOG-Datei geschrieben. Ist er negativ, werden diese Fehler zu den Zyklen 2^0 bis 2^{-ERR} und dann in Abständen von 2^{-ERR} abgespeichert.

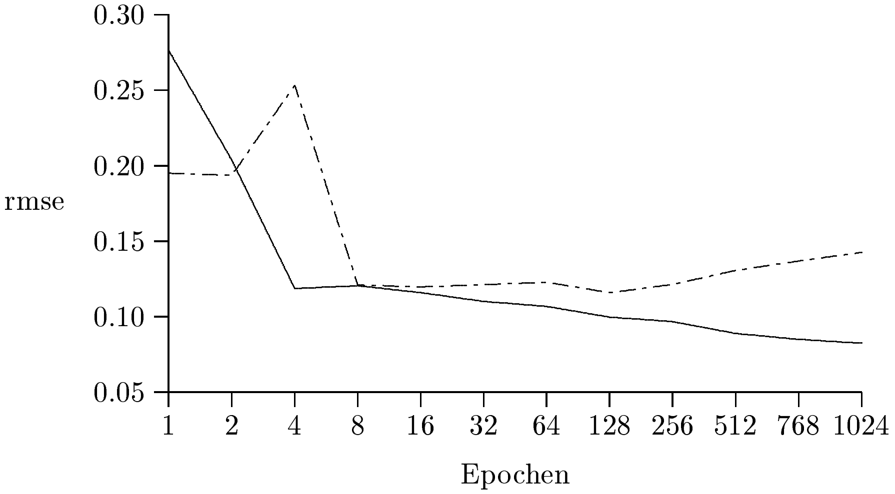


Abbildung 5.3: Trainings-, Testfehlerverlauf (solid, gestrichelt) `ERR`: -8

```
ERR: 8
epoch:  8 error: 0.229531 0.292499
epoch: 16 error: 0.225013 0.294161
epoch: 24 error: 0.223584 0.295090
epoch: 32 error: 0.222888 0.295679
epoch: 40 error: 0.222470 0.296089
epoch: 48 error: 0.222186 0.296394
epoch: 56 error: 0.221976 0.296631
epoch: 64 error: 0.221823 0.296821
epoch: 72 error: 0.221703 0.296977
epoch: 80 error: 0.221602 0.297108
epoch: 88 error: 0.221522 0.297219
epoch: 96 error: 0.221451 0.297316
```

```
ERR: -8
epoch:    1 error: 0.385265 0.311595
epoch:    2 error: 0.272466 0.295125
epoch:    4 error: 0.240203 0.291668
epoch:    8 error: 0.229531 0.292499
epoch:   16 error: 0.225013 0.294161
epoch:   32 error: 0.222888 0.295679
epoch:   64 error: 0.221823 0.296821
epoch:  128 error: 0.221261 0.297598
epoch:  256 error: 0.220965 0.298087
epoch:  512 error: 0.220810 0.298373
epoch:  768 error: 0.220757 0.298478
epoch: 1000 error: 0.220733 0.298529
```

Diese Wertepaare werden in der Auswertung eines Prognoselaufes benutzt, um die Generalisierung graphisch zu veranschaulichen (siehe Abb. 5.3).
Bei dem verwendeten Fehlermaß handelt es sich um den Root Mean Square Error, kurz *RMSE*.

MINEPOCHEN: ist die minimale Anzahl von Trainingszyklen, die das Netz durchläuft, bevor das Abbruchkriterium, welches in *MAXERROR* beschrieben wird, greifen kann oder trainierte Netze abgespeichert werden können. Dies verhindert ein zu frühes Beenden des Trainings.
SHUFFLE: ist ein boolscher, nur zu SNNS gehöriger Wert, der angibt, ob Pattern aus der Trainingsmenge in zufälliger Reihenfolge gewählt werden sollen. Dies ist für viele Lernalgorithmen erforderlich, insbesondere für die in diesem System eingesetzten. Eine Ausnahme bilden Jordan- und Elman-Netze, bei denen *SHUFFLE* den Wert `FALSE` haben muß, da diese Netztopologien ihren Input in fester Reihenfolge verlangen.
MAXERROR: Mit diesem Parameter wird ein Verhältnis angegeben, um das Training des Neuronalen Netzes zu dem Zyklus abzubrechen, an dem der aktuelle Ausgabefehler des

Netzes um den Anteil, den *MAXERROR* beinhaltet, vom bisher minimalen Ausgabefehler größer ist als der bis zu diesem Zyklus minimale Ausgabefehler (siehe Abbildung 5.4). Hinter diesem Vorgehen steht die Annahme, daß nach übermäßigem Ansteigen des Fehlers kein neues Minimum errreicht werden kann, da das Netz anstatt zu generalisieren nur noch „auswendig lernt". Es tritt dann der Effekt des *Overfitting* auf.

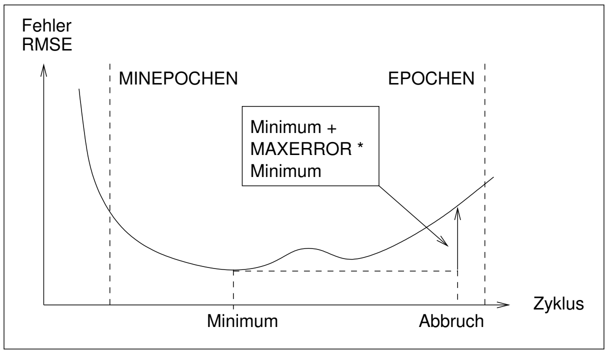


Abbildung 5.4: Parameter *MAXERROR*

TRNTST: kann die Werte `trn` und `tst` annehmen. Es gibt an, ob sich der Wert von *MAXERROR* auf den Fehler der Trainingsmenge oder auf den Fehler der Testmenge bezieht.
LEARNFUNC: bezeichnet den exakten Namen eines in SNNS implementierten Lernverfahrens, welches zum Training des Neuronalen Netzes benutzt wird.
LEARNPARAM: enthält durch Kommata getrennt die zur Lernfunktion zugehörigen Parameter. Diese müssen in Gleitkommaschreibweise vorhanden sein.
LEARNREM: Durch Kommata getrennt werden hier Stichworte zur Lernfunktion und zu Lernparametern gegeben, um eine schnelle Identifizierung des eingesetzten Verfahrens zu gewährleisten. Die Anzahl der Stichworte muß der Anzahl der Verfahrensparameter unter *LEARNPARAM* entsprechen, da die Eintragungen eine Funktion innerhalb der zum Prognosesystem gehörigen graphischen Oberfläche haben.
PARAMFAKTOR: Jeder Parameter aus *LEARNPARAM* wird mit dem entsprechenden Parameter aus *PARAMFAKTOR* nach jedem Trainingszyklus multipliziert, so daß sich exponentiell steigende oder fallende Lernparameter ergeben.
UPDATEFUNC: bezeichnet den exakten Namen einer in SNNS implementierten Updatefunktion. Diese Funktion gibt an, in welcher Reihenfolge die Ausgaben der Neuronen des verwendeten Netzes berechnet werden.
INITFUNC, *INITPARAM*, *INITREM:* In Anlehnung an die Parameter der Lernfunktion geben diese Parameter die Initialisierungsfunktion mit den dazugehörigen Parametern und deren Beschreibung an. Die Initialisierungsfunktion steuert die Einstellung der initialen Netzgewichte vor Trainingsbeginn.
BIGNET: ist einer der Stringwerte `FeedForward`, `Jordan` und `Elman`. Dieser Parameter bezeichnet, welche Art von Netztopologie für die Prognose verwendet wird. Zum einen gibt es die Feed-Forward-Netztopologie, also mehrschichtige, vorwärtsgerichtete, jeweils zwischen zwei

Schichten vollverbundene Netze, zum anderen die partiell rekurrenten Topologien wie sie bei Jordan- und Elman-Netzwerken vorkommen.
HIDDEN: Um die zu verwendende Netztopologie zu beschreiben, bedarf es der Angabe der Neuronenzahlen in der Eingabeschicht (Inputlayer), in den verdeckten Schichten (Hiddenlayers) und in der Ausgabeschicht (Outputlayer). Da nur einfache oder vollvernetzte aufeinanderfolgende Schichten vorkommen, genügt es, für jede Schicht einen Integerwert als Neuronenzahl anzugeben. Zusätzlich wird zwischen zwei Integerzahlen, also zwischen zwei Schichten, noch die Art der Vernetzung durch spezielle ASCII-Codes angegeben. Die Angabe der Neuronenzahlen der Ein- und Ausgabeschicht wird in gesonderten Parametern vorgenommen. Der Parameter *HIDDEN* umfaßt die in Abbildung 5.5 dargestellten Komponenten des Netzes.

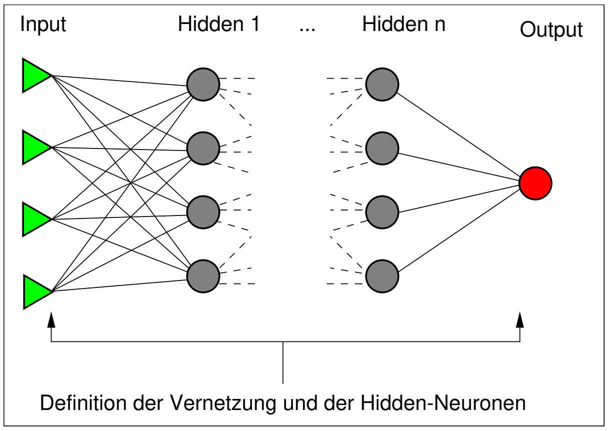


Abbildung 5.5: Parameter *HIDDEN*

Ein „`:`" steht für eine Vollvernetzung zwischen zwei Schichten, während ein „`.`" für eine Einfachvernetzung steht. Als Standardverbindung sowohl zwischen Input- und erster Hiddenschicht als auch zwischen letzter Hidden- und Outputschicht ist die Vollvernetzung eingestellt, so daß Ausdrücke wie `5:3` und `:5:3:` dieselbe Topologie beschreiben, nämlich Vollvernetzung zwischen allen Schichten, einer ersten Hiddenschicht mit fünf Neuronen und einer zweiten Hiddenschicht mit drei Neuronen. Es ist offensichtlich, daß eine Einfachvernetzung nur dann möglich ist, wenn die zu verbindenden Schichten dieselbe Neuronenzahl haben. Um eine Einfachvernetzung zwischen Input- und erster Hiddenschicht zu ermöglichen, wird der ASCII-Wert `X` benutzt, um die Anzahl der Inputneuronen zu beschreiben. So entspricht etwa der Ausdruck `.X:3:2` dem in Abbildung 5.6 dargestellten Netz, wenn vier Input- und ein Outputneuron angenommen werden.

OUTPUT: Hier wird die Anzahl der Ausgabeneuronen angegeben.
SHORTCUT: Mit Hilfe dieses Parameters können in der angegebenen Netztopologie sogenannte *Shortcut*-Verbindungen eingebaut werden. Das sind schichtenübergreifende, vollvernetzte Verbindungen. Jeweils zwei Schichten, die nicht benachbart sind, können so verbunden werden. Durch Kommata getrennt werden solche Schichtenpaare folgendermaßen angegeben: Ein `I` steht für Inputschicht, ein `O` steht für Outputschicht und `H`n steht für die n-te Hiddenschicht. So bedeutet der Ausdruck `I-O` eine Shortcut-Verbindung zwischen Input- und Outputschicht.

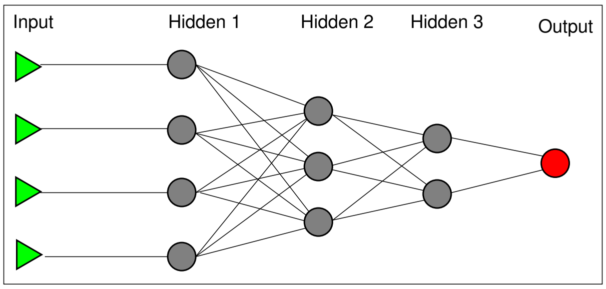


Abbildung 5.6: Beispiel für `.X:3:2`

ACT: Hier werden, durch Kommata getrennt, die Aktivierungsfunktionen für jede Schicht angegeben. Als Aktivierungsfunktion können alle dem Tool *SNNS* bekannten Funktionen benutzt werden. Als Standard ist die Sigmoidfunktion eingestellt. So stellt z.B. `H1:Act_TanH` die Aktivierungsfunktion für die erste Hiddenschicht auf den Tangens Hyperbolicus ein.
OUT: In zu dem Parameter *ACT* analoger Weise werden hier für jede Schicht die Ausgabefunktionen angegeben. Als Standard ist die Identität eingestellt.
RANDOMIZE: kann entweder die boolschen Werte `ON` und `OFF` oder eine Prozentzahl zwischen 0 und 100 annehmen. Mit *RANDOMIZE* wird die Art der Testpatternselektion aus der Patterngesamtheit gesteuert. Bei `OFF` werden die Testpattern aus den zeitlich jüngsten Daten gewählt, ohne daß eine Zufallsauswahl erfolgt. Bei `ON` werden sie zufällig aus der gesamten Datenmenge gewählt, während eine Prozentzahl P die Selektion aus den letzten, sprich jüngsten, P Prozent der Datenmenge gestattet.
TEST_PAIRS: gibt die Anzahl zu selektierender Testpattern an, entweder als numerische Konstante oder als Prozentzahl, die den Anteil der Testmenge an der Patterngesamtheit angibt. Eine Prozentzahl wird durch Anfügen des Prozentzeichens „`%`" an die Zahl kenntlich gemacht.
VAL_PAIRS: ist die Anzahl der Validierungspattern, also solcher Pattern, die nicht am Lernprozeß des Neuronalen Netzes beteiligt waren. Abbildung 5.7 verdeutlicht die Zusammenhänge der Parameter, die die Aufteilung der Patterngesamtheit steuern.

Abbildung 5.7: Parameter zur Aufteilung der Patterngesamtheit

Die folgenden Parameter beschreiben die zu verwendenden und prognostizierenden Zeitreihen. Unter *OUTPUT* steht die zu prognostizierende Zeitreihe in einer bestimmten Skalierung. In obigem Beispiel handelt es sich um eine Abverkaufszeitreihe `ABV`, die linear skaliert ist, kenntlich gemacht durch die Endung `LIN`.
Zur Erklärung der *UNKNOWN* und *USE* Zeitreihen folgendes vorab: Der jeweils zu prognostizierende Wert (Output) eines Pattern ist — ausgehend vom Beginn Z des jeweils aktuellen Zeitfensters (siehe Abbildung 5.8) — der Wert an der Position, die sich durch Addition des *ADVANCE*-Wertes und des maximalen *UNKNOWN*-Wertes ergibt. Von dem sich durch diese Position ergebenden Referenzzeitpunkt P aus gesehen, dem Ende des jeweils aktuellen Zeitfensters, sind manche Zeitreihen, wie z.B. Feiertage oder Aktionstage, schon bekannt, andere Zeitreihen, wie z.B. der Abverkauf, sind noch unbekannt. *USE* n bedeutet, benutze n Zeitreihenwerte, ausgehend von P bis $(P - n + 1)$. *UNKNOWN* n bedeutet, benutze n Zeitreihenwerte, ausgehend von $(P - ADVANCE)$ bis $(P - ADVANCE - n + 1)$ (zur Verdeutlichung dieser Zusammenhänge siehe Abbildung 5.8). Die Summe über alle n ergibt die Eingabeneuronenanzahl. Das Zusammenspiel zwischen den Patternmengen, die sich durch Verwendung von *USE* und *UNKNOWN* ergeben, dem Parameter *ADVANCE* und dem vorgegebenen Zeitraum verdeutlicht Abbildung 5.9.
Die Ausdrücke `$komp1` bis `$komp3` sind Variablen, die durch die Namen der jeweiligen Projektkomponenten ersetzt werden. Dadurch wird eine mögliche Veränderung des BP-Files verlagert zu einer Änderung der Übergabeparameter eines Trainings-Skripts (z.B. `training.pl`), welches unter anderem die Projektkomponenten erhält (siehe Abschnitt 2.3).
Schließlich folgt noch die Angabe des Typs der zu verwendenden Zeitreihe und ihre Skalierung. Dabei können alle diesem Typ zugeordneten Skalierungen aus dem *MODELL*-File benutzt werden.

5.2 Das Log-File

Das Log-File, das während eines Prognoselaufes geschrieben wird, enthält wichtige Informationen über den Verlauf der Prognose. Dazu gehören Ausgabe des Trainings- und Testfehlers zu bestimmten Zyklen, der Abbruchzeitpunkt des Trainings, die minimale Trainings- und Testfehler, Netzausgaben aller Patternmengen — insbesondere auch die der Validierungsmenge, die die Prognosepattern enthält. Informationen über Dauer des Prognoselaufes und über Einstellungen des BP-Files sind ebenfalls vorhanden. Es folgt ein Standard-Log-File, wie es sich aus dem oben gezeigten BP-File ergibt. Wenn im BP-File Sonderfunktionen aktiviert werden, so wird das Log-File um entsprechende Einträge erweitert. Dies wird in den folgenden Kapiteln erläutert.

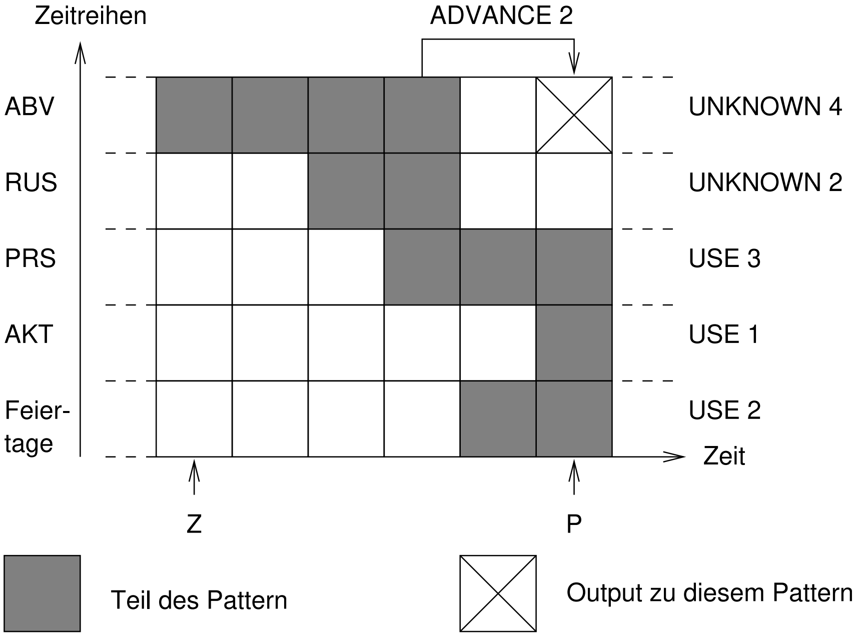


Abbildung 5.8: Die Parameter *USE* und *UNKNOWN*

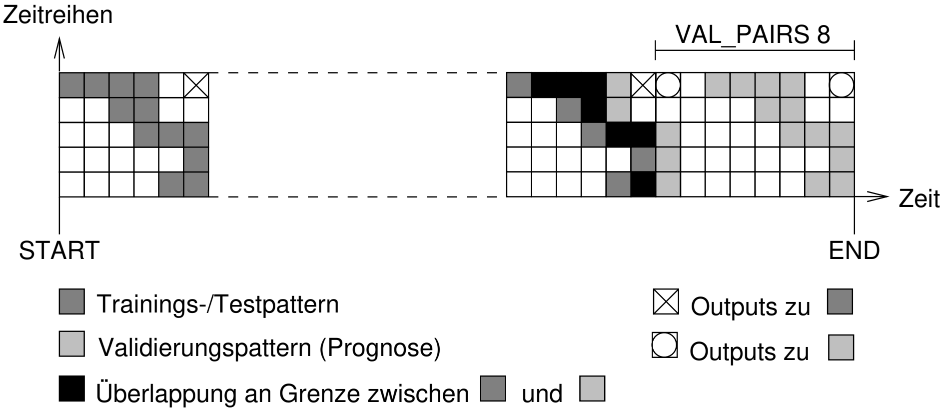


Abbildung 5.9: Zusammenhänge der verschiedenen Patternmengen auf Zeitraum *START* bis *END*

```
{{{  Fri Apr 10 00:07:03 MEST 1998
ZR-INFO: COMMENT test
ZR-INFO: START 2593 END 2765 ADVANCE 2
ZR-INFO: QUERY 0.000230 0.000260 0.000320 0.000360 0.000310
ZR-INFO: NETSIM SNNS
ZR-INFO: NETSIM-DESC SNNS-BPM_ln_cosh-Randomize_Weights
ZR-INFO: DESCRIPTION Prognoseverfahren;Kommentar;Netzkonfiguration;Advance;
```

```
ZR-INFO: DESCRIPTION min. Anzahl Epochen;max. Anzahl Epochen;Fehler alle Epochen;
ZR-INFO: DESCRIPTION max. Fehler;SHUFFLE;Validierungspaare;Verfahren;
ZR-INFO: DESCRIPTION ETA;IMP;FLATSPOT;DMAX;GLAETTUNG;NEG_AMP;POS_AMP;
ZR-INFO: DESCRIPTION Update-Verfahren;Initialisierung;a;b;BigNet;Out_Context;
ZR-INFO: DESCRIPTION Shortcut;Act;Out;Methode;zufllige Trainingsmenge;Testpaare
ZR-INFO: PARAMETERS SNNS-BPM_ln_cosh-Randomize_Weights;test;8:10:1;2;40;100;-4;10;
ZR-INFO: PARAMETERS TRUE;5;BPM_ln_cosh;0.2;0.5;0.0;0.0;10.0;0.12;0.12;
ZR-INFO: PARAMETERS Topological_Order;Randomize_Weights;0.01;-0.01;FeedForward;
ZR-INFO: PARAMETERS ;;;;;30%;55
ZR-INFO: VALPAIRS 5
ZR-INFO: TMISSING -161 0
ZR-INFO: VMISSING -5 0
ZR-INFO: LEARNING ONLINE
ZR-INFO: RANDOMIZE 30%
ZR-INFO: OUT Tegut|AAA|229104|Wochen.ABV.LIN
ZR-INFO: ZR -5 Tegut|AAA|229104|Wochen.ABV.LIN
ZR-INFO: ZR 1 Tegut|AAA|229104|Wochen.AKT.BIN
ZR-INFO: ZR 1 Tegut|AAA|229104|Wochen.PRS.TRD0
ZR-INFO: ZR 1 Global|Feiertage|Wochen.Hessen.BIN
SESSION STARTED : Fri Apr 10 00:07:03 MEST 1998
PROGRAM VERSION : SNNS V4.1 Stuttgart Neural Network Simulator
COMMANDLINE     : batchman -f Tegut.AAA.229104.bat -l tmpTegut.AAA.229104 &&
COMMANDLINE     : snnres_to_bpres.pl Tegut.AAA.229104.val.res &&
COMMANDLINE     : create_snnslog Tegut.AAA.229104.log
snns.net_create: configuration: 8:10:1
snns.net_reset: max_weight: 1
snns.net_training: file: Tegut.AAA.229104.trn.pat
snns.net_training: parameters for online learning :
snns.net_training: learnrem: Verfahren, ETA, IMP, FLATSPOT, DMAX,
snns.net_training: learnrem: GLAETTUNG, NEG_AMP, POS_AMP
snns.net_training: learnfunc: BPM_ln_cosh 0.2, 0.5, 0.0, 0.0, 10.0, 0.12, 0.12
snns.net_training: updatefunc: Topological_Order
snns.net_training: initrem: Initialisierung, a, b
snns.net_training: initfunc: Randomize_Weights 0.01, -0.01
snns.net_training: split: 0
snns.net_training: max_error: 10
snns.net_training: n_epochs: 100
snns.net_training: min_epochs: 40
snns.net_training: pairs: 106
snns.net_training: genpairs: 55
snns.net_training: err_epochs: -4
snns.net_training: epoch: 1 error: 1.51984 1.39938
snns.net_training: epoch: 2 error: 1.37011 1.36264
snns.net_training: epoch: 4 error: 1.34567 1.34847
snns.net_training: epoch: 8 error: 1.33884 1.34295
snns.net_training: epoch: 16 error: 1.33637 1.34072
snns.net_training: epoch: 32 error: 1.33535 1.33977
snns.net_training: epoch: 48 error: 1.33505 1.33949
snns.net_training: epoch: 64 error: 1.3349 1.33936
```

```
snns.net_training: epoch: 80 error: 1.33482 1.33929
snns.net_training: epoch: 96 error: 1.33476 1.33924
snns.net_training: epoch: 100 error: 1.33475 1.33923
snns.net_training: time: 10 s after 100 epochs
snns.net_training: configuration: 8:10:1
snns.net_training: net_saved: 1 times
snns.net_training: training_error: 1.33475
snns.net_training: test_error: 1.33923
snns.net_training: at_cycle: 100
snns.net_training: test_error_min: 1.33923
snns.net_training: sbic: NOT USED
snns.net_training: info: NOT USED
snns.net_training: query: 0.00019 0.00019 0.00018 0.00121 0.00023 0.00016 ...
snns.net_training: FOLDINFO: NOT USED
SESSION ENDED : Fri Apr 10 00:07:43 MEST 1998
}}}
```

Datei `Tegut.AAA.229104.log`: Beispiel eines Log-Files

Der Abschnitt, der mit `ZR-INFO` eingeführt wird, enthält neben den Einstellungen des BP-Files die prognostizierten Werte unter dem Eintrag `QUERY`. Diese liegen noch in der im BP-File unter `OUT` angegebenen Skalierung vor und werden bei Generierung der endgültigen Prognosezeitreihe rückskaliert. Die Anzahl der Werte entspricht der Anzahl des BP-File Parameters `VALPAIRS`.
Im Abschnitt `COMMANDLINE` findet sich der Kommandozeilenaufruf wieder, der den eigentlichen Prognoseprozeß gestartet hat bis hin zur Erstellung dieses Log-Files.
Der letzte Abschnitt `snns.net_training` beinhaltet Daten, die das Training mit dem entsprechenden Simulationstool betreffen. Einige Einträge wie `split` oder `sbic` sind aus Kompatibilitätsgründen vorhanden, haben aber unter *SNNS* keine Bedeutung. Es werden die Einträge erläutert, die nicht weitestgehend selbsterklärend sind.
`pairs` bezeichnet die Anzahl der zur Verfügung stehenden Trainingspattern.
`genpairs` bezeichnet die Anzahl der zur Verfügung stehenden Testpattern.
Unter `epoch` findet sich zur jeweiligen Epoche der Trainings- und der Testfehler, die durch ein Batchman-Skript ausgegeben wurden. Auf diese Werte greifen Tools zu, um die Fehler graphisch zu veranschaulichen.
`net_saved` sagt aus, wie oft das Netz als ASCII-Datei während des Trainings gespeichert wurde. Das Speichern ist nötig, um das beste Netz zu erhalten.
`training_error` gibt den Trainingsfehler und `test_error` den dazugehörigen Testfehler aus, den das zuletzt abgespeicherte Netz besitzt. Zu welcher Epoche dies der Fall ist, sagt `at_cycle` aus.
Den minimalen Testfehler bezeichnet `test_error_min`.
`query` hat als Werte die Netzausgaben der Trainings- und Testmenge. Auch diese werden noch in ein endgültiges Format umgewandelt (siehe `QUERY` unter `ZR-INFO`).

