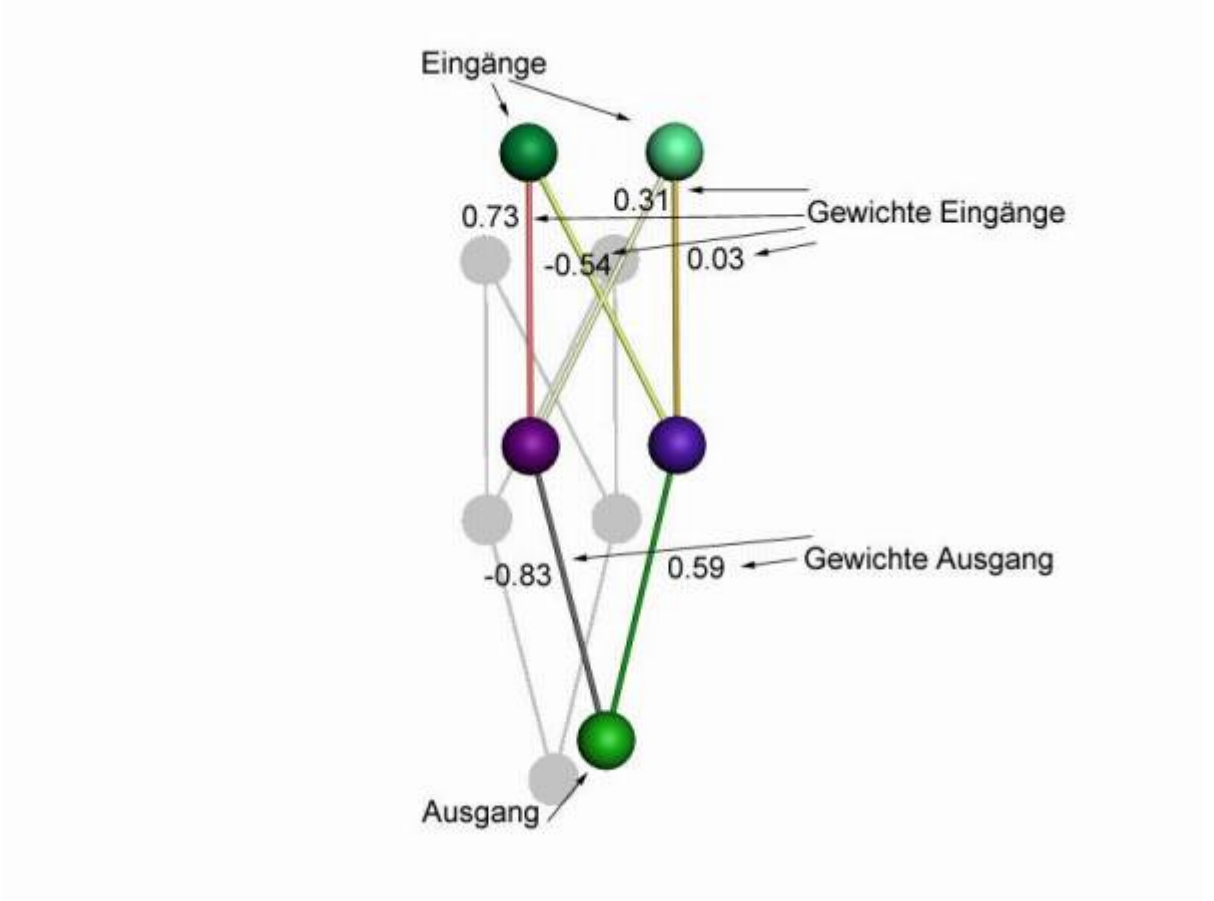
KURSPROGNOSE MIT NEURONALEN NETZEN

PROSEMINARARBEIT
Sommersemester 2002

Markus Glaser
Ralph Gollner



1 Geschichtlicher Abriss

Die Anfänge der Theorie der neuronalen Netze gehen zurück auf das Jahr 1943. Damals veröffentlichten die Mathematiker W.S. McCulloch und W. Pitts eine Arbeit, in der erstmals der Begriff des Neurons als ein logisches Schwellenwertelement mit *n* Eingängen und einem Ausgang beschrieben wurde. Das Neuron konnte zwei Zustände annehmen und feuerte, wenn die kumulierten Eingänge einen Schwellenwert überstiegen. Mc Culloch und Pitts zeigten, dass jede aussagenlogische Funktion durch eine geeignete Kombination dieser Neuronen beschreibbar ist.

Die McCulloch–Pitts-Neuronen waren in ein Netz mit unveränderlichen Gewichten eingebunden, das Netz war nicht lernfähig.

1949 erschien das Buch „The Organization Of Behavior“ des Psychologen D. Hebb, in dem die Lernfähigkeit des Gehirns auf die Anpassungsfähigkeit der Synapsen zurückgeführt wurde. Die mittlerweile klassische Hebb’sche Lernregel als einfaches universelles Lernkonzept individueller Neuronen ist heute Basis fast aller neuronaler Lernverfahren.

1958 stellte F. Rosenblatt das Perzeptron vor. Dieses Netzwerk arbeitete mit einer modifizierten Lernregel (Delta-Regel) und war lernfähig. Es konnte gezeigt werden, dass das Lernverfahren in endlich vielen Schritten konvergiert, dass das Netz fehlertolerant war und dass es für leicht veränderte Eingaben richtige Ausgaben lieferte. Das Perzeptron war weiters Vorbild für verwandte Netztypen wie z.B. das Adaline von B. Widrow und M. Hoff im Jahre 1960. Es wurden auch schon erste bescheidene Anwendungen entwickelt.

Die Zeit von 1955 bis 1969 wird allgemein als die erste Blütezeit der Neuronale-Netze-Forschung angesehen. Man glaubte damals, man hätte die grundlegenden Prinzipien selbstlernender „intelligenter“ Systeme bereits entdeckt. Folgend erkannte man aber die Grenzen der damals verwendeten Modelle und Lernverfahren und dies führte anschließend zu einem jähen Einbruch an Popularität, der einige stille Jahre folgen sollten.

1969 konnten M. Minsky und S. Papert in ihrem Buch „Perceptrons“ beweisen, dass das Perzeptron und die verwandten Netztypen viele logische Funktionen nicht darstellen können.

Interessant zu erwähnen wäre aber auch, dass während der 70er Jahre Paul Werbos in seiner Dissertation bereits das Backpropagation-Verfahren entwickelte, das allerdings erst ca. 10 Jahre später durch die Arbeiten von Rumelhart und McClelland seine große Bedeutung erlangte.

Nach Überwindung dieser „Krise“ fanden aber zu Beginn der achtziger Jahre die neuronalen Netze wieder ein regeres Interesse. Neue Ansätze waren die Ursache dafür, so zum Beispiel die Arbeiten von G. Hinton, D. Rumelhart und R. Williams über das Verfahren der Fehlerrückführung (Backpropagation).

John Hopfield, ein bekannter Physiker, führte 1982 in einem Vortrag ein Netz mit Rückkoppelung vor, welches heute ein wichtiges Grundmodell für die neuronale Mustererkennung darstellt.

Hinton, Rumelhart und Williams „schufen“ mit ihrem Verfahren der Backpropagation 1986 ein Lernverfahren für mehrschichtige Netze. Da Netze mit mindestens zwei Stufen nicht den bereits erwähnten Einschränkungen des Perzeptrons unterliegen, waren die 1969 von Minsky und Papert veröffentlichten Vorbehalte gegen neuronale Netze damit hinfällig.

Seit ca. 1986 hat sich das Gebiet geradezu explosiv entwickelt: Die Zahl der Forscher auf diesem Gebiet beträgt mehrere Tausend und es gibt eine Vielzahl von wissenschaftlichen Zeitschriften, die als Hauptthema neuronale Netze haben, sowie große anerkannte wissenschaftliche Gesellschaften.