Kapitel 6

Kreuzvalidierung

In diesem Kapitel wird ein Verfahren vorgestellt, mit dessen Hilfe verschiedene Netztopologien hinsichtlich ihrer Klassifikationsgüte verglichen werden können. Das Problem besteht darin, daß eine vermeintlich gute Netztopologie, mit der nur ein einziger Trainingslauf durchgeführt wird, keinen aussagekräftigen Testfehler hergibt, um ihn von Testfehlern von Läufen anderer Topologien abgrenzen zu können, da dem Netz nur genau eine Kombination von Trainings- und Testmenge präsentiert wird. Dieser Umstand wirkt sich umso negativer aus, je kleiner die Anzahl der Patterngesamtheit ist, da dann der Testfehler stark von der Wahl der Trainings- und Testmenge abhängt.
Ist diese vermeintlich gute Netztopologie sogar noch durch viele insignifikante Parameter überspezifiziert, dann versucht das Netz im Training „white noise" zu lernen, und es kann somit nicht richtig generalisieren. Dieser Effekt wird auch als *Overfitting* des Netzes bezeichnet. Um die oben genannten Probleme zu vermeiden, wird die Methode der *Kreuzvalidierung* eingesetzt, die im wesentlichen einen Testfehler hergibt, der auf mehreren dem Netz präsentierten Trainings- und Testmengenkombinationen beruht. Hier sind noch einmal die Vorteile dieser Methode aufgezählt:

- Indikator für die Klassifikationsgüte einer Netztopologie,
- empfehlenswert bei kleiner Patterngesamtheit,
- verhindert overfitting des Netzes.

Die Methode der Kreuzvalidierung umfaßt folgende Schritte:

1. Das Aufteilen der Patterngesamtheit P in n gleichgroße Teilmengen $P_i, i = 1, ..., n$,
2. Für jede Teilmenge $P_i, i = 1, ..., n$ führe folgende Schritte durch:
   (a) Trainiere das Netz mit der Trainingsmenge $P \setminus P_i$.
   (b) Teste das Netz mit Testmenge P_i.
   (c) Summiere den Testfehler (hier den *Root Mean Square Error*, kurz *RMSE*) auf.

3. Berechne den mittleren Testfehler oder auch kreuzvalidierten Testfehler (*KRMSE*) über die Anzahl der Teilmengen.

Mit der Kreuzvalidierung läßt sich nun eine Netztopologie mit minimalem KRMSE finden. Ist der KRMSE im Vergleich zum nicht kreuzvalidierten RMSE eines Trainingslaufes wesentlich höher, so deutet dies auf eine Überspezifikation (overfitting) des Netzes hin.
Ein Nachteil der Kreuzvalidierung ist, daß sie ein sehr rechenintensives Verfahren ist, wobei der Aufwand mit steigender Anzahl der Teilmengen wächst. Sie sollte also nicht auf beliebig viele Netztopologien angewendet werden, sondern nur auf wenige, vorselektierte Topologien.

6.1 Parameter im Steuerungsfile

Zur Aktivierung der Kreuzvalidierung innerhalb des Prognosesystems stehen einige Parameter im BP-Steuerungsfile bereit, die im folgenden erklärt werden:

```
#---------------------Kreuzvalidierung-----------------------------------
PARAM METHOD    CROSS           # Kreuzvalidierung aktivieren
PARAM SETS      7               # Anzahl der Teilmengen
PARAM USETST    TRUE            # Abbruchkriterium fuer Training:
                                # TRUE: Testmenge FALSE: Validierungsmenge
PARAM RANDOM    TRUE            # Pattern fuer Sets zufaellig aus Pattern-
                                # gesamtheit auswaehlen
#-----------------------------------------------------------------------
```

Datei `bp.cross`

METHOD: Wird hier der Stringwert `CROSS` eingetragen, so wird die Kreuzvalidierung aktiviert. Erst dann haben die weiteren Parameter eine Wirkung.
SETS: gibt die Anzahl der Teilmengen an, die aus der Patterngesamtheit generiert wird.
USETST: Hier kann entschieden werden, ob der Abbruch des Trainings anhand des Fehlers auf der Testmenge (`TRUE`) oder anhand des Fehlers auf einer zu generierenden Validierungsmenge (`FALSE`), die auch aus der Patterngesamtheit gewählt wird, erfolgt.
RANDOM: Hat dieser Parameter den Wert `TRUE`, dann werden die Pattern für die Teilmengen zufällig aus der Patterngesamtheit ausgewählt, ist er `FALSE`, so werden die Pattern der Patterngesamtheit in Reihenfolge den einzelnen Teilmengen zugeordnet.

6.2 Die Kreuzvalidierung im Prognosesystem

Der Ablauf einer Kreuzvalidierung im Prognosesystem weist einige Unterschiede zu einem Standard-Prognoselauf auf, die im weiteren erläutert werden. So gibt es nicht nur je eine

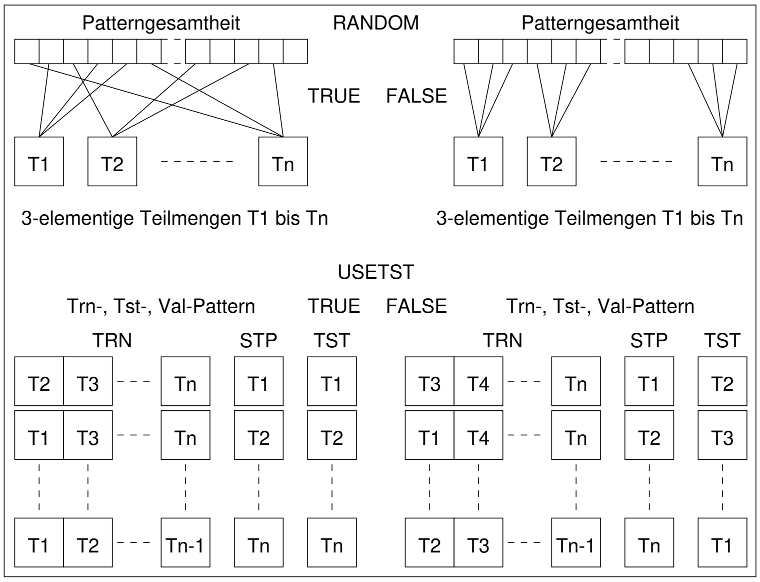


Abbildung 6.1: Die Parameter *RANDOM* und *USETST*

Trainings-, Test- und Validierungsmenge, sondern die im Parameter *SETS* angegebene Anzahl. Bei der Generierung dieser Patterndateien werden noch die Werte der Parameter *RANDOM* und *USETST* berücksichtigt (siehe Abbildung 6.1).

Die Dateinamen der Patternmengen setzen sich aus dem Zeitreihennamen, einem Kürzel (`trn` für Trainings-, `tst` für Test- und `val` für Validierungspattern), dem Index der jeweiligen Teilmenge und der Endung `pat` zusammen. So steht etwa `Tegut.AAA.229104.trn.2.pat` für das Trainingspatternfile der zweiten Teilmenge der Zeitreihe `Tegut/AAA/229104`.
Es folgt die Ausführung eines speziell für die Kreuzvalidierung erstellten Batchman-Skripts, das auf dem Template `bmanCROSS.tmpl` basiert. Das Ablaufschema des Skripts beschreibt die Abbildung 6.2.

Mit der so durchgeführten Kreuzvalidierung wird zweierlei erreicht. Zum einen ergibt sich durch die Bildung des KRMSE ein Gütewert der aktuellen Netztopologie, zum anderen wird aufgrund aller Validierungsdateien, mit denen das Netz abgefragt wird, eine ex-post Prognose auf dem die Patterngesamtheit umfassenden Zeitraum erzeugt.

6.3 Ausgabedateien

Im Gegensatz zu einem Standard-Prognoselauf ergibt sich nicht nur ein Resultfile, das durch Abfragen des Netzes mit den Validierungspattern entsteht, sondern so viele, wie es Teilmengen gibt. Die Dateinamen der Resultfiles enthalten zur Unterscheidung jeweils den Index der Teilmenge, mit der das Netz zur Prognose abgefragt wurde.

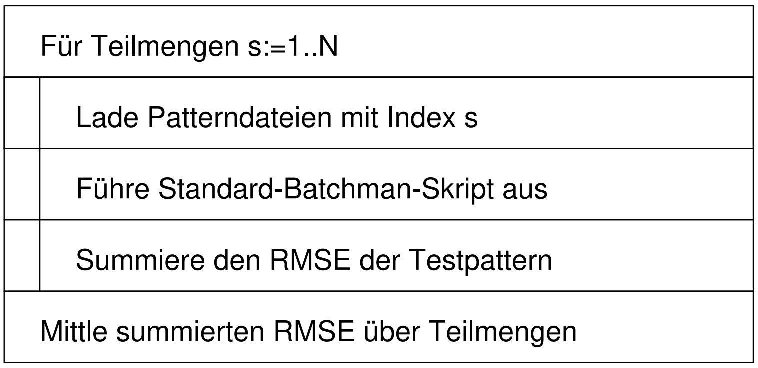


Abbildung 6.2: Ablauf der Kreuzvalidierung

Die Log-Datei, die für jede Zeitreihe nach einer Prognose geschrieben wird, enthält zusätzlich noch einen Abschnitt, der alle wichtigen Daten über die Kreuzvalidierung beinhaltet:

```
snns.net_training: CROSSVALIDATION:
snns.net_training: sets: 3
snns.net_training: random: FALSE
snns.net_training: usetst: FALSE
snns.net_training: test_error1: 1.31237
snns.net_training: test_error2: 1.37156
snns.net_training: test_error3: 1.35576
snns.net_training: add_error: 4.03968
snns.net_training: mean_add_error: 1.34656
```

Datei `cross.log`: Beispiel

Die drei ausgewiesenen Testfehler basieren jeweils auf Abfrage des Netzes mit den generierten Testpattern. Aus diesen wird dann ein gemittelter Testfehler berechnet, der dem KRMSE entspricht (hier *mean_add_error*).

Kapitel 7

Sensitivitätsanalyse

In diesem Kapitel geht es darum, Kriterien zu finden, anhand derer die Güte einer bestimmten Netztopologie für eine Prognose bestimmt werden kann. Das Ziel ist es, mit diesen Kriterien eine möglichst optimale, problemangepaßte Netztopologie zu finden. Eine solche Topologie zeichnet sich durch folgende Eigenschaften aus:

- minimale Anzahl an Inputneuronen,
- somit auch minimale Anzahl endogener und exogener Zeitreihen,
- minimale Zeitfensterbreite dieser Zeitreihen,
- angepaßte Anzahl an Hidden-Neuronen.

Ein möglicher Ansatz ist, eine Netztopologie zu bevorzugen, die sowohl während des Trainings einen kleinen Testfehler aufweist, also gut generalisieren kann, als auch bei ex-post Prognosen ein gutes Fehlerverhalten zeigt. Ein solches Vorgehen läßt aber noch keine Rückschlüsse auf die Netztopologie zu in bezug auf die oben genannten Minimalitätseigenschaften. Um auch Aussagen über die Signifikanz der Inputneuronen und damit auch der eingesetzten Zeitreihen machen zu können, finden Methoden der Sensitivitätsanalyse innerhalb des Prognosesystems Verwendung. Seien f die Abbildung, die das Netz beschreibt, x_i, $i = 1, ..., n$, die Werte der n Inputneuronen, dann ist $f(x_1, ..., x_n) = o$ mit o als Netzoutput. Für jedes Eingabeneuron i stellt sich nun die Frage, wie stark sich der Output ändert (Δo), wenn sich der angelegte Wert x_i um einen festgelegten Betrag ϵ ändert. Wichtig ist also folgende Änderungsrate $\Delta o = o - f(x_i - \epsilon)$ bzw. $\Delta o = o - f(x_i + \epsilon)$.

7.1 Praktischer Einsatz

Das Neuronale Netz kommt während des Trainings mit einer großen Zahl von Pattern in Berührung. Innerhalb der Sensitivitätsanalyse sollen alle Pattern für die Auswertung berücksichtigt werden. Dies kann auf folgende zwei Arten geschehen: Es wird ein *gemitteltes Pattern*

berechnet und dem austrainierten Netz als Inputpattern präsentiert, so daß man den dazugehörigen Output erhält. Das gemittelte Pattern läßt sich durch Bildung des Mittelwertes jeweils aller an einem Inputneuron angelegten Werte berechnen. Dabei ist es nicht notwendig, auf bestimmte Zeitreihenskalierungen wie es bei Binär- und Trendzeitreihen der Fall ist, gesondert einzugehen. Bei diesen ergeben sich zwar Mittelwerte, die außerhalb der ursprünglichen Wertemenge liegen; das Netz jedoch besitzt keine Information über das Wesen der angelegten Werte. Ist also auf diese Weise das gemittelte Pattern berechnet worden, so werden nun für jedes Eingabeneuron zwei Pattern generiert, in denen ein festgelegter Betrag ϵ sowohl zum gemittelten Wert dieses Eingabeneurons addiert als auch von diesem subtrahiert wird. Werden diese Pattern an das Netz angelegt, so erhält man zwei Outputergebnisse, von denen noch der gemittelte Output subtrahiert wird. Das Ergebnis δ ist dann das Maß für die Signifikanz dieses Neurons. Inputneuronen, an denen Binär- und Trendzeitreihen anliegen, erfahren bei der Änderung ihres gemittelten Wertes eine gesonderte Behandlung, da nicht um einen Betrag ϵ geändert wird. Statt dessen werden die Werte für *Wahr* und *Falsch* bei Binär- bzw. *Steigend* und *Fallend* bei Trendzeitreihen angelegt.
Der andere Ansatz zur Sensitivitätsanalyse, bei dem alle Pattern Verwendung finden, ist folgender: Es wird kein gemitteltes Pattern berechnet, sondern es wird für alle angelegten Trainingspattern der Netzoutput berechnet und über diesen der Mittelwert gebildet. Man erhält so einen mittleren Netzoutput. Die Änderungsrate ϵ wird nun auf jedes Trainingspattern in der schon oben beschriebenen Form angewandt. Von den so entstehenden Outputs wird der Mittelwert berechnet und vom mittleren Netzoutput subtrahiert. Dies ist dann das Maß für die Signifikanz. Die Abbildung 7.1 verdeutlicht beide Ansätze.

7.2 Parameter im Steuerungsfile

Um die Sensitivitätsanalyse innerhalb eines Prognoselaufes zu aktivieren und zu steuern, existieren einige zusätzliche Parametereinstellungen im BP-Steuerungsfile:

```
PARAM METHOD         SENSE           # Sensitivitaetsanalyse aktivieren
PARAM EPSILON        stdabw          # add/sub EPSILON oder StdAbw auf Input
PARAM STEPS          1               # Anzahl Schritte in positive und negative
                                     # Richtung
PARAM INPUT_NO       order           # Inputindex oder ORDER fuer alle Inputneuronen
                                     # nacheinander
PARAM PATTERN        trn,tst         # benutze trn, tst, val,
                                     # mean (gemitteltes Pattern ueber TRN-Daten)
PARAM SENSEOUT       pic(15,10,0),gnu # Ausgabe nach PiCTeX und Gnuplot
```

Datei `bp.sense`

METHOD: Wenn dieser Parameter den Stringwert `SENSE` enthält, so wird die Sensitivitätsanalyse aktiviert. Erst dann werden die folgenden Parameter überhaupt berücksichtigt.

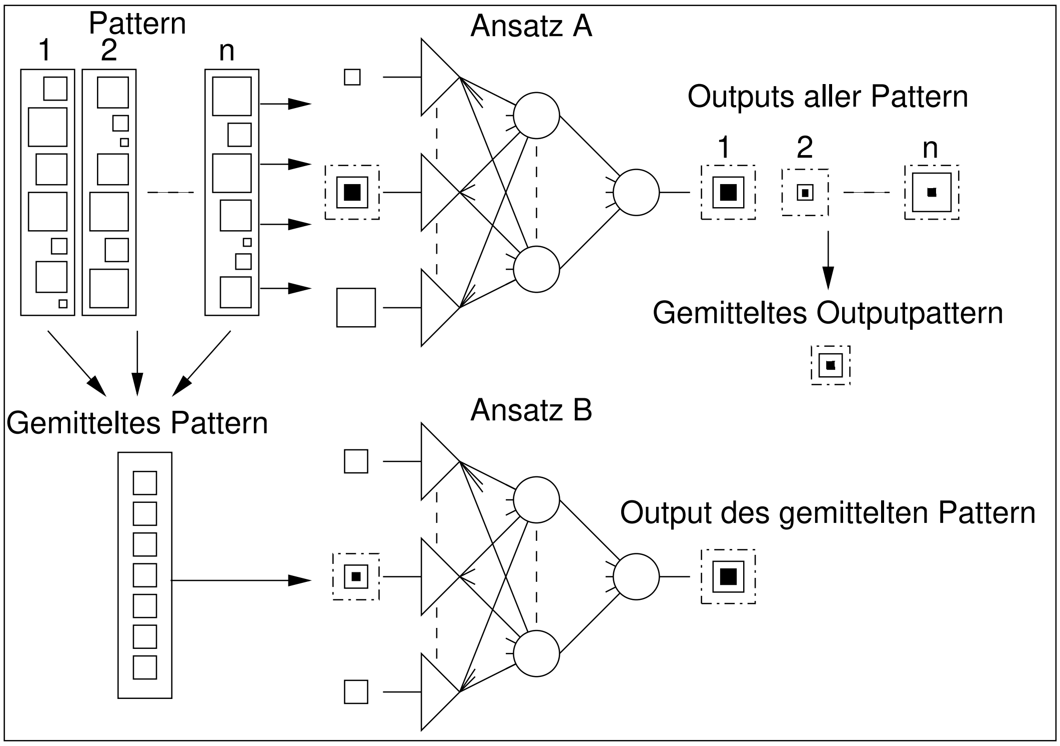


Abbildung 7.1: Die zwei Ansätze zur Sensitivitätsanalyse (Quadrate repräsentieren Input-/Outputwerte und ihre Änderungen)

EPSILON: Hiermit wird die Änderungsrate ϵ als Float-Wert eingetragen, die sich auf alle Inputneuronen außer Neuronen, die Binär- oder Trendzeitreihen repräsentieren, bezieht. Alternativ kann hier auch der Stringwert `stdabw` stehen. Dann wird die Standardabweichung für jedes Inputneuron über die in *PATTERN* angegebene Patternmenge berechnet.
STEPS: Die Anzahl der Durchläufe der Sensitivitätsanalyse kann mit diesem Parameter eingestellt werden. Der Parameter *EPSILON* nimmt dabei die Werte $1 * EPSILON$ bis $STEPS * EPSILON$ an. So ergibt sich eine diskrete Funktion in Abhängigkeit von ϵ, die die Wirkung der Inputänderung auf den Output beschreibt.
INPUT_NO: enthält den Index des Inputneurons, für das die Sensitivitätsanalyse durchgeführt werden soll. Alternativ kann der Stringwert `order` angegeben werden. Dann erfolgen Auswertungen über alle Inputneuronen.
PATTERN: Welche Pattern der Berechnung der Signifikanz zugrunde liegen, hängt von diesem Parameter ab. Der Parameter kann die Werte `trn`, `tst`, `val` oder `mean` für Trainings-, Test-, Validierungspattern und gemitteltes Pattern annehmen. Beliebig viele dieser Werte können auch durch Kommata getrennt angegeben werden, wodurch sich dann eine größere Patternmenge durch Zusammenfügen der entsprechenden Patternmengen ergibt. Durch diesen Parameter wird grundsätzlich die Vorgehensweise bei der Sensitivitätsanalyse bestimmt. Die beiden alternativen Ansätze wurden weiter oben schon erläutert.

SENSEOUT: Dieser Parameter steuert die graphische Ausgabe der Ergebnisse. Ist der Wert `gnu`, so erfolgt eine Ausgabe nach Gnuplot. Ist der Wert `pic(`*Br,H,Legende*`)`, so erfolgt die Ausgabe des Diagramms nach PICTEX mit den Maßen *Br* als Breite in cm, *H* als Höhe in cm und dem boolschen Wert *Legende*, der angibt, ob eine Legende geschrieben werden soll oder nicht.

7.3 Ablauf einer Sensitivitätsanalyse

Die Sensitivitätsanalyse beginnt nach dem normalen Ablauf einer Prognose und stützt sich auf die bis zu diesem Zeitpunkt erstellten Dateien. Es handelt sich dabei um die ASCII-Datei, die die Beschreibung des trainierten Netzes enthält, und die Patterndateien der Trainings-, Test- und Validierungsmenge. Gegebenenfalls wird noch eine einelementige Patterndatei erstellt, die ein gemitteltes Pattern enthält (wenn der Parameter *PATTERN* den Wert `mean` enthält). Es existiert neben dem Standard-Batchman-Skript ein weiteres Batchman-Skript, welches für die Abfrage des schon trainierten Netzes zuständig ist. Den Ablauf steuert im wesentlichen ein Perl-Skript namens `sense.pl`, dessen Ablaufdiagramm in Abbildung 7.2 dargestellt ist.