2 Allgemeines

Künstliche neuronale Netze sind Computerprogramme, die die Funktionsweise von menschlichen Nervennetzen nachahmen. Sie erlauben einem Computer, von einem Set an Inputs ein Muster oder eine Generalisierung zu lernen und seine eigenen Schlüsse aus den Daten zu ziehen.
Ein künstliches neuronales Netz kann als eine Reihe von Schichten bestehend aus einzelnen Netzwerkknoten (Neuronen) vorgestellt werden. Der Knoten hat die Aufgabe, Inputs zu sammeln, sie zu transformieren und einen Output zu generieren.

3 Backpropagation-Netzwerke

Dieser Netztyp hat große Bedeutung gewonnen und ist der am meisten verwendete.
Der Netztyp 'Multilayer Perceptron (Backpropagation)' besitzt eine Eingabe- und eine

Ausgabeschicht und kann eine oder mehrere verborgene Schichten besitzen. Die Richtung läuft immer von unten nach oben. Es können beliebig viele Zwischenschichten übersprungen werden. Diese gesamte Anordnung garantiert den „feed-forward“-Charakter des Netzwerks.

Jeder Knoten (**Neuron**) einer Schicht ist mit einem Knoten der vorhergehenden Schicht mittels eines **Gewicht**es verbunden, das lediglich eine Zahl ist, die die Stärke der Verbindung zwischen den beiden Knoten widerspiegelt.