7.4 Ausgabedateien

In diesem Abschnitt werden die möglichen Formen der Ausgabedateien, in denen die Ergebnisse einer Sensitivitätsanalyse enthalten sind, beschrieben. Zunächst wird die ASCII-Ausgabe behandelt, die im wesentlichen die Ergebnisse als Zahlenwerte enthält. Es folgt eine Beispielausgabe für Neuron 0, das erste Inputneuron. Für die weiteren Inputneuronen existieren äquivalente Dateien. Der Dateiname setzt sich dabei aus dem Namen der zu prognostizierenden Zeitreihe, dem Kommentar, dem Inputneuronenindex und dem String `sense` zusammen, wobei die einzelnen Komponenten durch einen Punkt getrennt werden.

```
# Analyseergebnisse basieren auf 'trn'
# Neuron 0 vom Typ OTHER
# Parameter: STEPS: 1   EPSILON: stdabw


# Current epsilon is 1 * stdabw
# Current step is 1,1
# Differenzen
-0.002780 -0.002370 -0.001250 -0.002400 -0.001630 -0.002190 -0.002030 -0.002190 -0.001600
-0.001450 -0.016850 -0.001480 -0.001890 -0.002110 -0.001610 -0.002030 -0.001980 -0.002200
-0.001200 -0.002200 -0.001660 -0.001690 -0.002020 -0.013050 -0.002310 -0.002360 -0.002320
-0.004160 -0.002070 -0.002200 -0.001840 -0.001710 -0.002230 -0.002230 -0.002040 -0.002800
-0.003080 -0.001950 -0.002910 -0.002340 -0.002180 -0.002370 -0.002210 -0.001690 -0.002110
-0.002000 -0.018520 -0.001780 -0.002260 -0.020710 -0.002280 -0.002570 -0.001870 -0.003710
-0.004260 -0.003370 -0.002010 -0.002090 -0.001690 -0.004660 -0.002450 -0.021260 -0.002940
```

```
-0.003580 -0.002280 -0.001920 -0.001240 -0.001500 -0.002240 -0.002070 -0.002170 -0.001580
-0.002120 -0.002240 -0.001900 -0.001920 -0.002480 -0.002350 -0.001910 -0.002130 -0.002140
# Mittlere Differenz: -0.003200

# Current step is -1,1
# Differenzen
0.003040 0.002590 0.001380 0.002610 0.001770 0.002390 0.002230 0.002400 0.001760
0.001590 0.017690 0.001620 0.002060 0.002300 0.001760 0.002230 0.002170 0.002410
0.001300 0.002410 0.001820 0.001840 0.002210 0.013890 0.002520 0.002580 0.002540
0.004530 0.002260 0.002400 0.002020 0.001880 0.002440 0.002440 0.002250 0.003060
0.003360 0.002130 0.003170 0.002560 0.002380 0.002600 0.002430 0.001850 0.002310
0.002190 0.019320 0.001950 0.002480 0.020010 0.002500 0.002820 0.002040 0.003390
0.003910 0.003660 0.002190 0.002290 0.001850 0.005080 0.002680 0.020600 0.003220
0.003900 0.002490 0.002100 0.001370 0.001640 0.002450 0.002260 0.002380 0.001730
0.002310 0.002450 0.002080 0.002090 0.002710 0.002570 0.002100 0.002320 0.002340
# Mittlere Differenz: 0.003391
```

Datei `n0.sense`

Welche Pattern dem Netz nun vorgelegt worden sind, findet sich in der ersten Zeile. Dort ist also der Parameter *PATTERN* wiedergegeben. In der nächsten Zeile findet man den Index des Inputneurons und seinen Typ, von dem abhängt, wie der jeweils aktuell anliegende Wert eines Patterns an diesem Neuron geändert wird, da Binär- und Trendzeitreihen anders behandelt werden als die übrigen Zeitreihentypen, wie unter Abschnitt 7.1 erläutert. Parametereinstellungen bezüglich der gewählten Änderung ϵ und der Anzahl der Wiederholungen sind in der nächsten Zeile eingetragen.
Für jede Wiederholung werden zwei Zahlenblöcke angelegt. Der erste Block enthält Ergebnisse, die sich durch Addition der Änderung ϵ auf einen angelegten Wert an Neuron 0 ergeben, der zweite diejenigen, die sich durch Subtraktion ergeben. In dem jeweiligen Abschnitt *Differenzen* stehen also so viele Zahlen, wie es Pattern in der Menge gibt, auf der die Analyseergebnisse basieren. Eine Zahl innerhalb eines Blockes bezeichnet die Änderung des Netzoutputs zwischen dem ursprünglich angelegten Pattern und dem um ϵ veränderten Pattern, wobei sich die Änderung um ϵ jeweils auf das aktuell betrachtete Inputneuron bezieht (hier nur auf Neuron 0). Außerdem wird noch eine mittlere Differenz ausgegeben, auf die sich die graphische Ausgabe stützt.
Die graphische Ausgabe als Diagramm zeigt auf der x-Achse namentlich alle Inputneuronen. Auf der y-Achse werden dann die mittleren Abweichungen dargestellt, die durchgezogenen Linie für eine positive Änderung um ϵ, die gestrichelte Linie für eine negative Änderung. So sind die mittleren Differenzen aus dem obigen Zahlenbeispiel in Diagramm 7.3 beim Wert $x = 1$ zu finden, der stellvertretend für das erste Inputneuron steht.

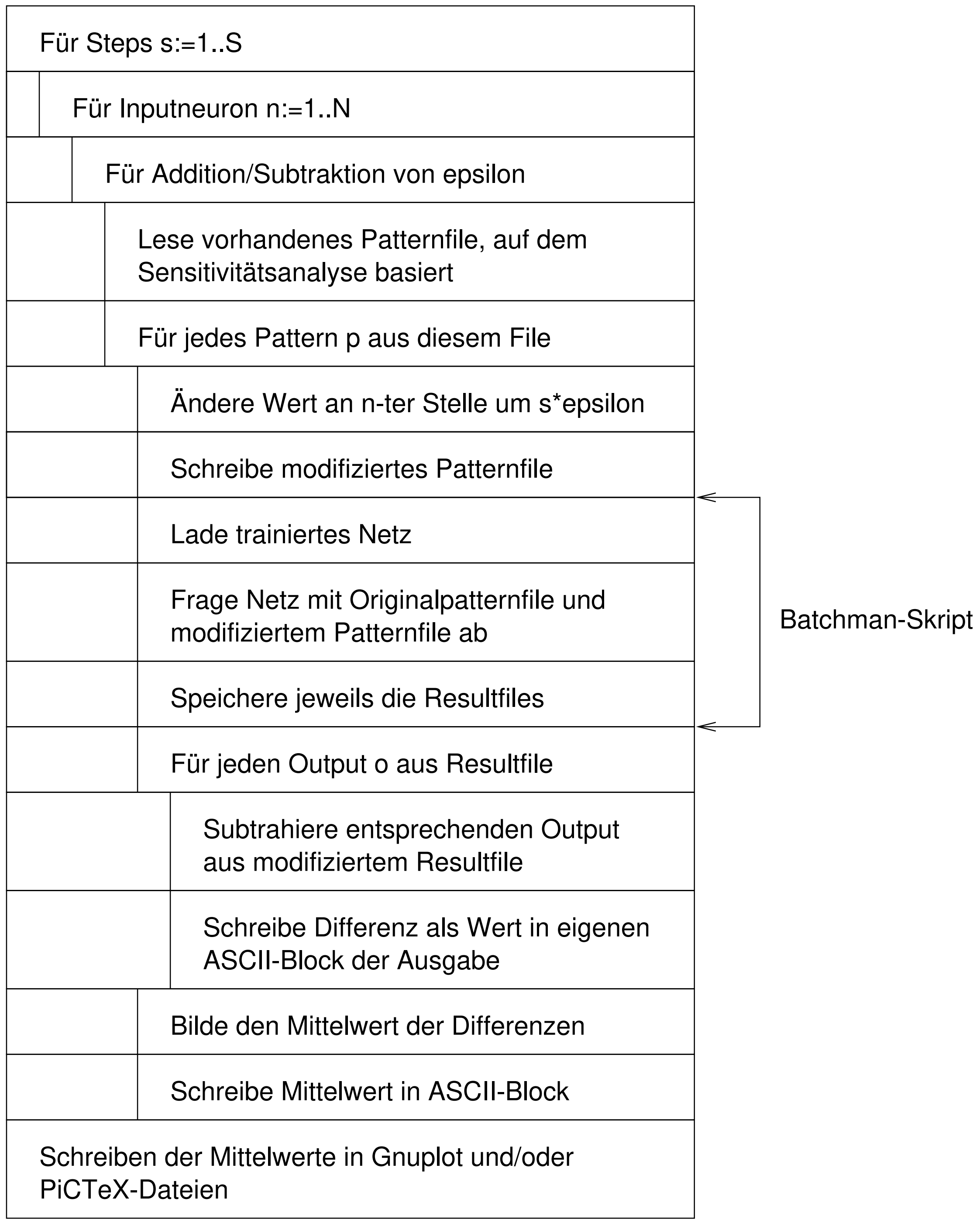
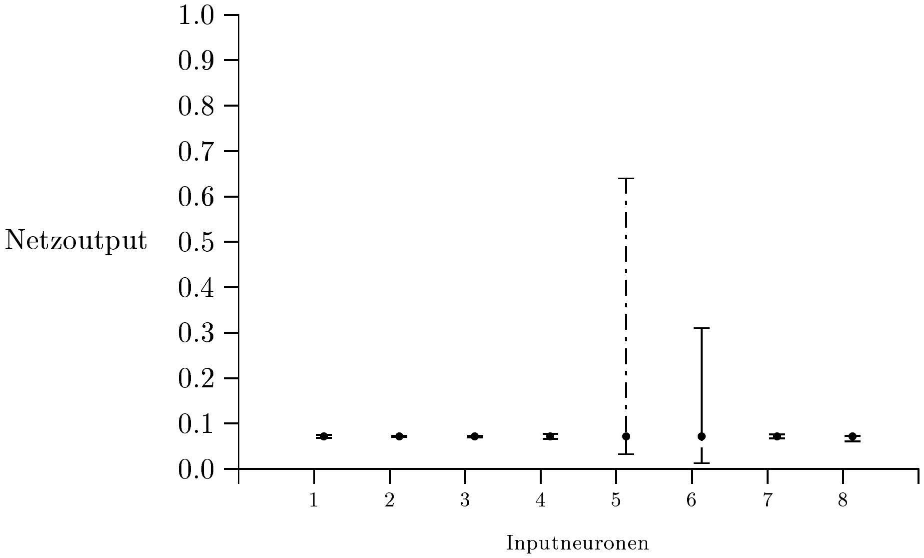


Abbildung 7.2: Ablauf einer Sensitivitätsanalyse



1=ABV.SIG4CUT0, 2=ABV, 3=ABV, 4=ABV, 5=AKT.BIN, 6=PRS.TRD0, 7=RUS.SIG4CUT0, 8=Hessen.BIN

Abbildung 7.3: Beispiel zur Sensitivitätsanalyse

Kapitel 8

Pruning-Verfahren

In Kapitel 7 wurde gezeigt, wie die Sensitivitätsanalyse benutzt werden kann, um die Signifikanz der Inputneuronen festzustellen und anhand der damit gewonnenen Ergebnisse eine dem Prognoseproblem angepaßte Netztopologie zu finden. Nun ist es aber recht mühsam, eine große Anzahl verschiedener Topologien zu vergleichen, um so eine beste Topologie zu finden. Wünschenswerter ist es, daß sich das Netz durch geeignete Lernverfahren selbst an eine beste Topologie, ausgehend von wenigen Basistopologien, anpaßt, und diese dann nur noch mittels der Sensitivitätsanalyse bestätigt wird. Genau diese Eigenschaft bieten sogenannte *Pruning-Algorithmen*. Sie versuchen, Netztopologien auszudünnen, indem sie überflüssige Links oder Neuronen entfernen. Folgende Vorteile ergeben sich damit:

- Es kann die beste Netztopologie gefunden werden.
- Speicherplatz und Rechenzeit können dadurch gespart werden.
- Die gefundene Netztopologie kann besser generalisieren.
- Das Auffinden überflüssiger Inputneuronen gibt Auskunft über unnötige Eingabewerte (Zeitreihen, Zeitfensterbreite einer Zeitreihe).

Im allgemeinen sind folgende Schritte beim Pruning durchzuführen:

1. Wähle eine geeignete Basistopologie.
2. Trainiere das Netz, bis ein möglichst geringer Testfehler ausgegeben wird.
3. Berechne für jeden Link oder jedes Neuron (in Abhängigkeit des gewählten Pruning-Algorithmus) seine „Wichtigkeit" innerhalb des Netzes.
4. Entferne (Prune) das Element mit der geringsten „Wichtigkeit".
5. Trainiere das Netz weiter.

6. Wenn der Testfehler nicht größer wird, beginne wieder mit Schritt 3.
7. Stelle das zuletzt entfernte Element wieder her, um den kleinsten Testfehler zu erhalten.

Alle in *SNNS* eingebauten Pruning-Algorithmen sind innerhalb des Prognosesystems anwendbar und im BP-Steuerungsfile frei konfigurierbar. Damit sind sowohl Verfahren einsetzbar, die Links entfernen, wie das einfache *Magnitude Based Pruning*, *Optimal Brain Damage* und die effizientere Version *Optimal Brain Surgeon*, aber auch Verfahren, die Neuronen entfernen, wie *Skeletonization* und das Verfahren der *Non-contributing Units*.

8.1 Parameter im Steuerungsfile

Folgende Parameter stehen im BP-Steuerungsfile bereit, um mit den vorhandenen Pruning-Algorithmen umzugehen:

```
#------------------Pruning------------------------------------
PARAM METHOD      PRUNE            # Pruning aktivieren
PARAM PRUNFUNC    Skeletonization  # Pruning-Algorithmus
PARAM PRUNEPOCHEN 50               # Epochen fuer Nachtraining
PARAM PRUNOBS     1e-06            # Fuer Optimal Brain Surgeon
PARAM PRUNI       TRUE             # Pruning von Inputneuronen
PARAM PRUNH       FALSE            # Pruning von Hidden-Neuronen
PARAM PRUNCYC     17               # Anzahl der Prunings
#--------------------------------------------------------------
```

Datei `bp.prune`: Ausschnitt

METHOD: Wird hier der Stringwert `PRUNE` eingetragen, so wird das Pruning aktiviert, und die folgenden Parameter werden relevant.
PRUNFUNC: Hier wird der Name des zu verwendenden Pruning-Verfahrens eingetragen.
PRUNEPOCHEN: Immer, wenn ein Element des Netzes entfernt wird, erfolgt ein Nachtraining. Hier wird die maximale Epochenanzahl dieses Nachtrainings eingestellt.
PRUNOBS: Es handelt sich bei *PRUNOBS* um einen speziell zum Verfahren *Optimal Brain Surgeon* zugehörigen Parameter, der den Initialwert der Diagonalelemente einer Hesse-Matrix angibt, die bei diesem Verfahren benötigt wird.
PRUNI: Dieser Parameter kann die boolschen Werte `TRUE` oder `FALSE` annehmen. Durch ihn kann bei Verfahren, die Neuronen entfernen, bestimmt werden, ob Neuronen in der Inputschicht entfernt werden dürfen.
PRUNH: Hier gilt das gleiche für Hidden-Neuronen wie bei *PRUNI* für Inputneuronen.
PRUNCYC: Hier wird angegeben, wie oft ein Element entfernt werden soll. In Abhängigkeit von den Parametern *PRUNI* und *PRUNH* darf der maximale Wert nicht die Anzahl der Inputneuronen (wenn *PRUNI* den Wert `TRUE` hat) plus die Anzahl der Hidden-Neuronen

(wenn *PRUNH* den Wert `TRUE` hat) überschreiten. Die angegebene Anzahl Schritte wird genau ausgeführt. Trotzdem wird die Netztopologie, die den kleinsten Testfehler aufweist, gespeichert. Wird allerdings ein Verfahren angewandt, welches Verbindungen entfernt, dann ist die maximale Anzahl der Schritte durch die Gesamtanzahl der Links im Netz gegeben.

8.2 Ablauf des Prunings im Prognosesystem

Bevor ein Pruningalgorithmus beginnt, wird ein Standard-Trainingslauf durchgeführt. Dann wird auf das trainierte Netz ein Pruning-Schritt angewandt, und das Nachtraining beginnt. Der neue Testfehler des Netzes wird mit dem vorherigen verglichen. Wenn dieser kleiner ist, wird das ausgedünnte Netz abgespeichert. Nachdem alle Pruning-Schritte durchgeführt worden sind, wird die Prognose mit dem Netz gestartet, das den kleinsten Testfehler aufwies. Dies muß nicht zwingend ein ausgedünntes Netz sein, wenn der Testfehler nach jedem Pruning-Schritt größer wird.
Der Ablauf des Standard-Trainings und des Prunings wird mit einem Batchman-Skript gesteuert, welches auf dem Batchman-Template `bmanPRUNE.tmpl` basiert (siehe Abbildung 8.1).

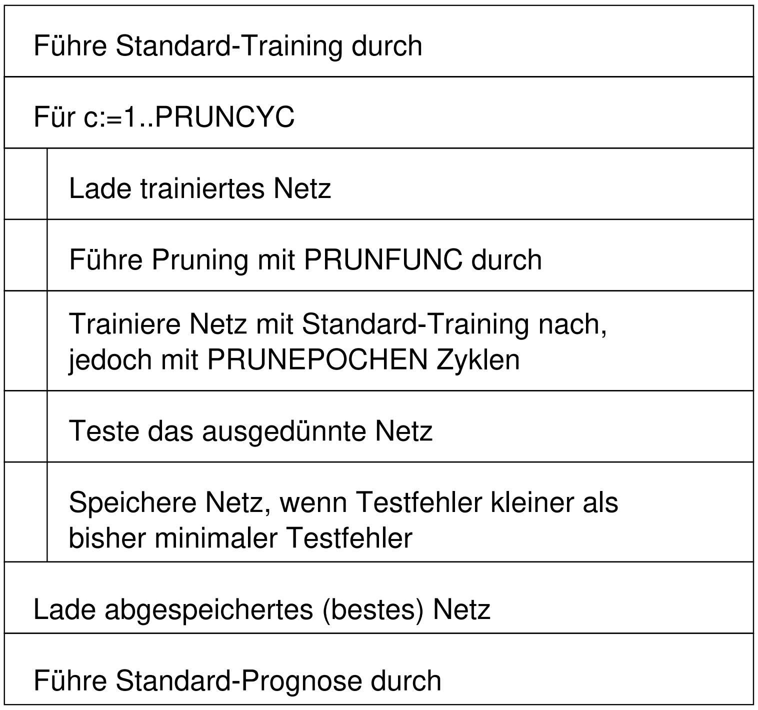


Abbildung 8.1: Ablauf des Prunings

8.3 Ausgabedateien

Neben der Netzdatei, die das trainierte Netz nach dem ersten Training enthält, gibt es nun noch eine Netzdatei, die das ausgedünnte Netz enthält. Diese Datei besitzt die Endung `.PRN.net`. In ihr kann nachvollzogen werden, welche Neuronen oder Links entfernt worden sind. Während entfernte Links im Abschnitt `connection definition section` der Netzdatei nicht mehr erscheinen, bekommen entfernte Neuronen im Abschnitt `unit definition section` das Stringattribut `si`, welches in der ursprünglichen Netzdatei entweder ein `i` für Inputneuronen oder ein `h` für Hidden-Neuronen ist[1].
Die Log-Datei, die für jede Zeitreihe nach einer Prognose geschrieben wird, enthält zusätzliche Eintragungen über den Ablauf des Prunings.

```
snns.net_training: parameters for pruning:
snns.net_training: prunfunc: Skeletonization  obs: 1e-06
snns.net_training: pruni: TRUE  prunh: FALSE  pruncyc: 17
snns.net_training: pruned: 5 time(s) after cycle 750
snns.net_training: prun_net_saved: 1
snns.net_training: prun_trn_error: 1.33282
snns.net_training: prun_tst_error: 1.43461
snns.net_training: prun_tst_error_min: 1.43461
snns.net_training: using: PRUNED Net
```

Datei `prune.log`: Beispiel

An diesem Beispiel wird folgendes deutlich: Es erfolgten tatsächlich die auch im zugehörigen BP-Steuerungsfile eingetragenen 17 Pruning-Schritte. Das beste Netz, das durch Pruning entstand, wurde aber nach nur 5 Schritten abgespeichert. Es besitzt die in der Beispieldatei angegebenen Trainings- und Testfehler. Da dieser Testfehler kleiner ist, als der, der zum standard-trainierten Netz gehört, wurde für die Prognose das ausgedünnte Netz benutzt, was durch `using: PRUNED Net` kenntlich gemacht wird. Im anderen Fall wäre dort `using: TRAINED Net` eingetragen.

[1] Leider läßt sich das *SNNS-Manual* nicht darüber aus, welche Bedeutung das Kürzel `si` hat.

Kapitel 9

Fehler- oder Zielfunktionen

Der Ansatz vieler Lernalgorithmen, insbesondere des Backpropagation und seiner Varianten, ist, die Gewichte eines Netzes so zu bestimmen, daß der mittlere quadratische Fehler zwischen Output (o) und Target (t) minimiert wird. Es wird im folgenden zur vereinfachten Darstellung ein Outputneuron und ein einzelnes Trainingspattern p angenommen, so daß sich die quadratische Fehlerfunktion als

$$E_p = \frac{1}{2}(o_p - t_p)^2$$

schreiben läßt. Diese Funktion wird auch als *Zielfunktion* bezeichnet. Die Auswertung der Fehlerfunktion erfolgt bei den Backpropagation-Algorithmen in der Rückwärtswelle und nur für die Outputschicht in einem Term

$$\delta = \frac{\partial E}{\partial net} = \frac{\partial E}{\partial o} \cdot \frac{\partial o}{\partial net} = (o - t) \cdot f'(net),$$

in der sie als Ableitung eingeht. Dabei ist net der Nettoinput des Outputneurons und $f(net)$ die gewählte Aktivierungsfunktion. Bei einer Lernrate η ergibt sich damit als Gewichtsänderung an den zur Outputschicht führenden Verbindungen

$$\Delta w_i = -\eta \cdot o_i \cdot \delta,$$

wobei i die Indizes der der Outputschicht vorgelagerten Neuronen durchläuft. Wie sich unschwer erkennen läßt, bewirken große Unterschiede zwischen Output und Target große Gewichtsänderungen, da die Differenz $(o-t)$ als Faktor in die Berechnung von δ eingeht. Für die Betrachtung ökonomischer Zeitreihen ist dies aber eher von Nachteil, da dort die Targetwerte von *Ausreißern* herrühren, die im Modell gar nicht erklärt werden können und sollen. Deshalb werden als Lösung dieses Problems weitere Zielfunktionen betrachtet, die diesen Umstand berücksichtigen.

9.1 Beispiele von Zielfunktionen

Die quadratische Übergewichtung bei großen Differenzen läßt sich z.B. durch die Betrags-Zielfunktion

$$E_p = |o_p - t_p|$$

vermeiden. Da aber die Ableitung der Zielfunktion in die Berechnung von δ eingeht und die Betragsfunktion an der Nullstelle nicht differenzierbar ist, erscheint diese ungeeignet. Um sie dennoch verwenden zu können, sollte bei kleinen Fehlerdifferenzen von der Betrags-Zielfunktion zur quadratischen Zielfunktion gewechselt werden. Es bleibt aber das Problem, daß an der Übergangsstelle die Ableitungen beider Zielfunktionen unstetig sind.
Eine Zielfunktion, die auf ihrem gesamten Definitionsbereich differenzierbar ist und die Betragsfunktion beliebig genau annähern kann, ist

$$E_p = \frac{1}{c} \cdot \ln(\cosh(c(o_p - t_p))).$$

Je größer die Konstante c ist, desto mehr gleicht sich diese Zielfunktion an die Betragsfunktion an (siehe Abbildung 9.1).

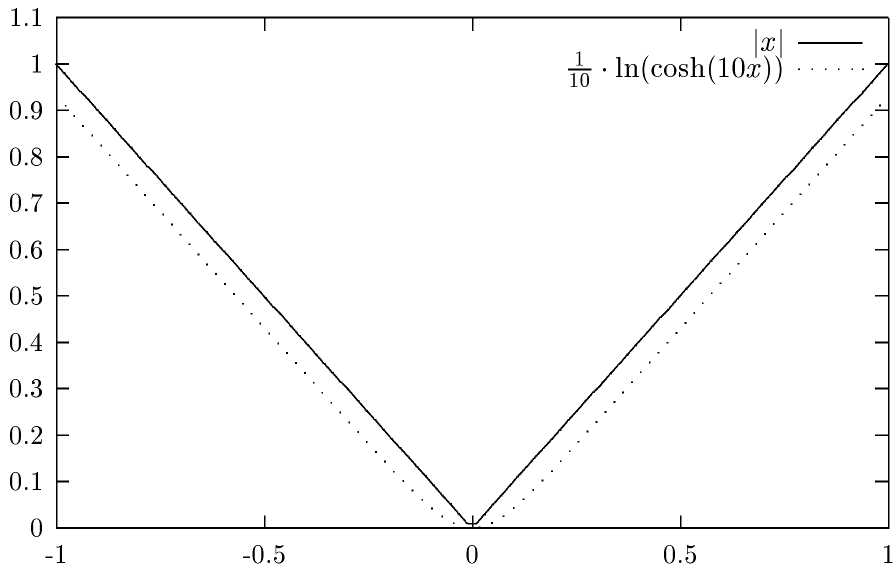


Abbildung 9.1: Zielfunktionen im Vergleich

Die Ableitung dieser Funktion nach o lautet $E'_p = \tanh(c(o_p - t_p))$ und ist in Abbildung 9.2 dargestellt.

Um zusätzlich eine Gewichtung in die Größe des über diese Zielfunktion errechneten Fehlers zu erhalten, reicht eine Modifikation mit einem Koeffizienten s aus zu

$$E_p = s \cdot \frac{1}{c} \cdot \ln(\cosh(c(o_p - t_p))),$$

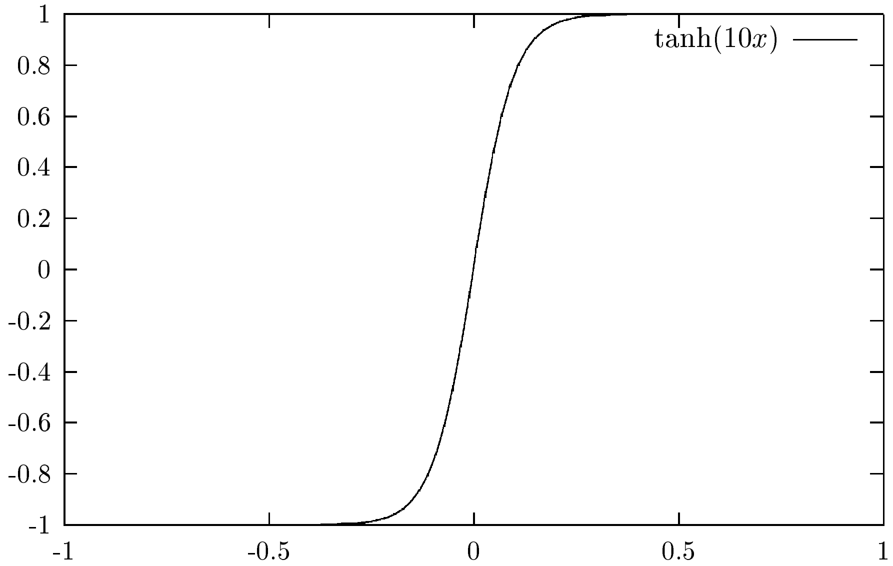


Abbildung 9.2: Ableitung der Zielfunktion $\frac{1}{10} \cdot \ln(\cosh(10x))$

oder, um positive und negative Ausreißer unterschiedlich zu bewerten, zu

$$E_p = \begin{cases} s_{neg} \cdot \frac{1}{c} \cdot \ln(\cosh(c(o_p - t_p))) & : \quad (o_p - t_p) > 0 \\ s_{pos} \cdot \frac{1}{c} \cdot \ln(\cosh(c(o_p - t_p))) & : \quad (o_p - t_p) \leq 0 \end{cases} .$$

Für die Berechnung von δ ergibt sich damit

$$\delta = \begin{cases} s_{neg} \cdot \tanh(c(o_p - t_p)) \cdot f'(net) & : \quad (o_p - t_p) > 0 \\ s_{pos} \cdot \tanh(c(o_p - t_p)) \cdot f'(net) & : \quad (o_p - t_p) \leq 0 \end{cases} .$$

9.2 Das Lernverfahren BPM_ln_cosh

Die letztgenannte Funktion erfüllt als Zielfunktion für ökonomische Zeitreihen alle Eigenschaften, um eventuelle Ausreißer in einer Zeitreihe adäquat behandeln zu können. Sie wurde deshalb in *SNNS* innerhalb eines *BackpropMomentum*-Algorithmus eingebaut, der eine Verbesserung des herkömmlichen *Vanilla Backpropagation*-Algorithmus darstellt. Zugänglich ist sie unter dem Namen `BPM_ln_cosh` und besitzt folgende sieben Parameter:

1. η: Schrittweite des Gradientenabstiegs.
2. μ: Momentum Term, der den Anteil der vorherigen Gewichtsänderung angibt, der zur aktuellen Gewichtsänderung addiert wird.
3. κ: Ein konstanter Wert, der zur Ableitung der Aktivierungsfunktion addiert wird, um *flat spots* auf der Fehlerfläche zu überqueren.

4. d_{max}: Die maximale Differenz von $o_p - t_p$, die als Null zurückpropagiert wird.
5. c: Konstante, um die Fehlerfunktion beliebig nahe an die Betragsfunktion annähern zu können.
6. s_{neg}: Faktor zur Streckung der Fehlerfunktion bei Ausreißern in negative Richtung.
7. s_{pos}: Faktor zur Streckung der Fehlerfunktion bei Ausreißern in positive Richtung.

9.3 Implementation von Zielfunktionen in SNNS

Wie das neue Lernverfahren mit der modifizierten Zielfunktion in *SNNS* implementiert wird, zeigt dieser Abschnitt[1]. Dateien, die hier angesprochen werden, befinden sich, ausgehend vom *SNNS*-Installationsverzeichnis, in `kernel/sources/`.
In der Funktionstabelle `func_tbl.c` werden die Zeilen

```
"BPM_ln_cosh", LEARN_FUNC | FF_LEARN_FUNC, 7, 1, (FunctionPtr) LEARN_BPM_ln_cosh
"BPM_ln_cosh", TEST_FUNC, 7, 1, (FunctionPtr) TEST_BPM_ln_cosh
```

Datei `func_tbl.c`: Ausschnitt

eingetragen, die das Lernverfahren zum Training und zum Testen eines Netzes definieren. Die Einträge in den Zeilen sind der Name des Verfahrens, der Typ, die Anzahl der Eingabeparameter des Verfahrens, die Anzahl der Ausgabeparameter des Verfahrens und ein Zeiger auf die benutzerdefinierte Funktion.
Funktionsprototypen müssen im Header File `learn_f.h` und im Private Header File `learn_f.ph` jeweils mit folgenden Einträgen definiert werden.

```
extern krui_err LEARN_BPM_ln_cosh(int start_pattern, int end_pattern,
                                  float *parameterInArray,
                                  int NoOfInParams,
                                  float **parameterOutArray,
                                  int *NoOfOutParams);

extern krui_err TEST_BPM_ln_cosh(int start_pattern, int end_pattern,
                         float *parameterInArray, int NoOfInParams,
                         float **parameterOutArray, int *NoOfOutParams);
```

Datei `learn_f.h`: Ausschnitt

[1] Implementationen von Transfer-, Initialisierungs und Updatefunktionen erfolgen gleichermaßen. Allein die C-Programme werden in unterschiedliche Dateien geschrieben (z.B. Updatefunktionen in `update_f.c`)

Diese Definitionen sind für alle Backpropagation-Algorithmen identisch.
Natürlich muß das Lernverfahren als C-Programm noch implementiert werden. Dies geschieht in der Datei `learn_f.c`. Es werden hier nur die Zeilen gezeigt, die den Fehler δ in der Outputschicht errechnen.

```
/* calc. error for output units  */
if (devit >= 0)
        error = tanh(streckung * devit) * neg_amplitude
        * ((unit_ptr->act_deriv_func) (unit_ptr) + FSE_term);
else
        error = tanh(streckung * devit) * pos_amplitude
        * ((unit_ptr->act_deriv_func) (unit_ptr) + FSE_term);
```

Datei `learn_f.c`: Ausschnitt

Zu beachten ist der FSE-Term, der den Momentumterm dieses Backpropagation-Verfahrens angibt. Ansonsten können die Zeilen direkt aus der Formel für δ angegeben werden.
Als abschließender Schritt ist die erneute Compilierung von *SNNS* mit dem Kommando `build` obligatorisch.

Kapitel 10

Selbstorganisierende Karten

In Kapitel 8 wurde eine Vorgehensweise besprochen, die es erlaubt, eine optimale Netztopologie zu finden, indem man störende Elemente, seien es Neuronen oder Verbindungen, entfernt, und somit unter anderem die Generalisierungsleistung des Netzes erhöht. Dieses Kapitel wendet sich nun den Inputpattern eines Netzes zu und deren Optimierung bzw. Ausdünnung. Es werden eine bestimmte Netztopologie und ein Lernverfahren angewandt, das selbständig in der Lage ist, überflüssige Inputpattern, wie solche, die Ausreißer in der Zeitreihe enthalten, zu erkennen und zu entfernen. Zwar werden auch über die Vorverarbeitung der Zeitreihen Pattern entfernt, die widersprüchlich sind, d.h. solche, die bei gleichen Inputpattern unterschiedliche Outputs besitzen, aber darüber hinaus läßt sich ohne weitere Hilfsmittel nicht feststellen, welche Pattern den Trainingsprozeß negativ beeinflussen und entfernt werden sollten.
Mit Hilfe von *Selbstorganisierenden Karten*, eine spezielle, von Kohonen entwickelte Netzarchitektur, können die gesamten Inputpattern in Klassen zerlegt werden. Alle Elemente (Pattern) einer Klasse besitzen eine gewisse Ähnlichkeitsbeziehung. Sollten also viele Trainingspattern einer Klasse angehören, so sind sie für einen Standard-Prognoselauf geeignet. Besitzt eine Klasse sehr viel weniger Elemente als andere, so werden diese aus der Trainings- und Testmenge entfernt (siehe Abbildung 10.1), da es sich um Ausreißer oder andere störende Pattern handeln kann.

Selbstorganisierende Karten sind Neuronale Netze mit einer aktiven Schicht, die mit unüberwachtem Lernverfahren trainiert werden. Die Gewichtsvektoren dieser Schicht sollen dabei so im n-dimensionalen Raum der Inputpattern verteilt werden, daß sie ihn möglichst gut abdecken[1] (siehe Abbildung 10.2).

SNNS bietet für Selbstorganisierende Karten ein spezielles Lernverfahren an, das im Prognosesystem über das BP-Steuerungsfile angesprochen wird.

[1]Eine Beschreibung des Algorithmus ist in [Zel94] zu finden.

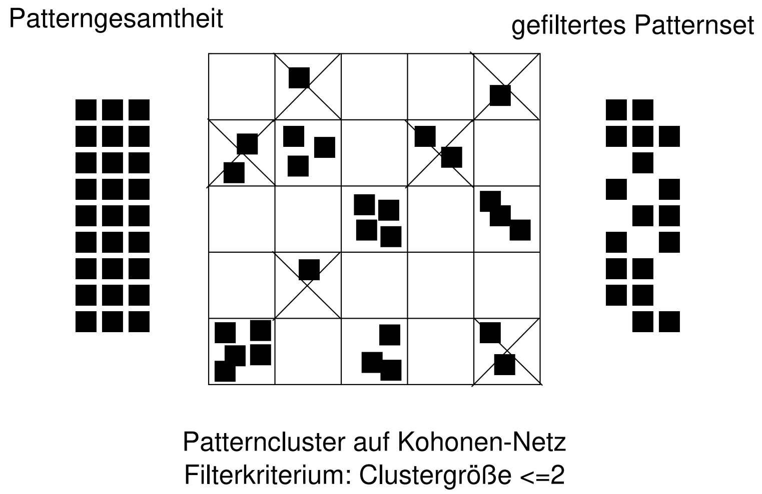


Abbildung 10.1: Zerlegung der Patterngesamtheit in Klassen

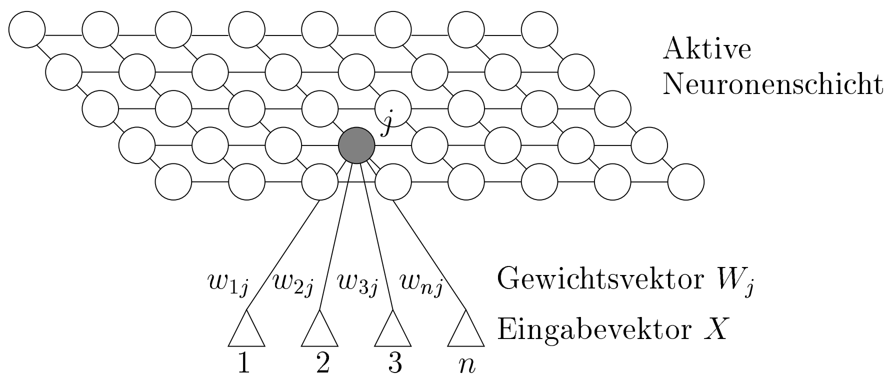


Abbildung 10.2: Kohonen-Netz mit Gewichtsvektoren eines Neurons j

10.1 Parameter im Steuerungsfile

Folgende Parameter stehen im BP-Steuerungsfile zum Umgang mit Kohonen-Netzen und Clustern der Inputpattern zur Verfügung:

```
#-------------------Kohonen-----------------------------------------------
PARAM METHOD      SOM                    # Aktivieren
PARAM KLEARNREM   Adaption Height, Adaption Radius, mult_H, mult_R, X_size
PARAM KLEARNPARAM 0.9, 4.0, 0.99998, 0.99998, 5.0
PARAM KINITREM    Initialisierung, a, b
```

```
PARAM KINITFUNC   Kohonen_Weights_v3.2
PARAM KINITPARAM  0.5, -0.5
PARAM KMAP        4:4:4:4:4              # Kohonen-Netz
PARAM KCYCLES     500                    # Trainingszyklen
PARAM KCLUSTER    7                      # Filtergroesse
#----------------------------------------------------------------------
```

Datei `bp.som`: Ausschnitt

METHOD: Durch den Stringwert `SOM` wird das Clustern mit Selbstorganisierenden Karten aktiviert und die übrigen Parameter wirksam.
KLEARNPARAM: Hierbei handelt es sich um die zum Lernverfahren `Kohonen`, das nicht explizit genannt werden muß, zugehörigen Parameter. Diese sind im einzelnen:

1. *Adaption Height*: Gibt den Initialwert der Lernrate an, der zwischen $0 < \eta(t) < 1$ liegen sollte.

2. *Adaption Radius*: Gibt den Initialwert des Nachbarschaftsradius an. Es sind Werte zwischen 1 und der Größe der Karte erlaubt. Die Änderung der Gewichte wirkt sich innerhalb dieses Radius aus.

3. *mult_H*: Der Wert *Adaption Height* fällt monoton um den Faktor dieses Parameters.

4. *mult_R*: Der Wert *Adaption Radius* fällt monoton um den Faktor dieses Parameters.

5. *X_size*: Hier muß die Breite des Kohonen-Netzes durch die Anzahl der Neuronen angegeben werden.

KLEARNREM, KINITREM: Die Namen der Parameter in *KLEARNPARAM* stehen bei *KLEARNREM*, die Namen der Parameter in *KINITFUNC* stehen bei *KINITREM*.
KINITFUNC: Ein Name der drei für Kohonen-Netze zulässigen Initialisierungsfunktionen muß hier eingetragen werden. Durch diese Funktion werden die Gewichtsvektoren initialisiert.
KINITPARAM: Notwendige Parameter der Initialisierungsfunktion stehen in diesem Parameter.
KMAP: Die Syntax dieses Parameters entspricht dem Parameter *HIDDEN* in Kapitel 5.1.1, wobei zu beachten ist, daß die Neuronenanzahl in jeder Schicht identisch ist und das Symbol „:" als Verbindung zwischen zwei Schichten gewählt wird. Es ergibt sich somit ein rechteckiges Kohonen-Netz. Jedes Neuron aus diesem Netz repräsentiert eine Klasse, der Inputpattern zugeordnet werden können.
KCYCLES: Dieser Parameter enthält die Anzahl der Trainingszyklen für Kohonen-Netze. Innerhalb dieser Zyklenanzahl können sich die Gewichtsvektoren so im Eingaberaum verteilen, daß alle Inputpattern dadurch optimal auf die Klassen verteilt werden.
KCLUSTER: Nach Beendigung des Trainings werden Pattern aus solchen Klassen entfernt, die höchstens so viele Elemente besitzen, wie in *KCLUSTER* angegeben sind. Hier wird also das eigentliche Filterkriterium spezifiziert.

10.2 Ablauf einer Klassenzerlegung mit anschließendem Prognoselauf

Bevor ein Standard-Prognoselauf mit den entsprechenden Eintragungen im BP-Steuerungsfile startet, erfolgt die Ausdünnung der Inputpattern. Dazu wird neben den Standard-Patternfiles noch ein Patternfile, bestehend aus Trainings- und Testpattern, für das Kohonen-Netz geschrieben. Es besitzt das Suffix `.som.pat`. Es unterscheidet sich von den Standard-Patternfiles dadurch, daß es keinen vorgegebenen Output pro Pattern gibt, da es sich bei dem Kohonen-Lernverfahren um unüberwachtes Lernen handelt. Neben der ASCII-Netzdatei für die Prognose gibt es noch eine Netzdatei, die das Kohonen-Netz enthält. Sie besitzt das Suffix `.SOM.net`. Nun wird vor der Standard-Prognose ein Batchman-Skript gestartet, dessen Ablaufdiagramm in Abbildung 10.3 dargestellt ist.

Abbildung 10.3: Ablauf der Entfaltung eines Kohonen-Netzes

Nachdem sich das Kohonen-Netz entfaltet hat, das heißt, seine Gewichtsvektoren den Eingaberaum möglichst gut abdecken, soll nun für jedes Inputpattern festgestellt werden, in welche Klasse es fällt. Die folgenden Schritte werden durch ein Perl-Skript namens `come_get_SOMe.pl` ausgeführt. Es wird der euklidische Abstand von jeweils einem Inputpattern zu allen Gewichtsvektoren berechnet. Der Index des Neurons der aktiven Schicht des Kohonen-Netzes, bei dem dieser Abstand minimal ist, bezeichnet dann die Klassenzugehörigkeit. Auf diese Weise werden alle Inputpattern verteilt. Dann werden aus den Trainings- und Testmengen, die nun der kommenden Prognose dienen, diejenigen Pattern entfernt, die zu solchen Klassen gehören, deren Elementanzahl kleiner als die im Parameter *KCLUSTER* eingestellte Anzahl ist. Ein Ablaufdiagramm zeigt Abbildung 10.4. Der übrige Ablauf entspricht einer Standard-Prognose, wobei das Log-File zusätzliche Einträge über das Ergebnis der Klassenzerlegung enthält.

10.3 Ausgabedateien

Zusätzliche Informationen über die Klassenzerlegung lassen sich im Log-File finden, von dem hier ein Ausschnitt dargestellt ist.

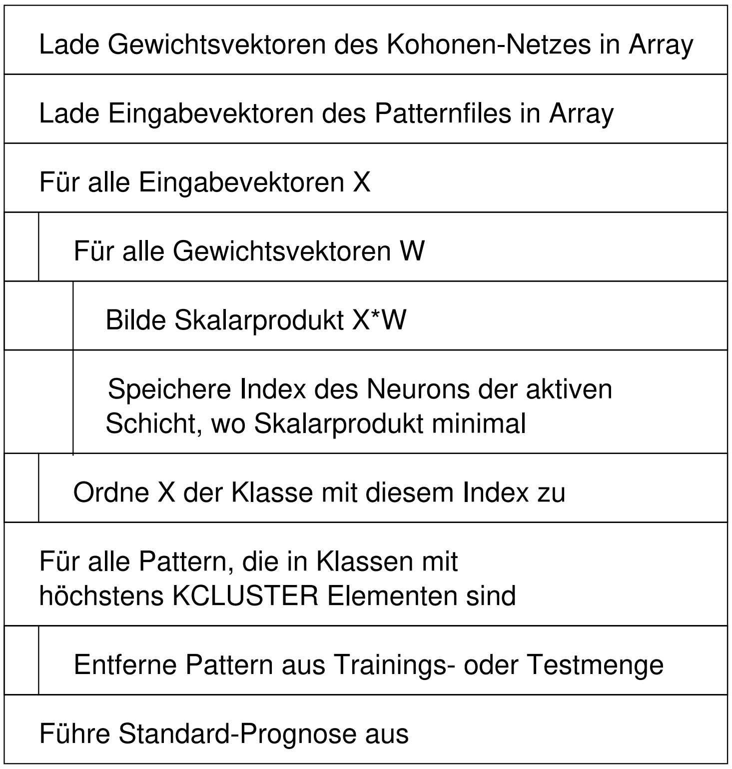


Abbildung 10.4: Ablauf des Perl-Skripts `come_get_SOMe.pl`

```
snns.som: learnrem: Adaption Height, Adaption Radius, mult_H, mult_R, X_size
snns.som: learnfunc: Kohonen 0.9, 4.0, 0.99998, 0.99998, 5.0
snns.som: updatefunc: Kohonen_Order
snns.som: initrem: Initialisierung, a, b
snns.som: initfunc: Kohonen_Weights_v3.2 0.5, -0.5
snns.som: map: 4:4:4:4:4
snns.som: epochs: 500
snns.som: cluster: 7
snns.som: file: Tegut.AAA.229104.som.pat
snns.som: Unit1 (11): 1,52,53,102,103,104,105,106,107,146,156
snns.som: Unit2 (3): 87,158,159
snns.som: Unit3 (9): 15,27,41,46,50,67,101,120,138
snns.som: Unit4 (0):
snns.som: Unit5 (21): 22,23,25,28,31,47,73,74,77,79,81,82,99,111,124,125,129,130
snns.som: Unit5 (21): 132,133,151
snns.som: Unit6 (9): 2,29,33,54,71,109,122,126,160
snns.som: Unit7 (6): 48,69,98,152,153,154
snns.som: Unit8 (11): 42,86,91,92,97,134,142,143,147,148,157
snns.som: Unit9 (2): 20,51
snns.som: Unit10 (5): 7,30,59,60,112
```

```
snns.som: Unit11 (8): 4,18,19,35,56,57,114,162
snns.som: Unit12 (5): 16,17,88,121,139
snns.som: Unit13 (8): 43,80,83,89,90,140,144,155
snns.som: Unit14 (7): 11,12,68,76,93,116,135
snns.som: Unit15 (11): 5,6,21,36,37,58,75,113,115,127,128
snns.som: Unit16 (9): 3,32,34,55,72,108,110,123,161
snns.som: Unit17 (7): 26,49,66,70,100,137,141
snns.som: Unit18 (12): 14,44,45,84,85,94,95,96,136,145,149,150
snns.som: Unit19 (7): 8,13,38,39,117,118,131
snns.som: Unit20 (11): 9,10,24,40,61,62,63,64,65,78,119
snns.som: entferne: 42 Pattern aus Trainings- u. Testmenge
snns.som: pattern:  7 8 11 12 13 16 17 20 26 30 38 39 48 49 51 59 60 66 68 69 70
snns.som: pattern:  76 87 88 93 98 100 112 116 117 118 121 131 135 137 139 141
snns.som: pattern:  152 153 154 158 159
```

Datei `som.log`: Ausschnitt

Neben den Einstellungen, die im BP-Steuerungsfile gemacht wurden, finden sich Informationen zur Klassenzugehörigkeit aller Trainings- und Testpattern in den Eintragungen unter `Unit`*Index*. Der *Index* ist eine eindeutige Zahl, die ein Neuron in dem Kohonen-Netz identifiziert. Die Zahl in Klammern ist die Anzahl der Eingabevektoren, die in der durch den Index bezeichneten Klasse liegen. Welche Pattern dies sind, zeigt die darauffolgende Liste, die als Elemente die Positionen der Pattern innerhalb der Patterndatei hat. Welche Pattern letztendlich entfernt werden, steht bei dem letzten Eintrag `pattern`.

Kapitel 11

Netztopologien

Neben dem herkömmlichen mehrschichtigen, vollvernetzten Feed-Forward-Netz werden in diesem Kapitel Modifikationen dieses Netztypen und zwei weitere Netzarchitekturen, die *Jordan*- und *Elman*-Netze, vorgestellt. Ausgehend von einem dreischichtigen Feed-Forward-Netz (siehe Abbildung 11.1) werden die Änderungen entwickelt.

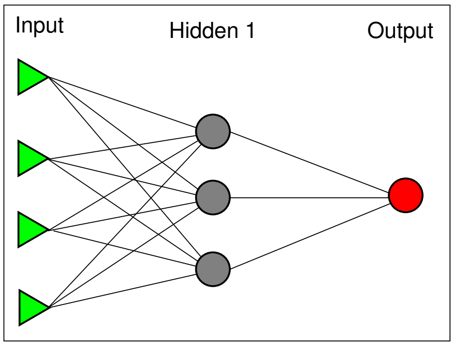


Abbildung 11.1: 3-schichtiges Feed-Forward-Netz

Dieses Netz kann mit dem Netzgenerator durch den Aufruf

```
snns_create_net.pl -f 4:3:1 -n test
```

generiert werden.

11.1 Shortcutverbindungen

Die einfachste Erweiterung des Standardmodells stellen Verbindungen zwischen Input- und Outputschicht dar. Damit lassen sich lineare Einflüsse im Modell durch die Parameter in dieser Direktverbindung repräsentieren, während nichtlineare Teile durch die Hiddenschichten modelliert werden. Damit die Linearität gewährleistet ist, muß als Aktivierungsfunktion in der Outputschicht die Identität oder andere lineare Funktionen stehen. Wie Shortcuts in einem Netz erstellt werden, wird in Kapitel 5.1.1 in der Erklärung des Parameters *SHORTCUT* beschrieben. Ein Schema zeigt Abbildung 11.2.

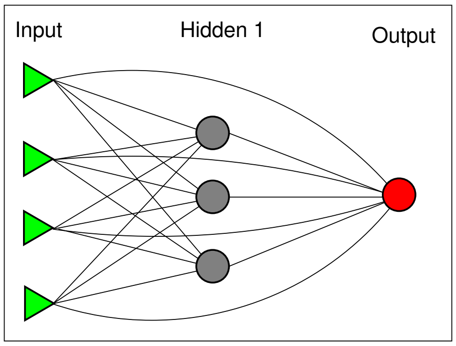


Abbildung 11.2: 3-schichtiges Feed-Forward-Netz mit Shortcut-Verbindungen

Dieses Netz kann mit dem Netzgenerator durch den Aufruf

```
snns_create_net.pl -f 4:3:1 -s I-O -a O:Act_Identity -n test
```

generiert werden.