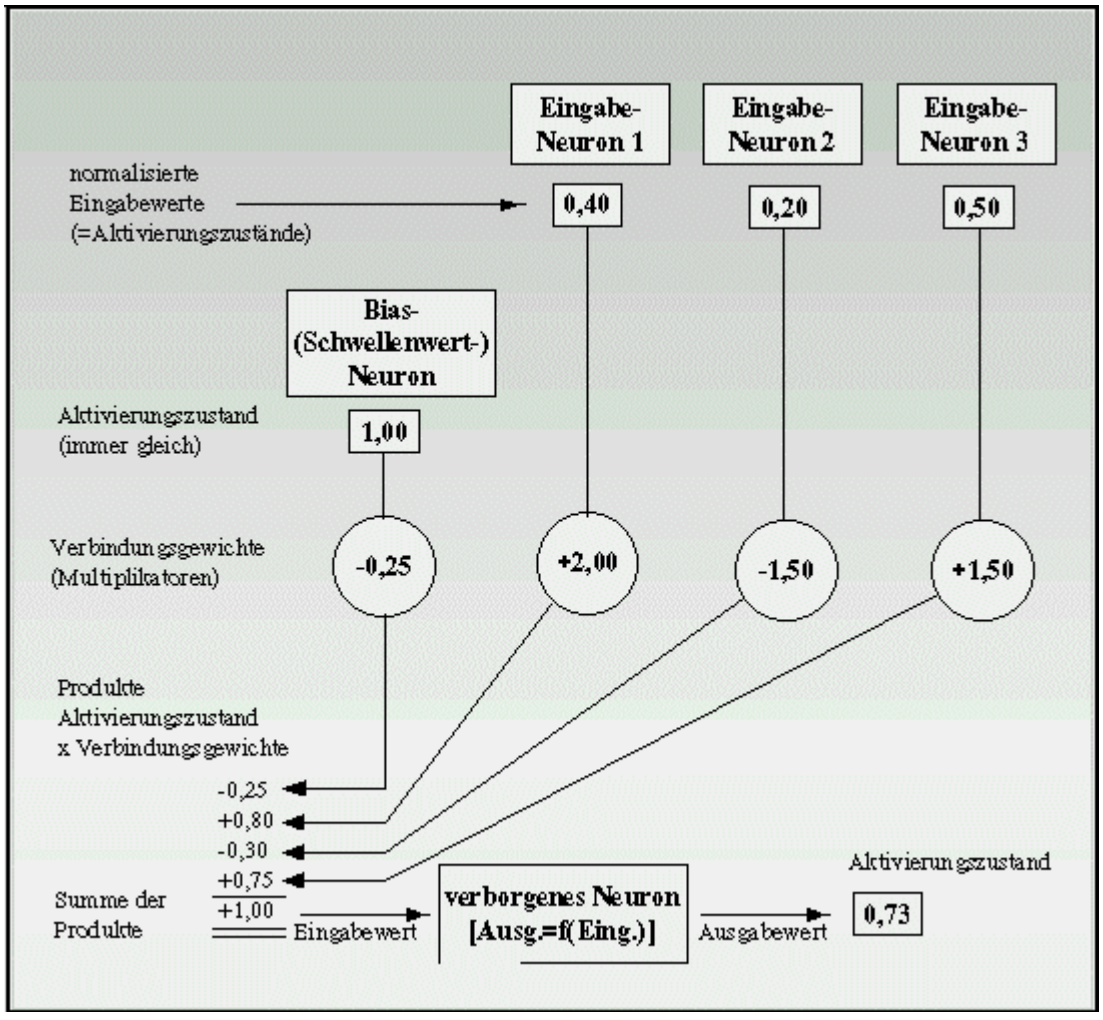


Abbildung 1: Aufbau eines Backpropagation-Netzwerks

Die Hauptgründe dafür, warum diese Art von Netz so verbreitet ist sind folgende:

- Mit dem Lernverfahren 'Backpropagation' steht ein mathematisch bewiesenes **Gradientenverfahren** zur Suche des Fehlerminimums zur Verfügung, das für ein Netz mit beliebig vielen Schichten eingesetzt werden kann.
- Es ist nachweisbar, dass jede logische Funktion durch dieses Netz darstellbar ist.

3.1 Begriffe

3.1.1 Gewichte

Die Gewichte stellen die Verbindungen zwischen den Neuronen dar. Sie bestimmen, wie stark und mit welchen Vorzeichen die Ausgangsaktivität eines Neurons auf den Eingang eines anderen Neurons wirkt. Weiters kann man noch erwähnen, dass sie dem biologischen Vorbild der Synapsen „entsprechen“.

3.1.2 Neuron

Das Neuron im Netzwerk wird durch seine Übertragungsfunktion beschrieben. Der sogenannte Schwellenwert wird durch die gewichtete Verbindung zu einem „statischen Neuron“ simuliert. Über die Gewichte (= Zusammenfassung von „biologischen“ Axon, Dendriten und Synapsen) werden die Signale von Neuronen gewichtet zu anderen Neuronen übertragen. Diese bilden jeweils die Summe der einkommenden Signale, und leiten diese über ihre **Übertragungsfunktion** an den Ausgang. Das Ausgangssignal wird nun weiter über die Gewichte auf die nachgeschalteten Neuronen übertragen.

Jedes Neuron summiert die Werte der mit ihm verbundenen Gewichte und transformiert diese Summe, um einen Outputwert für dieses Neuron zu generieren. Es gibt verschiedene Transformationsschemata, die alle darauf abzielen, die Funktionsweise eines Nerven zu imitieren.

Als günstige **Übertragungsfunktion** zwischen den Neuronen hat sich die sie sigmoide Funktion

f(x) = 1 / (1 + exp(-c * x))

erwiesen, da beim Einsetzen dieser Funktion in den Formeln des Backpropagation–Algorithmus der exponentielle Teil verschwindet und somit eine schnellere Berechnung bei der Simulation möglich ist.

Zu Beginn des Backpropagierens müssen die **Trainingsdaten** (= die Eingangsdaten) eingegeben werden, die zum Trainieren des Netzes benötigt werden.

Die **Trainingsdaten** bestehen aus mehreren Trainingsdatensätzen, deren Werte vor einem Arbeitsschritt auf die Eingangsneuronen des Netzes übertragen werden.

Die **Soll-Ausgabedaten** bestehen aus mehreren Soll-Ausgabedatensätzen. Jeder Soll-Ausgabedatensatz beinhaltet Werte, auf die die Ausgangsaktivitäten der Neuronen der Ausgabeschicht während der Lernphase hin trainiert werden sowie der entsprechenden Klasse, der er zugeordnet wird.