11.2 Verwendung einer Vorschicht

Zeitreihenwerte, die zu Ausreißern gehören, stören das Training eines Netzes und sollen auch gar nicht prognostiziert werden. Neben einer geeigneten Vorverarbeitung der Zeitreihendaten, die solche Werte findet und entfernt, kann auch eine Modifizierung der Netztopologie hilfreich sein, um die Wirkung großer positiver wie negativer Ausreißer auf die Generalisierungsleistung zu verringern. Dazu wird eine Vorschicht als erste Hiddenschicht hinter die Inputschicht in das Netz gebracht, die die logistische Aktivierungsfunktion besitzt (siehe Abbildung 11.3). Diese neue erste Hiddenschicht ist mit der Inputschicht einfach vernetzt, damit die Inputwerte durch gewichtete logistische Aktivierungsfunktionen skaliert werden. Wird nun an ein Inputneuron ein Wert angelegt, der zu einem Ausreißer gehört, so wird dieser im Laufe des Trainings auf den Sättigungsbereich der logistischen Aktivierungsfunktion abgebildet und verliert im Output des entsprechenden Vorschichtneurons seinen Charakter als Ausreißer. Die Erstellung einer Vorschicht in einem Netz wird in Kapitel 5.1.1 in der Erklärung des Parameters *HIDDEN* beschrieben.

Dieses Netz kann mit dem Netzgenerator durch den Aufruf

```
snns_create_net.pl -f 4.4:3:1 -n test
```

generiert werden.

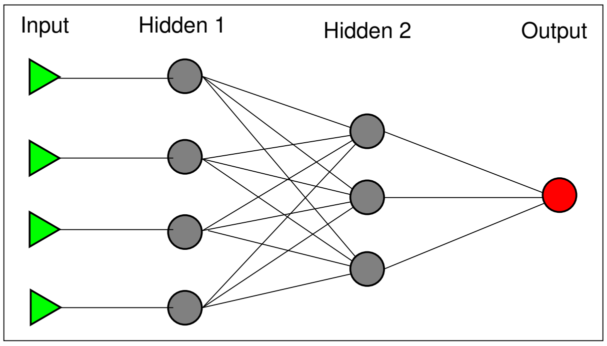


Abbildung 11.3: 3-schichtiges Feed-Forward-Netz mit Vorschicht

11.3 Jordan- und Elman-Netze

Ein einfaches Feed-Forward-Netz führt aufgrund seiner Topologie immer eine Klassifikationsaufgabe durch, ohne die Trainingsmuster in einem weiteren Kontext zu sehen. Dabei ist gerade bei Zeitreihen deren zeitlicher Kontext wichtig, das heißt, nicht nur das angelegte Muster ist wichtig, sondern auch seine Position in der gesamten Mustersequenz. Wird ein Muster an ein Netz angelegt, so sollte dessen Ausgabe nicht nur von der aktuellen Eingabe, sondern auch von den vorhergehenden Eingaben abhängig sein.
Ein Ansatz zur Repräsentation von Zeit in Neuronalen Netzen verwenden sogenannte *partiell rekurrente Netze*. Es handelt sich dabei um Feed-Forward-Netze, die durch *Kontextzellen* erweitert sind, welche einen Speichermechanismus von vergangenen Werten beinhalten. Die Outputs von Zellen einer Hidden- oder Ausgabeschicht werden an diese Kontextzellen weitergeleitet, verarbeitet und dann wieder an das übrige Netzwerk weitergegeben.
Die Verarbeitung in den Kontextzellen wird exemplarisch am *Jordan-Netz* gezeigt (Abbildung 11.4).

Dieses Netz kann mit dem Netzgenerator durch den Aufruf

```
snns_create_net.pl -j 4:3:1 -n test
```

generiert werden. Das Gewicht γ der Verbindung von der Outputschicht zur Kontextzelle sowie das Gewicht λ der selbstrekurrenten Verbindung der Kontextzelle sind nicht trainierbar, sondern werden mit konstanten Werten voreingestellt. Das Gewicht γ bewertet die Intensität des Ausgabewertes und das Gewicht λ steuert das Erinnerungsvermögen des Netzes. Die Information, die die erste Hiddenschicht erhält, besteht nun aus den Werten der Inputschicht und einem Wert der Kontextzelle, der die Vergangenheit repräsentiert.
Die Aktivierungsfunktion der Kontextzellen ist die Identität. Sei O der Netzoutput, O^K der Output der Kontextzelle, dann gilt für die Ausgaben der Kontextzelle zum Zeitpunkt t:

$$O^K(t) = \begin{cases} O^K(0) & : \quad t = 1 \\ \lambda O^K(t-1) + \gamma O(t-1) & : \quad t > 1. \end{cases}$$

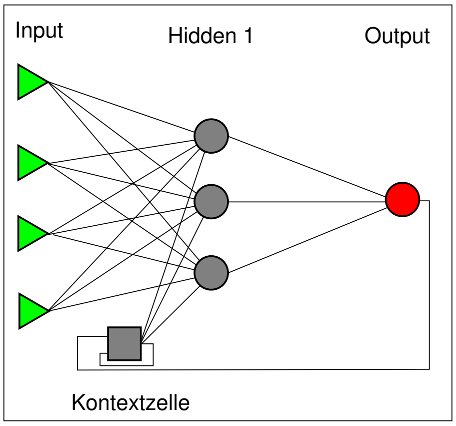


Abbildung 11.4: Jordan-Netz mit Kontextzelle

Aus dieser iterativen Darstellung ergibt sich die explizite Darstellung

$$O^K(t) = \lambda^{t-1} O^K(0) + \gamma \sum_{n=1}^{t-1} \lambda^{n-1} O(t-n).$$

Über den Wert λ in der exponentiell gewichteten Summe läßt sich nun das Erinnerungsvermögen des Netzes steuern. Je kleiner der Wert ist, desto weniger werden vergangene Netzoutputs berücksichtigt.
Um Jordan-Netze mit dem Prognosesystem zu generieren und zu verwenden, müssen einige Parameterwerte in der BP-Steuerungsdatei wie folgt gesetzt sein:

```
PARAM SHUFFLE          FALSE
PARAM BIGNET           Jordan
PARAM RANDOMIZE        OFF
```

Da bei Jordan-Netzen die zeitliche Reihenfolge der Muster wichtig ist, müssen *SHUFFLE* und *RANDOMIZE* deaktiviert werden. Damit der Netzgenerator die Kontextzellen und alle zugehörigen Verbindungen generiert, muß *BIGNET* den Wert `Jordan` besitzen. Außerdem ist zu beachten, daß es spezielle Lernverfahren, Initialisierungs- und Updatefunktionen gibt, die mit Jordan-Netzen angewendet werden müssen. Als Beispiel mögen die Einstellungen zu einem Standard-Backpropagation-Algorithmus dienen.

```
PARAM LEARNREM        Verfahren, ETA, DMAX, TEACHER_FORCING
PARAM LEARNFUNC       JE_BP
PARAM LEARNPARAM      0.2, 0.0, 1.0
PARAM UPDATEFUNC      JE_Order
PARAM INITREM         Initialisierung, a, b, lambda, gamma, psi
PARAM INITFUNC        JE_Weights
PARAM INITPARAM       0.0, 1.0, 0.5, 1.0, 0.5
```

Datei `bp.jordan`: Ausschnitt

Das Lernverfahren bekommt den zusätzlichen Parameter *Teacher Forcing*. Ist der Wert kleiner oder gleich 0.0, so wird der tatsächlich zum gerade angelegten Muster gehörige Output im Rückwärtsschritt des Backpropagation-Algorithmus verwendet, ist er größer oder gleich 1.0, so wird der Netzoutput dazu verwendet, liegen die Werte dazwischen, so wird eine gewichtete Summe von Netzoutput und tatsächlichem Output benutzt. Bei der Initialisierungsfunktion bleibt noch der Parameter ψ zu erklären. Er gibt die initiale Aktivierung der Kontextzellen zum Zeitpunkt $t = 0$ an.

Bei *Elman*-Netzen handelt es sich um modifizierte Jordan-Netze (Abbildung 11.5).

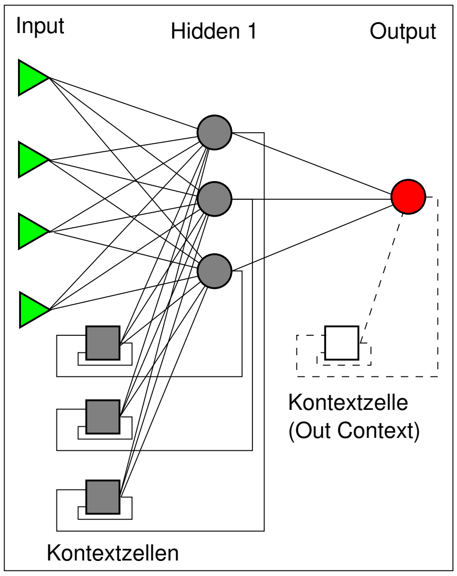


Abbildung 11.5: Elman-Netz mit Kontextzellen

Dieses Netz kann mit dem Netzgenerator durch den Aufruf

```
snns_create_net.pl -e 4:3:1 -c 1 -n test
```

generiert werden. Der Parameter `-c 1` muß übergeben werden, wenn Kontextzellen für die Outputschicht gewünscht sind (siehe die gestrichelten Linien in Abbildung 11.5). Für die weiteren Parameter in der BP-Steuerungsdatei gelten die gleichen Aussagen wie für die Jordan-Netze. Nur der Parameter *BIGNET* muß den Wert `Elman` erhalten. Außerdem gibt es einen zusätzlichen Parameter *OUT_CONTEXT*, der den Wert 1 erhält, wenn Kontextzellen für die Outputschicht gewünscht sind, anderenfalls erhält er den Wert 0.

Auch in Elman-Netzen haben die Kontextzellen eine Speicherfunktion von vergangenen Werten, die aber im Gegensatz zu Jordan-Netzen nicht nur aus der Outputschicht sondern auch aus den Hiddenschichten kommen. Der zeitliche Kontext wird damit in den Neuronen der Hiddenschichten kodiert. Elman-Netze haben gegenüber Jordan-Netzen den Vorteil, daß der Typ des Netzoutputs nicht vom gleichen Typ wie der Netzinput sein muß, da die rekurrenten Verbindungen in den jeweiligen Schichten bleiben.

Kapitel 12

Auswertungen

12.1 zr_error.pl

Um die Vergleichbarkeit statistischer und neuronaler Prognoseverfahren hinsichtlich ihrer Prognosegüte zu gewährleisten, ist es notwendig, die anfallenden Ergebnisse zu aggregieren und für die Analyse in kompakter Form sowohl tabellarisch als auch graphisch darzustellen.
Die anfallenden Daten bestehen aus Werten verschiedener Fehlermaße, die Auskunft über die Abweichung des Sollwertes zum Istwert einer Prognose liefern. Eine bestimmte Prognose ist gekennzeichnet durch das Prognoseverfahren (z.B. lineare Regression), die Parametereinstellungen des Verfahrens (z.B. Wahl der Koeffizienten bei der Regression) und die Zeitreihe, auf die sich diese Prognose bezieht (siehe Abb. 12.1).

Ein einzelnes Datum der zu vergleichenden Datenmenge besteht also aus einem Quadrupel, dessen Dimensionen beschrieben sind durch:

- Zeitreihe,
- Prognoseverfahren,
- Parametereinstellungen,
- Fehlermaß.

Beispiel: Innerhalb des Tegut-Projektes sind 28 Zeitreihen (Artikel), 3 Prognoseverfahren (SNNS, Tegut, Naiv), 9 Fehlermaße und innerhalb einer Testreihe ca. 500 Parametereinstellungen realistisch. Dies ergibt ein Datenaufkommen von 378000 Werten (Fehlerwerte).

Es ist ein Tool entwickelt worden, das diese Menge in eine für die Analyse zweckmäßige Form bringt, ohne dabei Informationen zu verlieren. Im nächsten Abschnitt werden die Möglichkeiten des Tools erläutert. Geschrieben wurde das Tool in der interpretierten Programmiersprache *Perl*, und es ist über Kommandozeilenparameter aufrufbar.

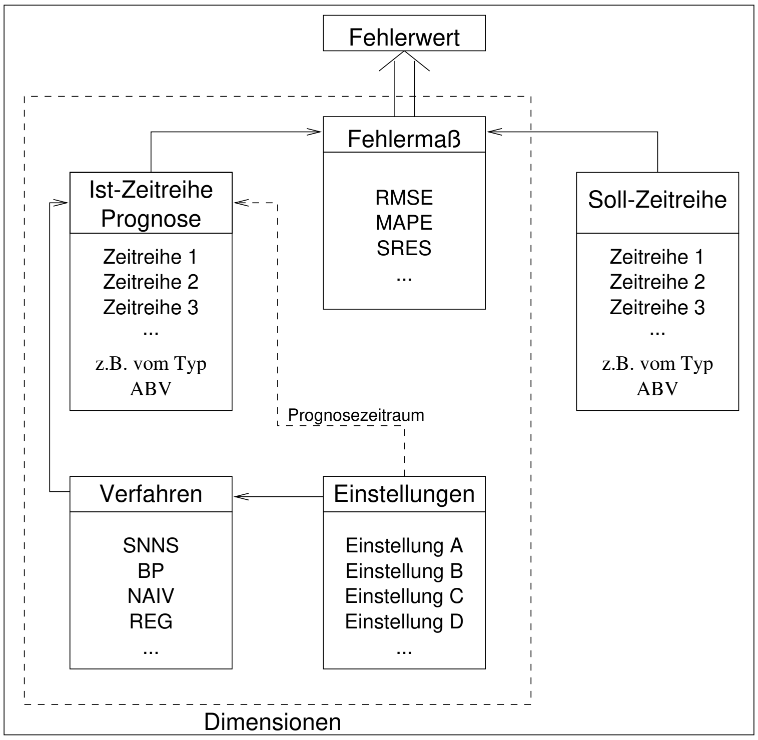


Abbildung 12.1: Die vier Dimensionen

12.1.1 Leistungsmerkmale

1. Ausgabe als ASCII-Tabelle:
   Eine einzelne ASCII-Tabelle enthält als Zeilen- und Spalteneintrag zwei der vier Dimensionen. Die übrigen zwei Dimensionen werden fixiert; somit stellt sie einen zweidimensionalen Schnitt durch den vierdimensionalen Raum dar, den Zeitreihe, Verfahren, Einstellungen und Fehlermaß beschreiben (siehe Tabellen 12.1 und 12.2).

   Der in den Tabellen eingerahmte Wert ist durch dieselben Werte der vier Dimensionen beschrieben, nur die Darstellungsform[1] ist eine andere.

2. Ausgabe als LaTeX-Tabelle:
   Neben der Ausgabe als ASCII-Tabelle, läßt sich auch eine Ausgabe als LaTeX-Tabelle realisieren, die i.a. dazu dient, Tabellen von Fehlerauswertungen innerhalb von LaTeX-Dokumenten homogen aussehen zu lassen, wie dies auch in dieser Arbeit der Fall ist.

3. Ausgabe als PICTeX-Balkendiagramme:
   Der nächste Schritt zur übersichtlichen Handhabung des Datenmaterials ist die graphische

[1]Eine Besonderheit innerhalb des Tegut-Projektes ist die zusätzliche Ausgabe von Packungsgröße und durchschnittlichem Abverkauf — in bezug auf den gesamten zur Verfügung stehenden Erfassungszeitraum — für jeden Artikel.

Tabelle 12.1: Fehlermaß: mape Prognoselauf: Test3

	snns	tegut	naiv
372206	23.426	29.212	43.750
392785	16.340	19.555	35.330
468978	17.374	23.877	32.569
468985	24.063	30.417	37.863
399883	27.109	26.194	44.338
229104	28.738	18.873	26.880
228572	48.599	45.265	36.571
228558	32.266	32.215	36.846
215732	$\boxed{21.508}$	19.064	23.951
AGG	26.603	27.186	35.344

Tabelle 12.2: Zeitreihe: 215732 Prognoselauf: Test3

	snns	tegut	naiv
mean	37.619	37.619	37.619
mre	0.1995	0.2105	0.2281
mape	$\boxed{21.508}$	19.064	23.951
rmse	8.9889	9.1542	9.7211
rmse_m	0.2389	0.2433	0.2584
theil	1.2894	1.3131	1.3944
sres	808	838	945

Darstellung als Balkendiagramm. Dies geschieht mittels des PICTEX-Moduls innerhalb von LATEX, für das lesbarer und übersetzbarer ASCII-Code generiert wird. Wie ein solches Diagramm aussieht, zeigt Abbildung 12.2. Man sieht die Zeile von Tabelle 12.1, die den gerahmten Wert enthält, graphisch dargestellt. Im übrigen können Format und Stil des Diagramms frei gewählt werden.

4. Variable Auswahl einer Datenteilmenge, die durch die vier Dimensionen Zeitreihe, Verfahren, Einstellungen und Fehlermaß gegeben ist:
   Ein wichtiger Aspekt bei der Entwicklung des Tools war es, den Benutzer beliebige Teilmengen der Fehlerwertegesamtheit auswählen zu lassen, so daß in einfacher Weise Abfragen folgender Form möglich sind:

   - Auswertung von neuronalen Verfahren mit verschiedenen Parametereinstellungen,
   - Auswertung einzelner, bestimmter Fehlermaße,
   - Auswertung nur solcher Parametereinstellungen, die einen bestimmten Prognosezeitraum enthalten.

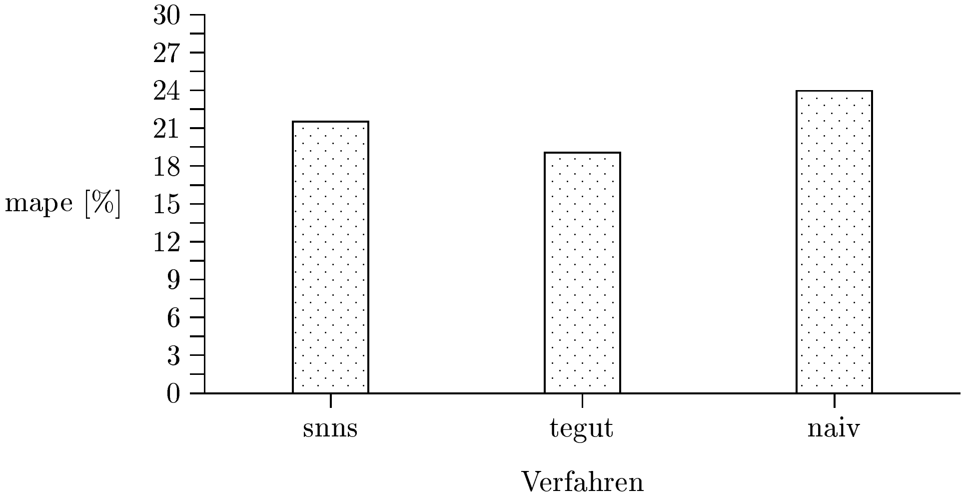


Abbildung 12.2: Zeitreihe: Tegut/AAA/215732, Prognoselauf: diplbsp

Wenn in diesem Zusammenhang von Parametereinstellungen gesprochen wird, so sind unter anderem Parameter bzgl. Netzarchitektur, Topologie oder verwendeter Inputzeitreihen bei neuronalen Verfahren gemeint. Immer aber gehören Parameter, die den Prognosezeitraum betreffen, dazu.

5. Freie Wahl der tabellarischen und graphischen Anordnung dieser Auswahl:
   Ist erst einmal die interessierende Teilmenge spezifiziert worden, so können beliebig viele Ausgabeformen eingestellt werden. In der Tabellenform bedeutet dies für die Zeilen- und Spalteninhalte eine Auswahl aus den drei Dimensionen Verfahren, Zeitreihe und Fehlermaß. Die jeweils dritte Dimension und die Parametereinstellungen werden für jede ausgegebene Tabelle fixiert. In der graphischen Darstellung kann auf der x-Achse eine der Dimensionen *Verfahren*, *Zeitreihe* oder *Parametereinstellungen* abgetragen werden, während die y-Achse die drei übrigen Dimensionen fixiert. In der Abbildung 12.2 sieht man auf der Abszisse die Verfahren und auf der Ordinate einen Fehlerwert, der durch Fixierung der *Zeitreihe* (*215732*), der *Parametereinstellungen* (identifiziert durch den Prognoselaufnamen *diplbsp*) und natürlich durch das *Fehlermaß* (*mape*) gegeben ist.
   Die Tatsache, daß es nicht gestattet ist, die Dimension *Fehlermaß* auf der x-Achse zuzulassen, erscheint sinnvoll, da die verschiedenen Maße nicht untereinander vergleichbar sind.

6. Möglichkeit der Aggregation von Fehlerwerten über mehrere Zeitreihen:
   Sowohl in der tabellarischen als auch in der graphischen Darstellung ist es möglich, einen durchschnittlichen Fehler über mehrere gleichartige Zeitreihen auszugeben. Dieses Feature kommt vornehmlich dem Tegut-Projekt zugute, in dem 28 gleichartige Abverkaufszeitreihen von Cerealien verglichen werden.

7. Aufteilen und Darstellen der Fehlerwerte in zwei Mengen nach dem Vergleichskriterium „Fehler von Verfahren a < Fehler von Verfahren b":

Innerhalb der Tabellen ist es bei genau einer Darstellung möglich (Spalten: *Verfahren*, Zeilen: *Zeitreihen*), die Zeileneinträge so zu sortieren, daß oberhalb einer Trennlinie diejenigen Zeitreihen zu finden sind, bei denen das Vergleichskriterium zutrifft, und unterhalb alle übrigen erscheinen. Somit ergibt sich eine schnelle quantitative Vergleichsmöglichkeit, um zwei Verfahren einzuschätzen. Zusätzlich enthält die Trennlinie in der ASCII-Darstellung eine numerische Information über die Anzahl der Zeitreihen oberhalb der Linie und über die verglichenen Verfahren.

8. Defaulteinstellungen für jedes Projekt und damit Verminderung der Kommandozeilenparameter bei Aufruf des Tools:
   Da sich aufgrund der Möglichkeiten des Tools eine große Zahl an Kommandozeilenparametern ergeben haben, wurde eine Defaultdatei erstellt, in der ein Teil dieser Parameter ausgelagert werden kann. Diese Datei kann für jedes Projekt angelegt werden und enthält hauptsächlich Informationen über die Datenteilmenge und deren Darstellungsform. Näheres wird weiter unten beschrieben.

12.1.2 Aufrufkonventionen

Es werden nun die Kommandozeilenparameter erläutert. Die Verwendung und der alternative Gebrauch von List-Files anstelle der Kommandozeilenparameter werden im nächsten Abschnitt erklärt. Das Tool `zr_error.pl` kennt zwei Aufrufmöglichkeiten:

1. `zr_error.pl [-a|t|p @<p_list>] <comment/idx> <ZR-Typ>`
   `[-force] <von> <bis> <Tick>`
   `[ <komp1/komp2/.../> <Verfahrensliste> <Fehlerliste>`
   `<Darstellungsliste> <Aggregation> <Sortierung> ]`

2. `zr_error.pl [-a|t|p @<p_list>] @<zr_list>`

Die Schalter `a`, `t` und `p` bestimmen den gewünschten Ausgabestil. So steht `a` für die Ausgabe als ASCII-Tabelle, `t` für die Ausgabe als LaTeX-Tabelle und `p` in Zusammenhang mit einem List-File für die Ausgabe als PiCTeX-Balkendiagramm. Die Schalter können beliebig miteinander kombiniert werden.
Der Parameter *ZR-Typ* identifiziert den Zeitreihentyp, auf den sich die Einstellungen der vier Dimensionen beziehen. So kann etwa der Zeitreihentyp, der sich auf Supermarktartikel bezieht, der Abverkauf (`ABV`), der Umsatz (`UMS`) oder auch der Preis (`PRS`) sein. Bei einer CO_2-Zeitreihe hingegen kann er den CO_2-Gehalt (`CO2`) identifizieren. Der Wert des Übergabeparameters ist eine Zeichenfolge, die sich auf Einstellungen in einer Systemdatei des Prognosesystems bezieht. Als Beispiel dienen die Stringwerte in den Klammern.
Tick ist eine Zeichenfolge aus einer Wertemenge, die die Zeiteinheit der Zeitreihe identifiziert. Die Wertemenge umfaßt die Elemente `Monate`, `Quartale`, `Wochen`, `Tage` und die unbestimmte Zeiteinheit `Ticks`. Auch sie sind in einer Systemdatei spezifiziert. Sowohl *Tick* als auch *ZR-Typ* legen die zu berücksichtigenden Zeitreihen fest. Dies können z.B. wöchentliche Abverkäufe von verschiedenen Supermarktartikeln sein oder auch tägliche CO_2-Werte in der Luft.

von und *bis* geben den Start- und Endzeitpunkt des Zeitraums an, auf den sich die Fehlerberechnung stützt. Das bedeutet, für alle verwendeten Verfahren wird der gleiche hier angegebene Zeitraum benutzt. Dies ist aber nicht möglich und führt zu einer Fehlermeldung, wenn Prognosezeitreihen der in der *Verfahrensliste* angegebenen Verfahren sowohl den Startzeitpunkt als auch den Endzeitpunkt nicht beinhalten. Dieser Umstand kann aber mit Hilfe des Schalters `force` umgangen werden. So können zur Berechnung des Fehlers nicht nur die Prognosezeitreihe sondern auch die zeitlich davorliegenden Zeitreihen für Training- und Testmenge eines statistischen oder neuronalen Verfahrens benutzt werden. Das Format der beiden Parameter *von* und *bis* hat eine der Formen `q`*Quartal/Jahr*, `m`*Monat/Jahr*, `w`*Woche/Jahr*, `t`*Tag/Monat/Jahr* oder *Ticks*.
Es werden nun die Kommandozeilenparameter beschrieben, die die vier Dimensionen identifizieren:
komp1/komp2/.../: Bezeichnet eine einzelne Zeitreihe, die durch ihre Komponenten *komp1* bis *kompN* bestimmt wird. Die Komponenten bezeichnen ein physikalisches Verzeichnis auf einem Speichermedium, in dem das Zeitreihenverzeichnis liegt.
Zum Beispiel gibt es innerhalb des Tegut-Projektes drei Komponenten, die ein Zeitreihenverzeichnis identifizieren. Namentlich sind dies etwa der Projektname `Tegut`, der Filialenname `AAA` und ein Artikel `341110`. In dem so erreichten Verzeichnis kann es dann verschiedene Zeitreihentypen geben, die sich auf diesen Artikel beziehen (siehe Parameter *ZR-Typ*). Beim CO_2-Projekt hingegen gibt es nur eine Komponente (nämlich `CO2`), da keine weitere Unterteilung in verschiedene Prognosezeitreihen erfolgt.
Um mehrere Zeitreihen gleichzeitig in die Fehlerberechnung einzubeziehen – dies ist nur im Tegut-Projekt möglich, da es sich zur Zeit um das einzige Projekt mit mehreren Prognosezeitreihen handelt – , kann anstelle der Komponenten ein Stringwert stehen, der eine Gruppe von Zeitreihen umfaßt. Die Wertemenge beinhaltet beispielsweise `SAP`, `LAP`, `SP`, `LP` und `S`, die die 28 bei Tegut zur Verfügung stehenden Zeitreihen in fünf logische Gruppen aufteilt. Darüberhinaus gibt es noch `ALL`, was die Gesamtheit aller Artikel und deren Zeitreihen umfaßt und `AKT`, womit eine beliebige Artikelteilmenge beschrieben werden kann.
Verfahrensliste: Hier werden durch Kommata getrennt Stringwerte für die in der Auswertung in Frage kommenden Prognoseverfahren eingegeben. Hierzu zählen unter anderem die naive Prognose (`Naiv`), das Tegut-Prognoseverfahren (`Tegut`), SNNS-Prognoseverfahren (`SNNS`) oder eine Regression (`REG`). Wird der Wert `ALL_V` angegeben, so werden alle Verfahren berücksichtigt, die das Tool kennt.
`Fehlerliste`: Ähnlich der Verfahrensliste werden hier verschiedene Fehlermaße angegeben, die in der Auswertung vorkommen sollen. Es sind dies derzeit `MEAN`, `MRE`, `MAPE`, `RMSE`, `RMSE_M`, `THEIL`, `DETERM`, `SRES` oder `ALL_F`. Auch hier bedeutet `ALL_F` die Verwendung aller bekannten Fehlermaße.
comment/idx: Die Parametereinstellungen eines Verfahrens werden durch einen Stringwert angegeben, der dem Prognoselauf, der zuvor ja stattgefunden haben muß, als Kommentar zugeordnet wird. Alternativ ist auch die Angabe eines numerischen Index möglich, da Kommentar und Index eineindeutig zugeordnet werden können.
Der Dateiname einer Prognosezeitreihe setzt sich zusammen aus dem Zeittick, dem Zeitreihentyp, dem verwendeten Verfahren und einem eindeutigen Index. Zu finden ist diese Datei dann in einem Verzeichnis, das durch die einzelnen Komponenten *komp1* bis *kompN* bestimmt ist.
Mit der *Darstellungsliste* werden nun in den tabellarischen Ansichten verschiedene Ausgabefor-

men der vier Dimensionen gewählt. Durch Kommata getrennt werden jeweils zwei Characterwerte benutzt, um den Spalten- und Zeileneintrag einer Tabelle festzulegen. So gibt es `z` für Zeitreihe, `v` für Verfahren und `f` für Fehlermaß. So bedeutet `vz` etwa, Darstellung der Verfahren in den Spalten und Darstellung der Zeitreihen in den Zeilen. Für jeden Wert, der in der jeweils fehlenden Dimension vorkommt (in dem Beispiel sind es die Fehlermaße), wird eine Tabelle ausgegeben, so daß ein zweidimensionaler Schnitt entsteht. Diese Tabellen werden unter einem Dateinamen, den die Dimension der Parametereinstellung angibt, und der Extension `.tab` physikalisch gespeichert.

Aggregation: Eine Aggregation und Mittelung von Fehlerwerten ist nur für gleichartige Zeitreihen möglich und sinnvoll. Dieser Parameter kann die Werte `-1` für tabellarische Darstellung ohne Aggregation, `0` für Darstellung mit Aggregation und `1` für alleinige Darstellung der Aggregation annehmen. Als Beispiel möge nochmal die Tabelle 12.1 dienen mit Augenmerk auf den Zeileneintrag `AGG`. Hier hatte der Parameter `Aggregation` den Wert `0`.

Sortierung: Dieser Parameter erlaubt es, die Zeitreihen innerhalb einer Tabelle nach dem Kriterium „Fehlerwert von Verfahren a < Fehlerwert Verfahren b" zu sortieren und in zwei Gruppen aufzuteilen. Dies ist nur in der Darstellungsform `vz` möglich, in der die Verfahren in der Spalte und die Zeitreihen in der Zeile eingetragen sind (siehe Tabelle 12.3). Die Angabe des Parameters erfolgt mit dem Stringwert `comp`_*VerfahrenA*_*VerfahrenB*.

Tabelle 12.3: Fehlermaß: mape Prognoselauf: Test3 Sortierung: comp_snns_tegut

	MEAN	PCK	snns	tegut	naiv
372206	30.52	7	23.426	29.212	43.750
392785	57.55	12	16.340	19.555	35.330
468978	80.88	14	17.374	23.877	32.569
468985	58.20	14	24.063	30.417	37.863
399883	33.64	8	27.109	26.194	44.338
229104	184.90	15	28.738	18.873	26.880
228572	34.76	12	48.599	45.265	36.571
228558	51.64	15	32.266	32.215	36.846
215732	37.62	12	21.508	19.064	23.951
AGG			26.603	27.186	35.344

Oberhalb der Trennungslinie befinden sich die Zeitreihen, mit denen das Verfahren SNNS besser abgeschnitten hat als das Verfahren Tegut. Unterhalb der Linie verhält es sich umgekehrt.

12.1.3 Eingabedateien

Defaultdatei

Da das Tool `zr_error.pl` 13 Übergabeparameter kennt, ist es sinnvoll, solche Parameter, die sich selten ändern, in eine Defaultdatei zu schreiben. Eine solche Datei existiert un-

abhängig für jedes Projekt in seinem entsprechenden `Include`-Verzeichnis unter dem Dateinamen `zr_error.default`. Folgende unter 12.1.2 beschriebenen Parameter können dort eingetragen werden:
`Zeitreihen`, **`Verfahrensliste`**, **`Fehlerliste`**, **`Darstellungsliste`**, **`Aggregation`** und **`Sortierung`**.
Diese Werte stehen in einer Zeile getrennt durch Whitespaces.
Werden bei einem Aufruf des Tools einzelne Parameter ausgelassen, so tritt der entsprechende Defaultwert in Kraft. Das bedeutet auch, daß die Reihenfolge dieser Parameter in der Übergabezeile willkürlich ist. Unterschieden werden sie anhand ihrer Syntax, die für jeden Parameter eindeutig ist. So kann der Aufrufaufwand auf ein Minimum reduziert und vereinfacht werden.
Weitere Abschnitte der Defaultdatei werden im folgenden innerhalb der zugehörigen Sektion abgehandelt.

zr_list-File

Es wird nun der alternative Aufruf des Tools mit Hilfe eines List-Files beschrieben, durch das der Aufruf weiter vereinfacht wird. Dies macht sich vor allem im Batch-Betrieb bemerkbar. Ein solches List-File besteht aus beliebig vielen Zeilen, von denen eine Zeile den Inhalt der Kommandozeilenparameter enthält, die man tatsächlich übergeben hätte. Auch hier greift der Mechanismus, der im Abschnitt über die Defaultdatei beschrieben wurde. Die Angabe eines List-Files ist in etwa mit einem mehrfachen Aufruf des Tools zu vergleichen, mit dem Unterschied, daß sämtliche Fehlerwerte eines Aufrufes erhalten bleiben und somit zur weiteren Verarbeitung für Tracktabellen (siehe Abschnitt 12.1.4) und PICTEX-Ausgaben zur Verfügung stehen.

pic_list-File

Die Übergabeparameter für die PICTEX-Ausgabe sind immer in einem List-File zu finden. Jeder Zeileneintrag in diesem File entspricht der Ausgabe genau eines Balkendiagramms. Die Parameter innerhalb einer Zeile werden durch Whitespaces getrennt.
Der erste Parameter gibt an, welche Dimension auf der x-Achse des Balkendiagramms abgetragen wird. Möglich sind `Z` für Zeitreihe, `V` für Verfahren und `C` für Parametereinstellungen, die – wie weiter oben erklärt – mit einem Kommentar identifiziert werden. Das Abtragen von Fehlermaßen auf der x-Achse macht keinen Sinn, da unterschiedliche Fehlermaße nicht direkt vergleichbar sind.
Der zweite Parameter besteht aus einem durch Kommata getrennten Wertetripel, welches jeweils einen Wert der Dimensionen enthält, die nicht durch den ersten Parameter beschrieben sind. Wenn etwa der erste Parameter ein `Z` für Zeitreihe ist, dann müssen Werte für ein Fehlermaß, ein Verfahren und Parametereinstellungen angegeben werden. Hier handelt es sich also um die graphische Darstellung eines zweidimensionalen Schnittes durch den schon beschriebenen vierdimensionalen Raum. Die Wertemenge des Wertetripels ist identisch mit der unter Abschnitt 12.1.2 beschriebenen.
Der dritte Parameter ist ein boolscher Wert, der angibt, ob eine Legende zum Diagramm ausgegeben werden soll oder nicht. Dies ist notwendig für eine übersichtliche Ausgabe entlang der

x-Achse des Diagramms.
Die *pic-Liste* kann nur auf Informationen zugreifen, die durch eine *zr-Liste* gegeben ist. Die Zusammenhänge beschreibt die Abbildung 12.3.

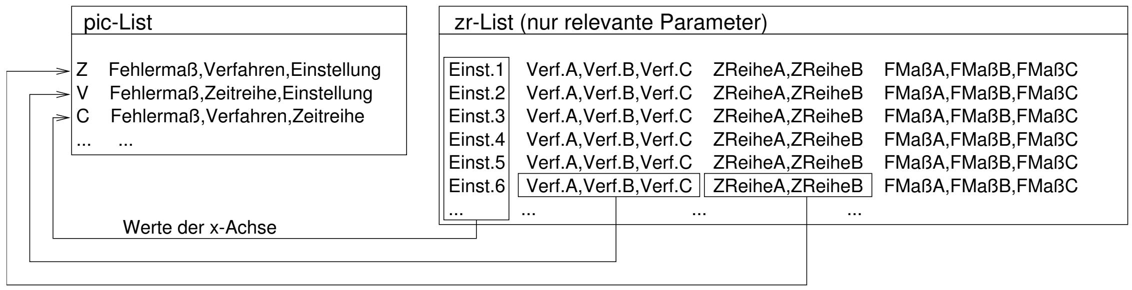


Abbildung 12.3: Zusammenhang zwischen *zr-List* und *pic-List*

In der projektbezogenen Defaultdatei gibt es auch einen Abschnitt, um die Form der PICTEX-Diagramme zu modifizieren. So können Höhe und Breite des Diagramms sowie die maximale Balkenbreite angegeben werden, ob die Balken an der y-Achse beginnen oder versetzt sein sollen, und die Skaleneinteilung der y-Achse kann geändert werden.
Um den PICTEX-Quelltext flexibel in Dokumente einzubinden, kann man zwei LATEX-Quellen angeben, die vor und hinter dem PICTEX-Quelltext erscheinen. Als Standard sind die LATEX-Quellen derart, daß eine schnelle Kompilierung erfolgen kann.

12.1.4 Ausgabedateien

Je nach gewünschtem Ausgabestil werden beliebige Kombinationen von ASCII- und LATEX-Tabellen oder PICTEX-Diagrammen erstellt. Die Dateinamen der Tabellen ergeben sich durch den Übergabeparameter `comment/idx` mit der Extension `.tab` bei ASCII-Tabellen oder mit der Extension `.tex` bei LATEX-Tabellen. PICTEX-Diagramme werden unter dem Präfix `pic.`, dem Stringwert von `p_list` und der Extension `.tex` abgelegt. Besonderes Augenmerk gilt noch der Tracktabelle, deren Dateinamen sich aus dem Präfix `track.`, dem Stringwert von `zr_list` und der Extension `.tab` zusammensetzt. Mit ihrer Hilfe können bei einem großen zr_list-File Fehlerwerte zweier Verfahren sortiert nach Fehlerwert des ersten Verfahrens in Hinblick auf die Parametereinstellungen verglichen werden. Werden etwa durch ein 100-zeiliges zr_list-File 100 Tabellen mit den Namen der jeweiligen *comments*, die ja den Parametereinstellungen des jeweiligen Prognoselaufes entsprechen, geschrieben, so kann die beste Einstellung sofort in der Tracktabelle abgelesen werden, ohne die einzelnen Tabellen durchsuchen zu müssen. Es folgt ein Auszug einer solchen Tabelle.

```
         mape,snns,Sunspots/   mape,naiv,Sunspots/
T1I3H8        39.492                 68.273
T1I3H6        40.005                 68.273
T1I8H3        40.606                 68.273
T1I4H5        43.605                 68.273
T1I4H7        44.824                 68.273
T1I8H5        44.927                 68.273
```

Datei `track.tab`

Welches Fehlermaß, welche Zeitreihe und welches Verfahren in der Tracktabelle erscheint, kann in der projektspezifischen Defaultdatei angegeben werden.

12.1.5 Anwendungsbeispiele

Vergleich verschiedener Prognoseverfahren mit einer Zeitreihe am Beispiel CO2

Gegeben seien ein Prognoselauf mit SNNS, mit BP und der Regression. Bei den Prognoseläufen wurde dazu jeweils derselbe *comment* vergeben. Sei dies zum Beispiel `lauf1`. Prognostiziert wurde jeweils der CO_2-Gehalt im Zeitraum vom 01.01.1997 bis zum 10.01.1997.
Gewünscht ist eine Ausgabe der Fehlermaße *mre*, *rmse* und *mape* dieser Verfahren in einer Tabelle. Des weiteren soll für jedes Fehlermaß auch eine graphische Ausgabe erfolgen mit den Verfahren auf der x-Achse und dem jeweiligen Fehlermaß auf der y-Achse.
Für die graphische Ausgabe werden folgende Einträge — jede Zeile entspricht dabei einem PICTEX-Diagramm — in eine Datei geschrieben:

```
V  mre,lauf1,CO2/ 0
V rmse,lauf1,CO2/ 0
V mape,lauf1,CO2/ 0
```

Datei `pic.lst`

Dann erfolgt der Aufruf von `zr_error.pl` folgendermaßen:

```
zr_error.pl -ap @pic.lst lauf1 CO2 t01/01/97 t10/01/97 Tage
            CO2/ SNNS,BP,REG mre,rmse,mape vf -1 comp_snns_reg
```

Es entstehen dann die Datei der ASCII-Tabelle `lauf1.tab` und die Datei mit LaTeX-Quelltext namens `pic.lauf1.tex`, die alle drei Balkendiagramme enthält.

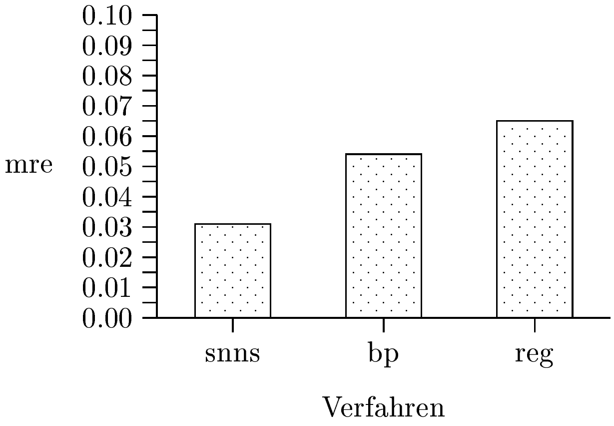


Abbildung 12.4: Zeitreihe: CO2, Prognoselauf: lauf1

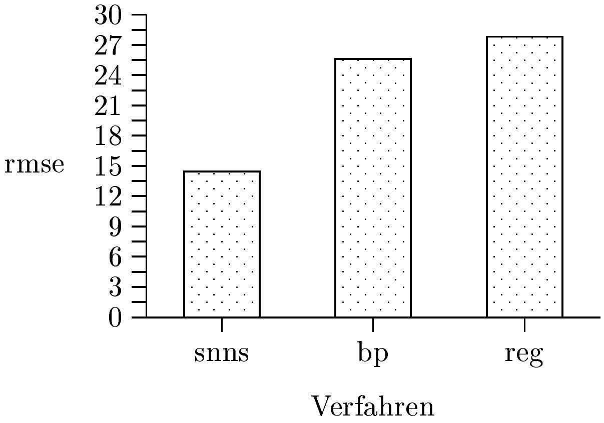


Abbildung 12.5: Zeitreihe: CO2, Prognoselauf: lauf1

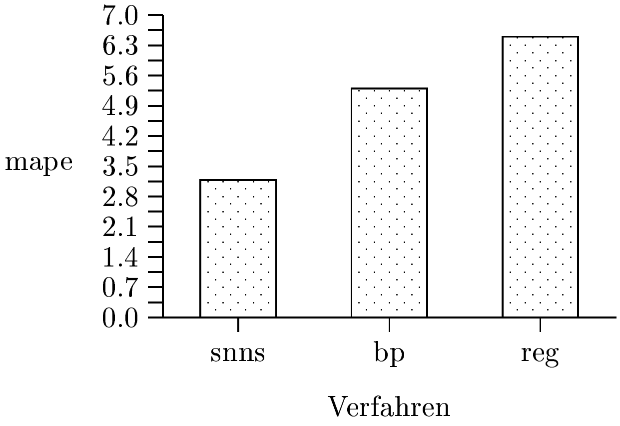


Abbildung 12.6: Zeitreihe: CO2, Prognoselauf: lauf1

```
Parameter: dipl, CO2, t12/11/93, t31/12/93, Tage, CO2/,
snns,bp,reg, mre,rmse,mape, vf, -1, comp_snns_reg

CO2/; 12.11.1993 - 31.12.1993
        |    snns|      bp|     reg
--------+--------+--------+--------
mre     |  0.0312|  0.0543|  0.0646
rmse    |  14.463|  25.611|  27.820
mape    |  3.1800|  5.2995|  6.4947
```

Datei `lauf1.tab`

Vergleich verschiedener Prognoseläufe mit einer Zeitreihe am Beispiel Sunspots

Gegeben seien fünf verschiedene Prognoseläufe mit SNNS, die sich durch ihre Lernrate im Backpropagation-Algorithmus unterscheiden. Prognostiziert wurde der prozentuale Sonnenfleckenanteil von den Monaten Januar bis Oktober des Jahres 1988. Die *comments* dieser Läufe seien `eta02`, `eta04`, `eta06`, `eta08` und `eta10`.
Zu untersuchen seien die Fehlermaße *rmse* und *mape* der Zeitreihe in Abhängigkeit der Prognoseläufe. Zusätzlich soll es eine graphische Ausgabe geben, mit den Parametereinstellungen (comments) abgetragen an der x-Achse.
Für die graphische Ausgabe werden folgende Einträge in eine Datei geschrieben:

```
C rmse,snns,Sunspots/ 1
C mape,snns,Sunspots/ 1
```

Datei `pic2.lst`

Zur Erinnerung: Die 1 am Ende jeder Zeile sagt lediglich aus, daß unter dem Diagramm eine Legende geschrieben werden soll, weil die *comments* sich ansonsten überschnitten, wenn sie an der x-Achse abgetragen würden. Da mit einem Aufruf von `zr_error.pl` ohne ein *zr_list*-File nur ein *comment* verarbeitet werden kann, ist nun auch ein solches notwendig, beispielsweise mit dem Namen `eta.lst`. In dieses werden folgende Einträge geschrieben:

```
eta02 SUN  m01/88 m10/88 Monate Sunspots/ SNNS rmse,mape vf -1 comp_snns_naiv
eta04 SUN  m01/88 m10/88 Monate Sunspots/ SNNS rmse,mape vf -1 comp_snns_naiv
eta06 SUN  m01/88 m10/88 Monate Sunspots/ SNNS rmse,mape vf -1 comp_snns_naiv
eta08 SUN  m01/88 m10/88 Monate Sunspots/ SNNS rmse,mape vf -1 comp_snns_naiv
eta10 SUN  m01/88 m10/88 Monate Sunspots/ SNNS rmse,mape vf -1 comp_snns_naiv
```

Datei `eta.lst`

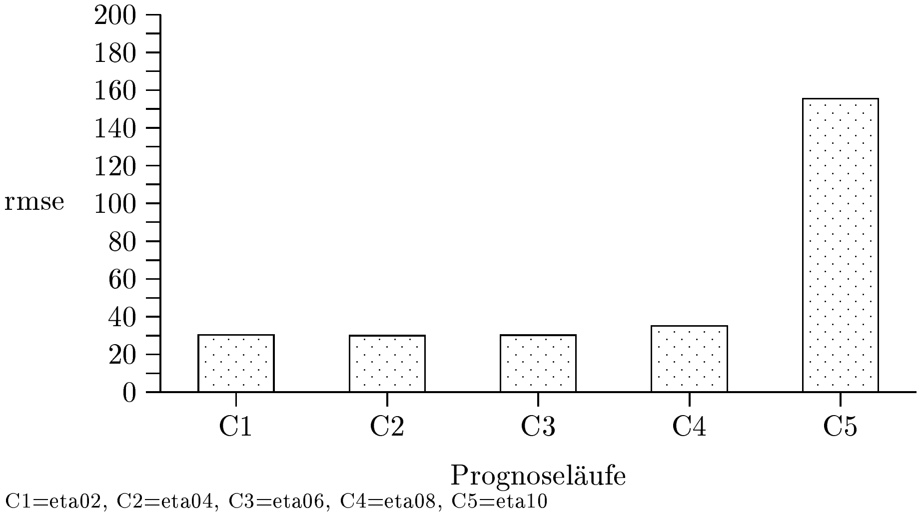


Abbildung 12.7: Verfahren: SNNS, Zeitreihe: SUNSPOTS

Natürlich können die letzten sechs Parameter einer Zeile in die entsprechende Defaultdatei eingetragen werden, so daß sich der Schreibaufwand verringert. Der letzte Parameter hat in diesem Beispiel keine Bedeutung und erhält einen beliebigen Eintrag. Der Aufruf von `zr_error.pl` ist nun der folgende:

```
zr_error.pl -ap @pic2.lst @eta.lst
```

Es entstehen nun fünf Dateien mit ASCII-Tabellen, in denen das Verfahren in der Spalte und die Fehler in der Zeile zu finden sind, namens `eta02.tab`, `eta04.tab`, `eta06.tab`, `eta08.tab` und `eta10.tab`. Exemplarisch ist im folgenden die Datei `eta02.tab` angegeben. Die Balkendiagramme sind in der Datei `pic.eta.tex` zu finden.

```
Parameter: eta02, SUN, m01/88, m10/88, Monate, Sunspots/,
snns, rmse,mape, vf, -1, comp_snns_naiv

Sunspots/; 01/1988 - 10/1988
        |    snns
--------+--------
rmse    |  30.304
mape    |  16.019
```