Bei Backpropagation-Netzen werden die Outputwerte mit gewünschten Werten (Targets) verglichen. Der Unterschied zwischen „Zielwert“ und tatsächlichem Output wird Fehler genannt. Dieser Fehler wird weiterverwendet, um die Gewichte derart zu verändern, dass der minimale Fehler gefunden wird. Diesen Prozess des „Back-feedings“ nennt man „Backpropagation“, daher der Name für diesen Netzwerktyp. Der Prozess der Gewichtsanpassung wird „Lernen“ genannt.

Die Inputschicht gibt die Werte einfach an die verdeckten Schichten weiter, die alle Gewichte summieren und dann einen Output produzieren. Der Output einer verdeckten Schicht wird dann weiterverwendet als Input für die nächste Schicht bis hin zur Outputschicht. Der endgültige Output wird nun also mit einem Zielwert verglichen und der Fehler berechnet. Dieser Fehler wird rückwärts durch das Netzwerk gereicht, um die Gewichte anzupassen und den Fehler in der Folge zu minimieren. Dieser Prozess ist eine einzige **Iteration**.

Das komplette Training eines neuronalen Netzwerks kann zehn- bis hunderttausend Iterationen benötigen, abhängig von der Größe des Netzes und der Datenmenge. Die bisher größte Anzahl an Iterationen, die je durchgeführt wurde, beträgt etwa zwei Millionen.

3.1.3 Gewichtsanpassung

Zur Anpassung der Gewichte gibt es prinzipiell zwei verschiedene Möglichkeiten:

- Die Gewichte werden nach jedem Lernschritt sofort angepasst. Dies bedeutet, dass die Minimierung der Fehlerfunktion in Richtung des steilsten Gradienten bezüglich der Quadrate der Fehler eines Musters erfolgt.

- Die andere Möglichkeit ist die kumulative Veränderung der Gewichte. Hierbei werden die Gewichte erst nach einer bestimmten Zahl von Lernschritten angepasst. Das hat den Vorteil, dass tatsächlich der Gesamtfehler der zu lernenden Muster berücksichtigt wird. Der Fehler wird über alle Muster gemeinsam berechnet und fließt dementsprechend in die Änderung der Gewichte ein. Die Reihenfolge der Gewichte spielt dann keine Rolle mehr. Bei einer großen Anzahl von Mustern spielt allerdings die kumulative Anpassung keine Rolle mehr, da zu viele Rechenoperationen nötig wären.

3.1.4 Lernen

Lernen bedeutet, wie bereits oben angeführt, das Adaptieren der Gewichte des Netzes so, dass das Netz bei der Anregung durch die entsprechenden Eingabedaten die gewünschten Soll-Ausgabedaten immer besser reproduziert und entsrechenden Klassen häufiger erkennt.
Die Datensätze der Trainingsdaten und der entsprechenden Soll-Ausgabedaten werden dem Netz nacheinander präsentiert. Dies kann auch in zufälliger Reihenfolge geschehen. Die Bearbeitung eines Datensatzes wurd als Lernschritt bezeichnet.

Weiters ist auch im Rahmen des Netzwerktrainings noch zu sagen, dass man dies noch unterteilen kann:

a) nach Lernkonzept (wiederholte Präsentation der Inputs <> einmalige Präsentation)
b) Lernmethode (unsupervised > Kohonen-Netze, supervised > back-propagation)
c) Und auch nach verschiedenen Lernregeln (Hebb-, Delta-, Konkurrenz-Regel).

3.1.5 Testdaten

Dies sind Daten, die nicht zum Trainieren des Netzes verwendet werden. Die Testdaten sind dem Netz während des Lernens nicht bekannt. Testdaten können dazu verwendet werden, um die **Generalisierungseffekt** des Netzes zu testen. Testdaten bestehen aus mehreren Testdatensätzen, deren Werte vor einem Arbeitsschritt auf die Eingangsneuronen des Netzes übertragen werden.

3.1.6 Generalisierungsfähigkeit

bedeutet, mit der Leistung eines neuronalen Netzes nach dem Lernen mit den Trainingsdaten auch solche Daten zu erkennen, die nicht in den Trainingsdaten enthalten waren. Diese Leistung beruht darauf, Ähnlichkeiten in den Datensätzen der Trainingsdaten zu erkenn, die jeweils der gleichen Klasse zugeordnet wird.

3.1.7 Lernfaktor

Mit dem **Lernfaktor** kann man einstellen, wie stark eine Gewichtskorrektur während eines Lernschrittes des Netzes erfolgt.

- Wird ein kleiner Lernfaktor gewählt, lernt das Netz nur langsam.
- Wird der Lernfaktor zu groß gewählt, kann es vorkommen, dass das Netz nicht zu einem stabilen Zustand konvergiert, also dass das Lernziel nicht erreicht wird, da beim Gradientenverfahren des Lernens das gesuchte Minimum „übersprungen“ wird.

3.1.8 Gradientenverfahren

Berechnen die Steigung einer Zielfunktion (Fehlerfunktion des Netzes) in Abhängigkeit von den Gewichten des Netzes.
Bei jedem Lernschritt des Netzes wird für alle Gewichte eine Gewichtsänderung um einen Bruchteil des negativen Gradienten der Zielfunktion vorgenommen.
Die Höhe der Änderung der Gewichte pro Lernschritt ist wiederum abhängig vom Lernfaktor, auf welchen dann später im Laufe des Beispiels noch eingegangen wird.

4 Prinzipielle Probleme der Backpropagation-Netze

Der Lernmechanismus entspricht einem sogenannten „Hill-climbing“-Algorithmus. Wenn man sich die Fehlerfunktion als Hügellandschaft vorstellt, entspricht ein bestimmter Ort immer der Einstellung der Gewichte. Die Höhe des Ortes entspricht dem Wert der Fehlerfunktion. Ziel ist es, in das tiefste Tal des Gebirges zu kommen, also den kleinsten Fehler zu erreichen. Die Oberfläche der Fehlerfunktion kann Einbuchtungen und Täler besitzen. Daher ist es mehr oder weniger schwierig, das globale Minimum zu finden.

Es liegt in der Natur des Backpropagation-Algorithmus’, dass er immer in die Richtung läuft, die den steilsten Abstieg bedeutet. Dabei treten zwei Probleme auf:

- Es wird von einer lokalen Linearität im Bereich der eingestellten Schrittweite ausgegangen. Wenn nun der Lernschritt zu groß ist, gilt diese Annahme nicht mehr. Es kann hierbei sogar zum Anwachsen des Fehlers kommen, nämlich dann, wenn in Richtung des steilsten Abstiegs das Fehlergebirge gleich wieder ansteigt. Insofern ist nur bei einer möglichst kleinen Lernrate die Konvergenz in ein lokales Minimum gesichert, was einen wenig praktischen Ansatz darstellt. Eine zu große Lernrate führt aufgrund der nicht mehr vorhandenen lokalen Linearität zu chaotischem Verhalten ohne jede Konvergenz. Zu finden wäre also immer die „goldene Mitte“, wozu die Beschaffenheit des Fehlergebirges bekannt sein müsste.

- Das zweite Problem ist die Gefahr, in ein lokales Minimum zu geraten, das aufgrund der Fehlerfunktion und der Lernregel meist nicht mehr verlassen werden kann. Diesem Problem kann man durch künstlich erzeugtes „Rauschen“ begegnen.