Datei `eta02.tab`

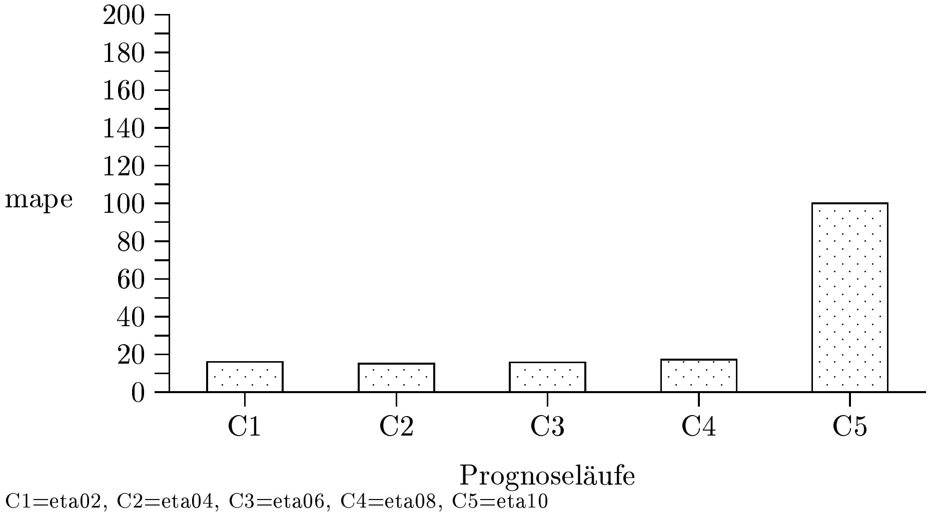


Abbildung 12.8: Verfahren: SNNS, Zeitreihe: SUNSPOTS

Vergleich mehrerer Zeitreihen desselben Typs am Beispiel Tegut

Gegeben sei ein Prognoselauf namens `test` mit SNNS über alle Artikel der Gruppe LAP, der Langsamdreher mit Aktionen und Preisveränderungen. Der Prognosezeitraum sei durch die 37. Woche bis zur 46. Woche des Jahres 1997 gegeben. Es soll in Hinblick auf das Fehlermaß *mape* die SNNS-Prognose mit der Tegut-Prognose innerhalb einer Tabelle verglichen werden.
Dazu reicht folgender Aufruf des Tools:

```
zr_error.pl test ABV w37/97 w46/97 Wochen LAP SNNS,Tegut mape vz 1 comp_snns_tegut
```

Es entsteht eine Datei `test.tab`, die die ASCII-Tabelle enthält. Die Verfahren SNNS und Tegut sind in den Spalte, die Zeitreihen der Artikelgruppe LAP in den Zeilen zu finden. Zusätzlich werden die Fehlerwerte über die Zeitreihen noch gemittelt ausgegeben.

```
Parameter: test, ABV, w37/97, w46/97, Wochen, LAP,
snns,tegut,naiv, mre,mape,rmse,rmse_m,theil,sres,
vz,fz, 0, comp_snns_tegut

mape; 37/1997 - 46/1997
        |    MEAN| PCK|    snns|   tegut|    naiv
--------+--------+----+--------+--------+--------
341110  |   6.430|  18|   34.72|  42.063|   20.04
341127  |   5.311|  18|    0.00|  12.222|   32.22
341134  |  14.430|  18|   27.98|  28.788|   40.10
362238  |  10.887|  12|   39.85|  41.239|   27.14
392914  |   5.179|  20|   16.67|  23.333|   10.83
567411  |   3.742|   8|   19.84|  57.143|   51.59
```

```
952576  |  13.609|  20|    8.61|  12.917|   16.94
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^#7^snns<tegut
215749  |   8.881|  12|   22.22|  21.429|   30.16
215718  |  12.093|  14|   32.28|  30.159|   21.69
215695  |   5.464|  10|  106.67|  88.333|  101.67
--------+--------+----+--------+--------+--------
  AGG   |        |    |   30.88|  35.763|   35.24
```

Datei `test.tab`

Kapitel 13

Ergebnisse der Tegut-Abverkaufsprognosen

In diesem Kapitel werden Ergebnisse von Abverkaufsprognosen von 28 zu einer Produktgruppe gehörigen Artikeln einer Supermarktfiliale vorgestellt. Dabei handelt es sich sowohl um qualitative Ergebnisse, daß heißt, allgemeine, diese Produktgruppe betreffenden Aussagen angefangen bei der Parameterwahl bis hin zu den relevanten Einflußgrößen, als auch um quantitative Ergebnisse, die sich in den Werten der betrachteten Fehlermaßen widerspiegeln.
Wenn t der aktuelle Zeitpunkt (Woche) ist und die Abverkaufszahlen bis zum Zeitpunkt $t-1$ vorliegen, werden Prognosen immer für den Zeitpunkt $t+1$ gemacht, da in der aktuellen Woche Dispositionsaufgaben (z.B. Bestimmung der Bestellmengen für Woche $t+1$) durchzuführen sind. Aus diesem Grund muß der Parameter `ADVANCE` den Wert 2 erhalten.
Sämtliche Prognosen werden unter dem Prognosesystem mit dem Verfahren *SNNS* ausgeführt, wobei als direkter Vergleich die sogenannte *Tegut-Prognose* — eine statistische Methode, die bisher im Supermarkt Verwendung findet — und die *Naive Prognose*, die als Prognosewert für den Zeitpunkt $t+1$ den Wert zum Zeitpunkt $t-1$ nimmt, dienen.

13.1 Beschreibung der Daten

13.1.1 Artikeldaten

Bei den vorliegenden Daten handelt es sich um wöchentliche Abverkaufswerte von 28 Artikeln aus der Produktgruppe der Cerealien, die aus einer Filiale der Supermarkthandelskette *Tegut* stammen. Fortlaufende Werte existieren seit Kalenderwoche 36/1994. Daneben gibt es zwei weitere zur Verfügung gestellte Attribute der Artikel, die sich in der Zeit ändern können. Zum einen handelt es sich um das Attribut „Artikel ist in Woche x einer *Aktion* ausgesetzt", zum anderen das Attribut „Artikel erfährt in Woche x eine *Preisänderung*". Bei einer *Aktion* kann es sich um eine Sonderpreisaktion oder um eine besonders exponierte örtliche Darstellung eines Artikels handeln. Ob eine Aktion in Woche x stattfindet oder nicht, wird auf Zeitreihenebene durch binäre Werte, wie z.B. 1 und 0, identifiziert. Die Preisänderungen werden durch eine Trendzeitreihe beschrieben, die drei Werte, wie z.B. -1 (Preis fällt), 0 (Preis konstant) und 1 (Preis steigt), annehmen kann. Die Wertemenge bei Binär- und Trendzeitreihen hängt von der gewählten Skalierung dieser Zeitreihen im *MODELL*-File ab.

13.1.2 Globale Daten

Neben den von *Tegut* zur Verfügung gestellten Daten wird als weitere wöchentliche Zeitreihe eine des binären Typs verwendet, um anzugeben, ob in Woche x Feiertage auftreten oder nicht.

13.1.3 Gruppierung der Artikel

Es hat sich als sinnvoll erwiesen, die Artikel in Gruppen einzuteilen, die gleiche Eigenschaften besitzen. So können die 28 Artikel zunächst einmal in sogenannte *Schnelldreher* und *Langsamdreher* eingeteilt werden. *Schnelldreher* sind solche Artikel, deren durchschnittliche wöchentliche Mindestbestellmenge kleiner ist als ihr wöchentlicher Abverkauf. Bei *Langsamdrehern* verhält es sich umgekehrt.
Des weiteren unterliegen nicht alle 28 Artikel Aktionen und Preisänderungen, so daß an Hand dieser Kriterien feiner gruppiert werden kann. Auf diese Art ergeben sich fünf[1] Gruppen:

- S: Schnelldreher ohne Aktionen und ohne Preisänderungen,
- SP: Schnelldreher ohne Aktionen und mit Preisänderungen,
- LP: Langsamdreher ohne Aktionen und mit Preisänderungen,
- SAP: Schnelldreher mit Aktionen und mit Preisänderungen,
- LAP: Langsamdreher mit Aktionen und mit Preisänderungen.

Es ergeben sich die in Tabelle 13.1 vorliegenden Anzahlen in den einzelnen Gruppen.

Tabelle 13.1: Artikelanzahlen in den Gruppen

S	1
SP	2
LP	6
SAP	9
LAP	10

Die Gruppierung hat den weiteren Vorteil, daß nur fünf anstatt 28 Neuronale Netze trainiert werden müssen und somit der Bearbeitungsaufwand drastisch sinkt. Neu hinzukommende Artikel

[1]Die Gruppe der reinen Langsamdreher (L) gibt es unter den 28 Artikeln nicht.

können sofort den verschiedenen Gruppen zugeordnet werden, ohne daß ein weiteres Training erfolgen muß.

13.2 Prognoseläufe

Durch die mit *SNNS* durchgeführten Prognoseläufe soll festgestellt werden, ob sich aus gegebenen Abverkaufszahlen zukünftige prognostizieren lassen und inwieweit weitere Einflußfaktoren wie die schon beschriebenen Aktions-, Preiszeitreihen — sofern diese bei dem jeweiligen Artikel vorhanden sind — und Feiertagszeitreihen die Prognose signifikant verbessern. Als Güteziel wird eine Verbesserung der Prognose mit *SNNS* gegenüber der *Tegut*-Prognose postuliert. Die naive Prognose dient als zusätzlicher Vergleichsmaßstab. Die *Tegut*-Prognose und weitere Einzelheiten zu den vorliegenden Daten wird in [Thi98] erläutert.

13.2.1 Zeiträume

Prognostiziert werden Abverkäufe in sechs Zeiträumen von je zehn Wochen ab Kalenderwoche 50/96 bis Kalenderwoche 05/98, die hier als Tabelle dargestellt sind. Es wird damit ein Zeitraum von über einem Jahr untersucht, in dem genügend Feiertage und bei den entsprechenden Artikeln auch Aktionen und Preisveränderungen vorkommen.

Tabelle 13.2: Prognosezeiträume

Zeitraum	Kalenderwochen
T1	50/96 – 07/97
T2	08/97 – 17/97
T3	18/97 – 27/97
T4	28/97 – 37/97
T5	38/97 – 47/97
T6	48/97 – 05/98

Als Trainings- und Testmenge stehen die Daten von Kalenderwoche 36/94 bis zum jeweiligen Beginn des Prognosezeitraums zur Verfügung unter Berücksichtigung des *ADVANCE*-Wertes.

13.2.2 Parametereinstellungen der Prognoseläufe

Für die 28 Artikel wurden in der Testphase über 45000 Prognoseläufe gestartet, die im ganzen bei einer Trainingszyklenzahl von 1000 circa drei Wochen ununterbrochene Rechenleistung erforderten (HP Apollo 9000/730). Bei den Parametereinstellungen wurde der Schwerpunkt auf die Wahl der eingehenden Zeitreihen und deren Tiefe gelegt. Tabelle 13.3 zeigt die verwendeten Anzahlen.

Tabelle 13.3: Art und Anzahl eingehender Zeitreihen

Zeitreihe	Neuronenanzahl	Abkürzung
Abverkauf	3 – 5	A3 – A5
Aktion	1 – 2	K1 – K2
Preis	1 – 2	P1 – P2
Feiertag	1 – 3	F1 – F3
Jahressaison	1	J1

Bei den eingesetzten Lernverfahren handelt es sich um Backpropagation-Verfahren, zum einen ein Verfahren mit Momentumterm und modifizierter Zielfunktion (siehe Kapitel 9), zum anderen um Resilient Propagation (kurz Rprop). Diese Verfahren werden jeweils bei Netztopologien mit und ohne Vorschicht verwendet. Tabelle 13.4 erklärt die abgekürzte Schreibweise.

Tabelle 13.4: Verwendete Lernverfahren

Lernverfahren	Vorschicht	Abkürzung
BPM_ln_cosh	nein	L
BPM_ln_cosh	ja	LX
Rprop	nein	R
Rprop	ja	RX

Alle oben genannten Abkürzungen werden für die Vergabe des Comments benutzt, um dadurch schnell Informationen über Zeitraum, Inputneuronen und Lernverfahren zu erhalten. So bezeichnet etwa `T4A5K1P2F1J1sRX` einen Prognoselauf über Zeitraum T4 mit insgesamt 10 Inputneuronen (5 Abverkauf, 1 Aktion, 2 Preis, 1 Feiertag, 1 Jahressaison). Rprop ist das eingesetzte Lernverfahren und die Netztopologie besitzt eine Vorschicht. Der Buchstabe `s` im Comment bezieht sich auf die Skalierung der Abverkaufszeitreihe (hier `MY_SIGMA`-Skalierung).

Das folgende BP-File soll exemplarisch zeigen, welche Einstellungen diesen Prognoseläufen zugrunde liegen:

```
MEASURE Wochen                  # Zeitraum der Prognose festlegen
START 36/94
END  07/97
ADVANCE 2                       # Luecke zwischen letztem Wert und Prognose

#--------------------allgemeine Daten-----------------------
PARAM EPOCHEN      1000         # Anzahl der Epochen
PARAM ERR            -4         # Fehlerwerte alle N Epochen
PARAM MINEPOCHEN     50         # Mindestanzahl der Epochen
PARAM SHUFFLE      TRUE
PARAM MAXERROR       10
#-----------------------------------------------------------
```

```
#--------------------Net_Func-----------------------------

PARAM LEARNREM    Verfahren, ETA, IMP, FLATSPOT, DMAX, GLAETTUNG, NEG_AMP, POS_AMP
PARAM LEARNFUNC   BPM_ln_cosh
PARAM LEARNPARAM  0.3, 0.1, 0.0, 0.0, 10.0, 0.12, 0.12
PARAM UPDATEFUNC  Topological_Order
PARAM INITREM     Initialisierung, a, b
PARAM INITFUNC    Randomize_Weights
PARAM INITPARAM   0.5, -0.5
#--------------------------------------------------------

#----------------Netztopologie----------------------------
PARAM BIGNET       FeedForward  # FeedForward
PARAM HIDDEN          5
PARAM OUTPUT          1
PARAM SHORTCUT       I-O
#--------------------------------------------------------

PARAM RANDOMIZE  90    # Testpaare aus den letzten 90%
PARAM TEST_PAIRS 30%   # Anzahl der Testpaare
PARAM VAL_PAIRS  10    # Anzahl der Validierungspaare

# diese Zeitreihe wird prognostiziert
OUTPUT $KOMP1 $KOMP2 $KOMP3 ABV.SIG4CUT0

# Benutzte Zeitreihen
UNKNOWN 5 $KOMP1 $KOMP2 $KOMP3 ABV.SIG4CUT0
USE     1 $KOMP1 $KOMP2 $KOMP3 AKT.BIN
USE     1 $KOMP1 $KOMP2 $KOMP3 PRS.TRD0
USE     1 Global Feiertage Hessen.BIN
```

Datei `BP.T1A5K1P1F1sL`

Über die Parametereinstellungen, die den Abkürzungen entnommen werden können, soll der Einfluß der Inputneuronen und des eingesetzten Lernverfahrens auf die Prognosegüte untersucht werden. Die Parameter des jeweiligen Lernverfahrens sind fest gewählt und ändern sich über alle Läufe nicht.

13.3 Ergebnisse

Einige qualitative Ergebnisse wurden schon im Rahmen eines Praktikums im Wintersemester 96/97 gefunden. Diese werden hier zusammengefaßt beschrieben. Rekurrente Netztopologien, wie sie als Jordan- und Elman-Topologien in *SNNS* implementiert sind, haben sich nicht als vorteilhafter gegenüber einfachen dreischichtigen Feed-Forward-Netzen herausgestellt, so daß diese als Topologie in den Prognoseläufen keine Verwendung finden.

Als vorteilhaft hingegen haben sich Shortcutverbindungen von der Input- zur Outputschicht herausgestellt, die die linearen Anteile des Prognosemodells darstellen, so daß Shortcutverbindungen in allen Läufen benutzt werden.
Bezüglich der Skalierungen der Zeitreihen haben sich gegenüber linear skalierter die `MY_SIGMA`-skalierten Zeitreihen durchgesetzt. Es finden sich somit nur derartig skalierte Abverkaufszeitreihen in den Prognoseläufen.
Um zu zeigen, wie gut die besten getesteten Netztopologien prognostizieren können, sind die geringsten sich ergebenden MAPE-Fehler in Tabelle 13.5 eingetragen. Bei diesem Fehler handelt es sich um die mittlere prozentuale Abweichung vom Sollwert. Um genau einen Fehler für alle Artikel einer Gruppe angeben zu können, wird der Mittelwert über die MAPE-Fehler aller Artikel einer Gruppe berechnet. Es zeigt sich, daß bei den Gruppen S, SP, SAP und LP in mindestens fünf von sechs Zeiträumen eine Topologie gefunden werden kann, die bessere Ergebnisse liefert als die Tegut-Prognose. Nur die Gruppe LAP zeigt ein deutlich schlechteres Ergebnis, da sie in drei von sechs Zeiträumen einen größeren Fehler aufweist. Die verwendeten Netztopologien, die zu diesen Fehlern führen, sind nicht zwingend gleich innerhalb der Gruppen für jeden Zeitraum. Für den praktischen Gebrauch sollte aber genau eine Netztopologie pro Gruppe verwendet und beibehalten werden.

Tabelle 13.5: Beste Prognosegüten (MAPE) in den Zeiträumen T1 – T6

Cluster	Prognose	Zeitraum					
		T1	T2	T3	T4	T5	T6
S	SNNS	27,3	14,7	23,1	12,5	19,9	185,8
	Tegut	36,3	17,6	37,3	35,8	22,2	161,1
	Naiv	55,8	22,1	46,3	50,4	21,8	360,2
SP	SNNS	29,4	27,1	25,5	27,7	19,5	32,1
	Tegut	35,1	38,0	30,4	41,9	28,8	38,7
	Naiv	54,8	39,4	45,7	47,7	31,7	58,1
SAP	SNNS	30,2	24,8	25,4	12,9	33,1	37,1
	Tegut	37,2	28,4	30,9	27,4	33,0	43,6
	Naiv	45,3	30,9	37,2	35,3	49,0	50,3
LP	SNNS	52,0	63,6	57,2	32,9	40,2	22,8
	Tegut	63,1	67,6	62,7	51,9	38,9	28,8
	Naiv	65,3	95,9	81,6	71,0	62,9	38,5
LAP	SNNS	48,1	36,8	46,4	30,8	41,0	58,7
	Tegut	39,8	37,1	51,9	49,0	37,6	53,0
	Naiv	54,9	48,7	66,7	44,5	49,8	74,2

Da es unter den getesteten Topologien keine gibt, die in jedem einzelnen der sechs Zeiträume bessere Ergebnisse liefert als die Tegut-Prognose, werden nun die Topologien bevorzugt eingesetzt, die im Mittel über die Zeiträume bessere Ergebnisse liefern. Die Tabellen 13.8 bis 13.12 zeigen die mittleren Prognosefehler über alle Zeiträume und verschiedenen Topologien für jede

Artikelgruppe sortiert nach den Werten des MAPE-Fehlers. Jeweils fettgedruckt erscheint die ausgewählte Topologie einer Gruppe. Wie viele Topologien besser sind als die Tegut-Prognose, gibt die doppelte horizontale Linie in den Tabellen an. Unterhalb dieser Linie angegebene Topologien haben einen schlechteren Prognosefehler als die Tegut-Prognose. Bei den Gruppen LP und SP wird zugunsten einer kleineren Netztopologie auf die jeweils beste Topologie verzichtet. Bei den Gruppen S und LAP sind alle Topologien von ihrem Prognoseverhalten schlechter als die Tegut-Prognose, während bei der Gruppe SAP fast alle Topologien besser prognostizieren. Das Ergebnis der Gruppe S muß deshalb mit Vorsicht behandelt werden, als diese Gruppe nur aus einem Artikel besteht, und somit nicht repräsentativ ist. Die Fehlerwerte in den einzelnen Zeiträumen der gewählten Topologien geben die Tabellen 13.6 und 13.7 wider. Die ausgewählten Topologien werden durch Verwendung der Kreuzvalidierung in ihrer Güte gegenüber den übrigen Topologien bestätigt. Der Umstand, daß in den Gruppen SP, SAP, LP und LAP jeweils vier Abverkaufsneuronen vorkommen und dabei eine gute Prognosequalität erreicht wird (immer im Vergleich zur Tegut-Prognose), kann mit Hilfe des Pruningverfahrens *Skeletonization* und eingestelltem Pruning von Inputneuronen nachgewiesen werden. Dazu wird die Anzahl der Abverkaufsneuronen im Input auf einen hohen Wert gesetzt (hier 10 ABV) und das Training inklusive anschließendem Pruning gestartet. In der überwiegenden Zahl der Testläufe reduziert sich die Zahl der Abverkaufsneuronen auf vier.
Bezüglich der Lernverfahren läßt sich feststellen, daß sich das Einbinden der neuen Zielfunktion in das Lernverfahren *BPM_ln_cosh* als positiv erwiesen hat. Dieses Lernverfahren schneidet gegenüber dem *Rprop*-Verfahren deutlich besser ab. Dies kann an den Tabellen 13.8 bis 13.12 abgelesen werden, da unter den besten Topologien überwiegend solche mit diesem Lernverfahren zu finden sind.

Tabelle 13.6: Die besten Netze im Mittel über die Zeiträume T1 – T6 im Vergleich zur Tegut-Prognose (MAPE)

Cluster Topologie	Prognose	Zeitraum T1	T2	T3	T4	T5	T6	Mittel
S	SNNS	31,8	24,9	37,0	21,7	31,5	185,8	55,4
A3F3J1sL	Tegut	36,3	17,6	37,3	35,8	22,2	161,1	51,7
SP	SNNS	32,9	44,7	32,4	29,4	20,7	46,1	34,4
A4P1F1J1sL	Tegut	35,1	38,0	30,4	41,9	28,8	38,7	35,4
SAP	SNNS	34,3	25,9	28,6	19,5	35,7	38,8	30,5
A4K1P1F1J1sL	Tegut	37,2	28,4	30,9	27,4	33,0	43,6	33,4
LP	SNNS	56,0	71,9	67,8	39,2	43,3	27,2	50,9
A4P1F1J1sL	Tegut	63,1	67,6	62,7	51,9	38,9	28,8	52,1
LAP	SNNS	50,0	40,1	66,2	35,8	43,5	63,5	49,9
A4K1P1F1J1sL	Tegut	39,8	37,1	51,9	49,0	37,6	53,0	44,7

Tabelle 13.7: Die besten Netze im Mittel über die Zeiträume T1 – T6 im Vergleich zur Tegut-Prognose (RMSE)

Cluster Topologie	Prognose	Zeitraum						
		T1	T2	T3	T4	T5	T6	Mittel
S A3F3J1sL	SNNS	9,46	8,47	8,66	2,58	5,63	12,67	7,91
	Tegut	8,42	6,33	6,44	5,32	6,38	14,77	7,94
SP A4P1F1J1sL	SNNS	36,01	17,51	14,50	11,80	8,44	22,47	18,46
	Tegut	39,86	14,86	12,25	13,42	10,87	22,39	18,97
SAP A4K1P1F1J1sL	SNNS	5,98	4,80	4,73	3,70	6,82	5,99	5,34
	Tegut	6,53	5,41	5,50	5,31	7,28	7,38	6,23
LP A4P1F1J1sL	SNNS	3,32	3,42	3,56	3,45	3,28	2,48	3,25
	Tegut	3,55	3,15	3,17	3,56	2,43	2,47	3,05
LAP A4K1P1F1J1sL	SNNS	3,27	3,31	3,48	2,46	2,98	3,71	3,20
	Tegut	3,10	3,52	3,10	2,97	2,63	3,32	3,10

Tabelle 13.8: Über die Zeiträume T1 – T6 gemittelte Fehler getesteter Topologien der Gruppe S

Topologie	MAPE	RMSE
A3F3J1sL	**55.49**	**7.916**
A5F1J1sLX	55.74	7.295
A5F2J1sLX	55.86	7.661
A3F1J1sLX	56.10	7.521
A3F1J1sL	56.20	7.721
A4F2J1sLX	57.16	7.516
A5F3J1sLX	57.39	7.276
A3F2J1sL	57.40	7.574
A3F3J1sRX	58.39	7.539
A3F3J1sLX	58.74	7.653
A4F3J1sLX	58.91	7.654
A3F3J1sR	58.95	7.786
A3F2J1sLX	59.21	7.979
A3F2J1sRX	59.37	7.360
A4F1J1sLX	60.07	7.561
A4F3J1sRX	60.83	7.666
A3F1J1sR	61.07	7.419
A4F3J1sL	61.60	7.652

Topologie	MAPE	RMSE
A3F2J1sR	62.13	7.802
A3F1J1sRX	63.46	7.905
A4F1J1sL	63.77	7.769
A4F1J1sRX	64.43	7.691
A4F2J1sL	65.13	7.986
A4F2J1sR	65.21	7.861
A4F2J1sRX	65.51	7.708
A4F1J1sR	65.83	7.945
A4F3J1sR	66.05	7.828
A5F2J1sRX	67.73	8.082
A5F1J1sL	68.34	8.030
A5F3J1sL	68.36	7.704
A5F2J1sL	69.47	7.614
A5F3J1sRX	69.56	8.035
A5F1J1sRX	73.86	8.149
A5F3J1sR	74.31	8.309
A5F1J1sR	75.63	8.354
A5F2J1sR	77.55	8.385

Tabelle 13.9: Über die Zeiträume T1 – T6 gemittelte Fehler getesteter Topologien der Gruppe SP

Topologie	MAPE	RMSE
A5P2F1J1sLX	33.68	18.64
A5P1F1J1sLX	33.96	17.72
A4P1F1J1sL	**34.40**	**18.46**
A3P1F3J1sLX	34.42	17.60
A3P3F1J1sLX	34.44	17.71
A4P1F2J1sLX	34.50	18.05
A3P2F1J1sLX	35.47	17.90
A3P1F1J1sLX	35.73	17.59
A5P2F2J1sLX	36.08	18.65
A4P2F2J1sLX	36.31	18.74
A4P2F1J1sLX	36.50	19.47
A3P1F1J1sL	36.57	18.65
A3P1F2J1sLX	36.65	18.03
A4P1F2J1sL	36.76	18.79
A5P1F2J1sLX	36.89	19.27
A5P1F2J1sL	36.98	18.85
A3P1F3J1sL	37.02	18.35
A4P1F1J1sLX	37.02	18.86
A3P1F2J1sL	37.15	18.19
A4P2F1J1sL	37.82	19.38
A5P1F1J1sL	38.19	19.47
A3P2F1J1sL	38.66	19.12
A3P3F1J1sL	39.32	19.48
A5P1F2J1sRX	39.74	19.32
A4P2F2J1sL	40.52	19.79
A5P2F1J1sL	40.64	19.58

Topologie	MAPE	RMSE
A5P2F1J1sRX	40.83	20.01
A5P2F2J1sL	41.01	19.77
A3P1F1J1sRX	41.63	19.49
A3P3F1J1sRX	41.68	19.73
A4P1F2J1sRX	41.86	19.91
A4P1F1J1sRX	42.22	19.88
A3P1F3J1sRX	42.55	19.75
A5P1F1J1sRX	43.13	20.37
A3P1F2J1sRX	43.64	19.80
A3P1F1J1sR	43.68	19.91
A4P2F2J1sRX	44.40	19.99
A3P1F2J1sR	44.47	20.48
A5P1F2J1sR	44.49	21.25
A5P2F2J1sRX	44.50	20.61
A4P2F1J1sRX	45.54	21.23
A3P2F1J1sRX	45.56	20.34
A4P1F1J1sR	45.81	22.21
A3P2F1J1sR	46.36	21.71
A3P1F3J1sR	46.56	20.42
A4P2F2J1sR	46.61	21.68
A3P3F1J1sR	46.74	21.43
A5P1F1J1sR	47.12	21.75
A4P1F2J1sR	47.29	22.08
A5P2F1J1sR	48.93	22.83
A5P2F2J1sR	50.84	23.39
A4P2F1J1sR	51.03	23.65

Tabelle 13.10: Über die Zeiträume T1 – T6 gemittelte Fehler getesteter Topologien der Gruppe SAP

Topologie	MAPE	RMSE
A4K1P1F1J1sL	**30.51**	**5.343**
A4K1P1F1J1sRX	30.74	5.497
A5K1P1F1J1sLX	30.86	5.302
A4K1P2F1J1sL	30.87	5.414
A5K1P2F2J1sLX	30.97	5.331
A4K1P1F1J1sR	31.10	5.436
A4K2P2F2J1sL	31.14	5.487
A5K2P1F1J1sL	31.15	5.495
A4K2P1F1J1sL	31.18	5.519
A3K1P1F1J1sL	31.24	5.396
A4K1P2F1J1sRX	31.29	5.319
A4K1P1F1J1sLX	31.39	5.376
A5K1P1F1J1sL	31.41	5.533
A4K2P2F1J1sLX	31.43	5.313
A4K1P2F2J1sL	31.48	5.443
A4K2P2F1J1sL	31.52	5.513
A3K2P1F1J1sLX	31.55	5.318
A3K1P1F1J1sLX	31.56	5.330
A5K2P1F1J1sRX	31.57	5.478
A4K1P1F2J1sL	31.58	5.498
A4K1P2F1J1sLX	31.58	5.362
A5K2P2F1J1sL	31.58	5.517
A3K1P2F1J1sL	31.64	5.521
A5K2P2F1J1sRX	31.66	5.522
A3K1P1F2J1sL	31.71	5.489
A3K1P2F2J1sL	31.71	5.550
A4K1P2F2J1sLX	31.71	5.372
A4K1P1F2J1sLX	31.72	5.364
A4K2P1F1J1sRX	31.76	5.481
A5K1P2F1J1sRX	31.78	5.629
A4K1P1F2J1sR	31.79	5.644
A3K2P1F1J1sL	31.80	5.603
A5K1P2F2J1sRX	31.80	5.535
A5K1P2F2J1sL	31.82	5.575
A5K1P2F1J1sL	31.84	5.517
A4K2P2F2J1sLX	31.85	5.566
A5K2P2F1J1sLX	31.96	5.314
A3K2P2F1J1sL	31.97	5.649
A3K1P2F1J1sLX	31.99	5.432
A4K1P2F2J1sRX	31.99	5.474
A4K1P1F2J1sRX	32.01	5.523
A3K1P2F1J1sRX	32.02	5.697

Topologie	MAPE	RMSE
A4K2P2F2J1sRX	32.02	5.417
A4K1P2F1J1sR	32.03	5.648
A4K2P2F1J1sRX	32.03	5.526
A5K2P2F2J1sL	32.03	5.629
A5K2P1F1J1sR	32.08	5.645
A3K1P1F2J1sRX	32.13	5.601
A4K2P1F1J1sR	32.14	5.621
A4K2P1F1J1sLX	32.18	5.451
A3K2P2F2J1sL	32.21	5.634
A3K1P1F2J1sLX	32.28	5.404
A3K2P2F2J1sRX	32.30	5.722
A5K2P1F1J1sLX	32.42	5.481
A5K1P1F2J1sL	32.44	5.634
A3K2P2F1J1sRX	32.45	5.552
A3K2P1F1J1sRX	32.47	5.756
A3K2P2F2J1sLX	32.54	5.417
A5K1P1F2J1sLX	32.57	5.568
A5K1P1F2J1sRX	32.58	5.985
A5K2P2F2J1sLX	32.68	5.416
A5K1P1F1J1sRX	32.70	5.627
A3K2P2F1J1sLX	32.72	5.496
A5K1P2F2J1sR	32.75	5.922
A4K2P2F1J1sR	32.77	6.054
A5K1P1F1J1sR	32.83	5.774
A3K1P2F2J1sLX	32.88	5.516
A3K1P1F1J1sRX	32.94	5.616
A5K2P2F2J1sRX	32.95	5.659
A3K1P2F2J1sRX	32.98	5.578
A5K1P1F2J1sR	33.07	6.055
A3K1P1F2J1sR	33.12	6.152
A4K2P2F2J1sR	33.21	6.002
A5K2P2F2J1sR	33.24	6.01
A3K1P1F1J1sR	33.42	6.002
A5K1P2F1J1sLX	33.46	5.577
A3K1P2F1J1sR	33.47	5.855
A5K1P2F1J1sR	33.56	5.978
A5K2P2F1J1sR	33.60	6.095
A3K1P2F2J1sR	33.69	6.000
A3K2P2F1J1sR	33.78	5.982
A3K2P1F1J1sR	33.83	5.971
A4K1P2F2J1sR	33.97	6.017
A3K2P2F2J1sR	35.66	6.340

Tabelle 13.11: Über die Zeiträume T1 – T6 gemittelte Fehler getesteter Topologien der Gruppe LP

Topologie	MAPE	RMSE
A5P2F1J1sL	49.60	3.238
A4P2F1J1sL	50.08	3.285
A5P1F2J1sL	50.16	3.213
A5P1F1J1sL	50.27	3.294
A5P2F2J1sL	50.35	3.222
A5P2F1J1sR	50.74	3.292
A4P1F1J1sL	**50.92**	**3.254**
A4P1F2J1sLX	51.06	3.264
A4P2F1J1sLX	51.13	3.294
A4P1F1J1sRX	51.26	3.174
A4P2F2J1sLX	51.30	3.391
A4P1F1J1sR	51.64	3.364
A4P2F2J1sL	51.80	3.312
A5P1F1J1sLX	51.95	3.235
A5P2F2J1sLX	52.12	3.344
A3P2F1J1sL	52.26	3.285
A4P1F2J1sL	52.28	3.319
A5P2F1J1sLX	52.31	3.317
A5P1F2J1sLX	52.52	3.223
A3P3F1J1sL	52.54	3.307
A3P1F2J1sLX	52.72	3.337
A3P1F3J1sLX	52.83	3.391
A4P2F1J1sRX	52.96	3.292
A3P2F1J1sLX	52.99	3.349
A4P2F2J1sRX	53.12	3.353
A3P1F1J1sLX	53.16	3.349

Topologie	MAPE	RMSE
A3P1F2J1sL	53.24	3.320
A5P1F2J1sR	53.34	3.487
A5P2F2J1sRX	53.34	3.314
A5P1F2J1sRX	53.36	3.280
A4P2F2J1sR	53.47	3.342
A3P1F1J1sL	53.50	3.360
A5P2F1J1sRX	53.71	3.334
A4P1F1J1sLX	53.85	3.373
A4P1F2J1sR	53.88	3.370
A3P3F1J1sLX	53.89	3.392
A5P1F1J1sR	54.55	3.752
A4P2F1J1sR	54.65	3.569
A4P1F2J1sRX	54.68	3.296
A3P1F3J1sL	54.75	3.353
A5P1F1J1sRX	55.25	3.351
A3P1F3J1sRX	55.47	3.274
A3P2F1J1sRX	56.00	3.348
A3P3F1J1sR	56.05	3.556
A3P1F1J1sR	57.02	3.491
A3P3F1J1sRX	57.41	3.276
A3P2F1J1sR	57.67	3.540
A5P2F2J1sR	57.84	3.615
A3P1F2J1sRX	58.14	3.384
A3P1F1J1sRX	58.43	3.555
A3P1F3J1sR	59.34	3.606
A3P1F2J1sR	60.37	3.600

Tabelle 13.12: Über die Zeiträume T1 – T6 gemittelte Fehler getesteter Topologien der Gruppe LAP

Topologie	MAPE	RMSE
A4K1P1F1J1sL	**49.90**	**3.208**
A4K1P2F1J1sL	50.17	3.208
A3K1P1F1J1sL	50.97	3.265
A5K1P2F1J1sL	51.14	3.217
A5K2P2F1J1sL	51.21	3.262
A3K1P2F1J1sL	51.34	3.283
A3K2P1F1J1sL	51.46	3.298
A3K2P2F1J1sL	51.46	3.246
A4K2P1F1J1sL	51.57	3.219
A5K1P1F1J1sL	51.59	3.269
A5K1P2F2J1sL	51.61	3.257
A4K2P2F2J1sL	51.75	3.255
A4K1P1F2J1sL	51.77	3.243
A5K2P2F2J1sL	51.78	3.278
A5K2P1F1J1sL	51.83	3.320
A4K1P2F2J1sL	52.49	3.274
A5K2P1F1J1sRX	52.58	3.250
A4K2P2F1J1sR	52.60	3.397
A4K2P1F1J1sR	52.65	3.346
A4K2P2F1J1sL	52.72	3.251
A5K2P1F1J1sLX	52.78	3.300
A4K2P2F1J1sLX	52.79	3.251
A3K1P1F1J1sRX	53.00	3.293
A4K2P2F2J1sRX	53.01	3.292
A5K1P1F2J1sL	53.04	3.340
A3K1P2F2J1sL	53.06	3.360
A4K1P2F1J1sR	53.15	3.357
A5K2P2F1J1sR	53.15	3.464
A3K1P1F2J1sL	53.24	3.317
A4K2P2F2J1sLX	53.28	3.262
A4K1P1F1J1sRX	53.31	3.279
A4K1P2F2J1sR	53.44	3.441
A3K1P1F1J1sLX	53.45	3.361
A4K1P2F1J1sLX	53.50	3.254
A5K1P1F2J1sR	53.50	3.338
A4K1P1F2J1sLX	53.66	3.271
A4K1P2F2J1sLX	53.72	3.305
A5K2P2F1J1sRX	53.80	3.273
A5K1P2F1J1sLX	53.95	3.328
A5K2P2F1J1sLX	54.13	3.245
A3K2P1F1J1sR	54.20	3.297
A5K1P2F1J1sRX	54.26	3.245

Topologie	MAPE	RMSE
A5K2P2F2J1sRX	54.36	3.397
A5K1P1F1J1sR	54.43	3.458
A4K2P1F1J1sLX	54.46	3.339
A5K1P2F2J1sRX	54.50	3.335
A5K1P2F1J1sR	54.52	3.292
A3K2P1F1J1sLX	54.58	3.289
A4K1P2F1J1sRX	54.63	3.259
A3K2P2F2J1sLX	54.66	3.388
A4K2P1F1J1sRX	54.76	3.314
A4K1P1F2J1sR	54.87	3.354
A5K1P1F2J1sLX	54.90	3.345
A5K1P2F2J1sR	55.02	3.562
A5K2P1F1J1sR	55.07	3.408
A4K2P2F1J1sRX	55.10	3.309
A3K1P1F1J1sR	55.17	3.357
A3K1P2F1J1sLX	55.28	3.328
A3K2P2F1J1sRX	55.34	3.314
A3K2P2F2J1sL	55.34	3.362
A4K1P1F1J1sR	55.41	3.425
A5K1P1F1J1sRX	55.43	3.306
A3K1P2F1J1sRX	55.45	3.348
A4K1P2F2J1sRX	55.51	3.455
A4K2P2F2J1sR	55.74	3.446
A3K2P2F1J1sLX	55.80	3.353
A3K2P2F2J1sR	55.84	3.360
A5K1P1F1J1sLX	56.08	3.393
A5K1P1F2J1sRX	56.15	3.406
A5K1P2F2J1sLX	56.15	3.307
A4K1P1F2J1sRX	56.24	3.332
A3K1P1F2J1sR	56.31	3.326
A5K2P2F2J1sLX	56.31	3.373
A3K1P2F2J1sLX	56.32	3.339
A4K1P1F1J1sLX	56.42	3.383
A5K2P2F2J1sR	56.85	3.497
A3K2P1F1J1sRX	57.13	3.353
A3K1P1F2J1sLX	57.35	3.329
A3K1P1F2J1sRX	57.59	3.420
A3K1P2F2J1sRX	57.70	3.455
A3K2P2F1J1sR	58.17	3.501
A3K1P2F1J1sR	58.79	3.521
A3K2P2F2J1sRX	59.01	3.463
A3K1P2F2J1sR	59.06	3.455

Kapitel 14

Zusammenfassung

Hier werden die Ergebnisse diskutiert, die sich aus dem zusätzlichen Funktionsumfang des Prognosesystems ergeben haben. Im speziellen werden sie hinsichtlich der Prognosegüte innerhalb des Tegut-Projektes, im allgemeinen hinsichtlich der Anwendbarkeit des Prognosesystems diskutiert.
Es muß nochmals betont werden, daß das Ziel bei der Prognose der Tegut-Zeitreihen mit Neuronalen Netzen darin bestand, eine bessere Prognosegüte zu erreichen, als sie die Tegut-Prognose zu erreichen vermag. Das heißt, die angegebenen Fehlerwerte sollten nicht isoliert von Vergleichswerten als absolutes Kriterium betrachtet werden. Unter diesem Aspekt ergibt sich zusammenfassend folgendes:
Die Prognose mittels Neuronaler Netze liefert überwiegend bessere Ergebnisse als die Tegut-Prognose. Nur für die Produktgruppe LAP sollte weiterhin die Tegut-Prognose angewendet werden, da in dieser Gruppe die erreichten Prognosegüten durchweg schlechter sind als die der Tegut-Prognose. Die Produktgruppe S ist insofern ein Sonderfall, als sie nur einen einzigen Artikel enthält. Damit sind die mit dieser Gruppe verbundenen Ergebnisse wenig aussagekräftig. Die übrigen drei Produktgruppen SP, SAP und LP zeigen ein sehr positives Verhalten, insbesondere die Gruppe SAP, die in allen betrachteten Zeiträumen gute Ergebnisse liefert.
Es wird zudem deutlich, daß die Prognosegüte stark vom gewählten Prognosezeitraum abhängt, wie die großen Schwankungen der Fehlerwerte über die Zeiträume zeigen.
Seitens der Parametereinstellungen hat sich folgendes ergeben:
Als vorteilhaft erweist sich — auch in anderen Projekten — die Verwendung der `MY_SIGMA`-Skalierung, die unter anderem eine Ausreißerglättung vornimmt, so daß das Neuronale Netz leichter generalisieren kann. Die modifizierte, speziell für ökonometrische Zeitreihen konzipierte Fehlerfunktion, kann ebenfalls die Güte verbessern. Der Einsatz von Shortcuts kann die linearen Anteile des Abverkaufmodells erkennen. Die Verbesserung der Prognosegüte bestätigt diese Vermutung. Ähnlich verhält es sich mit der zusätzlichen Verwendung einer Cosinus-Zeitreihe im Input, die die Saisonalität des Modells unterstützt und damit bessere Ergebnisse erzielt.
Indifferent dagegen verhält sich die Verwendung einer Vorschicht. Es können keine konkreten Aussagen innerhalb des Tegut-Projektes getroffen werden. Es gibt Topologien, in der sie sich als vorteilhaft erweist, aber auch solche, bei denen sie von Nachteil ist. Mit Hilfe weiterer Funktio-

nalität (Kreuzvalidierung, Pruning und Sensitivitätsanalyse) konnten die Ergebnisse bezüglich der eingesetzten Topologien weitestgehend validiert werden. Die positive Wirkung von Kohonen-Netzen zur Filterung der Eingabewerte kann im Tegut-Projekt nicht nachgewiesen werden. Die Ergebnisse der diesbezüglichen Testläufe zeigten keine Verbesserung in der Prognosequalität (aber auch keine Verschlechterung). Obwohl die einzelnen Zeitreihen innerhalb des Tegut-Projekt im Gegensatz zu den anderen Projekten sehr wenig Datenmaterial aufweisen, können durch den Einsatz Neuronaler Netze dennoch akzeptable Ergebnisse erzielt werden.
Da es sich aufgrund der Implementation des Prognosesystems um ein sehr offenes System handelt, ist es nach einer Einarbeitungsphase zu verstehen und leicht erweiterbar. Bibliotheken für grundlegende Aufgaben wie für Zeitrechnung, Statistik und Zeitreihenverwaltung sind vorhanden und können in eigenen Systemerweiterungen genutzt werden. Von diesem Umstand wurde im Rahmen der Einbettung von *SNNS* Gebrauch gemacht.
Die Erweiterungen, die Funktionen aus *SNNS* betreffen, erfolgten weitestgehend nach folgendem Muster: Zuerst wurde ein Algorithmus für ein Batchman-Skript entworfen und dieser dann als Batchman-Template abgelegt. Perl-Skripte, die der Nachverarbeitung dienen, wurden hinzugefügt. Je nach Komplexität handelte es sich um ein (z.B. bei Kreuzvalidierung) oder mehrere (z.B. bei Sensitivitätsanalyse, Kohonen-Netze) Skripte. Andere Erweiterungen, die als eigenständige Programme konzipiert wurden (z.B. `zr_error.pl`), verlangten einen großen Programmieraufwand. Gerade dieses Tool war es, das bei der Auswertung der vielen Prognoseläufe wertvolle Hilfe geleistet hat. Besonderes Augenmerk verdient dabei die Nutzung der Track-Tabellen, deren Inhalt eine hohe Aggregationsstufe der Fehlerwerte darstellt. Die Tabellen wurden überwiegend mit individuell gestalteten Shell-Skripten weiterverarbeitet und in eine endgültige Form gebracht. Zwar wurde von reinen Nutzern des Prognosesystems manchmal weitere Funktionalität innerhalb des Tools `zr_error.pl` gewünscht, aber die meisten Belange wurden abgedeckt.
Der Stuttgarter Neuronale Netze Simulator hat sich im Prognosesystem bewährt. Durch den Einsatz zusätzlicher Lernfunktionen, Pruning-Algorithmen und Kohonen-Netze wird ein breites Spektrum abgedeckt, in dem Prognoseläufe stattfinden, so daß die empirischen Ergebnisse gut abgesichert sind. Mit Hilfe von Batchman- und Perl-Skripten wurden auch nicht direkt in *SNNS* zur Verfügung stehende Möglichkeiten entwickelt, wie die Kreuzvalidierung und die Sensitivitätsanalyse. Des weiteren ist *SNNS* als offenes System leicht durch eigene Funktionen erweiterbar und dadurch sehr flexibel, wie das Einbinden der neuen Lernfunktion `BPM_ln_cosh` beweist.

Literaturverzeichnis

[Bra97] BRAUN, HEINRICH: *Neuronale Netze.* Springer Verlag, 1997.

[Con96] CONRAD, OLAF: *Die Einsatzmöglichkeiten von neuronalen Netzen im Rahmen der Absatzprognose und Bezugsregulierung im Vertrieb von Publikumszeitschriften*, Band 1841 der Reihe *Europäische Hochschulschriften, Reihe V, Volks- und Betriebswirtschaft.* Peter Lang, Europäischer Verlag der Wissenschaften, Frankfurt am Main, 1996. Dissertation Universität Hamburg, Juli 1995.

[Efr83] EFRON, BRADLEY: *Estimating the Error Rate of a Prediction Rule: Improvement on Cross-Validation.* Journal of the American Statistical Association, 78(382):316–331, Juni 1983.

[GMS94] GOOSSENS, MICHEL, MITTELBACH, FRANK und SAMARIN, ALEXANDER: *Der LaTeX-Begleiter.* Addison-Wesley, 1994.

[Gün96] GÜNTHER, KARSTEN: *Einführung in LaTeX2e.* dpunkt, 1996.

[Han83] HANSMANN, KARL-WERNER: *Kurzlehrbuch Prognoseverfahren.* Verlag Gabler, 1983.

[Jaf96] JAFAR-SHAGHAGHI, FARIBA: *Maschinelles Lernen, Neuronale Netze und Statistische Lernverfahren zur Klassifikation und Prognose.* Shaker Verlag, 1996. Dissertation Fakultät für Wirtschaftswissenschaften der Universität Fridericiana zu Karlsruhe, Oktober 1995.

[Kop94] KOPKA, HELMUT: *LaTeX Band 1: Einführung.* Addison-Wesley, 1994.

[Kop95] KOPKA, HELMUT: *LaTeX Band 2: Ergänzungen.* Addison-Wesley, 1995.

[Kro97] KRONE, STEPHAN: *Eine interaktive Oberfläche zur grafischen Analyse von Zeitreihen.* Diplomarbeit, Universität Osnabrück, Juli 1997.

[Mid95] MIDDELBERG, ULRICH: *Parallele Backpropagation.* Diplomarbeit, Universität Osnabrück, September 1995.

[Roj93] ROJAS, R.: *Theorie der neuronalen Netze.* Springer Verlag, 1993.

[RZ94] REHKUGLER, H. und ZIMMERMANN, H.G. (Herausgeber): *Neuronale Netze in der Ökonomie.* Verlag Vahlen, 1994.

[Sch97] SCHILLI, MICHAEL: *Effektives Programmieren mit Perl 5.* Addison-Wesley, 1997.

[SP97] SCHWARZ, STEFAN und POTUCEK, RUDOLF: *Das TEXikon.* Addison-Wesley, 1997.

[Thi98] THIESING, FRANK M.: *Analyse und Prognose von Zeitreihen mit Neuronalen Netzen.* Dissertation, Universität Osnabrück, Mai 1998.

[TV97] THIESING, FRANK M. und VORNBERGER, OLIVER: *Abverkaufsprognose im Supermarkt mit Neuronalen Netzen.* In: BIETHAHN, J., KUHL, J., LEISEWITZ, M.C., NISSEN, V. und TIETZE, M. (Herausgeber): *Softcomputing-Anwendungen im Dienstleistungsbereich*, Seiten 137–147. Arbeitskreis "Softcomputing in der Betriebswirtschaft" in der AFN (Arbeitsgemeinschaft Fuzzy Logik und Softcomputing Norddeutschland, Februar 1997. Tagungsband zum 3. Göttinger Symposium Softcomputing am 27. Feb. 1997 an der Universität Göttingen.

[TV98] THIESING, FRANK M. und VORNBERGER, OLIVER: *Abverkaufsprognose im Supermarkt mit Neuronalen Netzen.* In: BIETHAHN, J., HÖNERLOH, A., KUHL, J., LEISEWITZ, M.-C., NISSEN, V. und TIETZE, M. (Herausgeber): *Betriebswirtschaftliche Anwendungen des Soft Computing*, Computational Intelligence, Seiten 1–13. Vieweg, 1998.

[WS91] WALL, LARRY und SCHWARTZ, RANDAL L.: *Programming perl.* O'Reilly & Associates, Inc., 1991.

[Zel94] ZELL, ANDREAS: *Simulation Neuronaler Netze.* Addison-Wesley, 1994.

[Zel95] ZELL, ANDREAS: *SNNS Stuttgart Neural Network Simulator User Manual, Version 4.1.* University of Stuttgart, Germany, 1995. Report No. 6/95.