5 Anwendung anhand des S&P 500 und der Einflussfaktoren

5.1 Die Erhebung der zum Training des neuronalen Netzes erforderlichen Daten

		SPX	M1	10 ytbil	unemployed	Fed Rate	Funds	Ind. Prod.Index	Spot Oil
		OUTPUT							
	NODES:	***INPUT:***	***1***	***2***	***3***	***4***		***5***	***6***
1	1981.04	132.81	427.04	13.68	7.2	15.72		80.644	38.000
2	1981.05	132.59	424.42	14.10	7.5	18.52		81.320	38.000
3	1981.06	131.21	425.47	13.47	7.5	19.10		81.830	36.000
4	1981.07	130.92	427.90	14.28	7.2	19.04		82.554	36.000
5	1981.08	122.79	427.84	14.94	7.4	17.82		82.184	36.000
6	1981.09	116.18	427.47	15.32	7.6	15.87		81.519	36.000
7	1981.10	121.89	428.46	15.15	7.9	15.08		80.897	35.000
8	1981.11	126.35	430.90	13.39	8.3	13.31		79.765	36.000
9	1981.12	122.55	436.18	13.72	8.5	12.37		78.870	35.000
10	1982.01	120.40	442.15	14.59	8.6	13.22		77.601	33.850
11	1982.02	113.11	441.49	14.43	8.9	14.78		79.274	31.560
12	1982.03	111.96	442.37	13.86	9.0	14.68		78.720	28.480
13	1982.04	116.44	446.77	13.87	9.3	14.94		77.977	33.450
14	1982.05	111.88	446.52	13.62	9.4	14.45		77.380	35.930
15	1982.06	109.61	447.88	14.30	9.6	14.15		77.121	35.070
16	1982.07	107.09	449.08	13.95	9.8	12.59		76.488	34.160
17	1982.08	119.51	452.46	13.06	9.8	10.12		76.084	33.950
18	1982.09	120.42	457.51	12.34	10.1	10.31		75.526	35.630
19	1982.10	133.72	464.59	10.91	10.4	9.71		74.900	35.680
20	1982.11	138.53	471.12	10.55	10.8	9.20		74.701	34.150

Abbildung 2: Auszug aus dem Input-Datenset

Um eben ein bestimmtes Netz trainieren zu können, benötigt es verschiedener Inputdaten; so in dem Fall des Netzes einer Aktie oder Index zum Beispiel fundamentale Faktoren (wie in unserem Beispiel) oder technische (MACD, Gleitende Durchschnitte, RSI, etc…).

Wir riefen uns hierzu die benötigten Daten im Internet ab und gaben sie in Form einer Excel Tabelle in den Computer ein. Hierbei musste man beachten, die amerikanische Kommaform beizubehalten, eben PUNKT statt BEISTRICH, da sonst die meisten „Netzwerkprogramme“ diese Eingaben nicht bzw. falsch werten, da sie auf die englische Schreibweise hin programmiert wurden.

Im nächsten Schritt formatierten wir diese Tabellen dann in .txt - files

5.2 Aufbereitung der Daten

hierbei muss eine Normierung (Skalierung) erfolgen, da die Backpropagation-Netze Neuronen verwenden, die einen sigmoiden Verlauf der Aktivierung besitzen.

5.3 Anwendung an einem Beispiel

Mit Hilfe des Backpropagation-Verfahrens stellten also wir ein neuronales Netz zusammen, bestehend aus 7 Input-Variablen aus dem Zeitraum April 1981 bis April 2001 per Monatsende:

1. ***Standard&Poor's 500 Indexstand***
2. ***Geldmenge M1***
3. ***10-jährige T-Bills***
4. ***US-Arbeitslosenrate***
5. ***US-Leitzinssatz***
6. ***US-Industrieproduktionsindex***
7. ***Öl-Spotpreis***

Wir verwendeten das Programm *„Qnet for Windows“ Version 2K build 721*.
Beim Aufbau des Netzwerks gingen wir in folgenden Schritten vor.

5.3.1 Dateneingabe

Nachdem wir einige Einflussfaktoren auf den S&P500, die unserer Meinung nach einen großen Einfluß auf den Kurs des Index haben, in Erfahrung gebracht hatten, gaben wir diese in den

Computer ein und konvertierten diese Tabelle in ein „.txt“-File, was aufgrund des Programmes notwendig war und weil neuronale Netze grundsätzlich nur mit solchen Dateitypen arbeiten.

Die Daten mussten nur jeweils unterteilt sein, entweder per Komma oder Beistrich, etc. Es war relativ leicht, die Daten aus dem Excel-Spreadsheet in das erforderliche ASCII (text) format zu transformieren. (***.CSV***), (***.TXT***) oder (***.PRN***) sind praktischerweise alle kompatibel mit Qnet.

5.3.2 Trainingssetup

5.3.2.1 Training Data

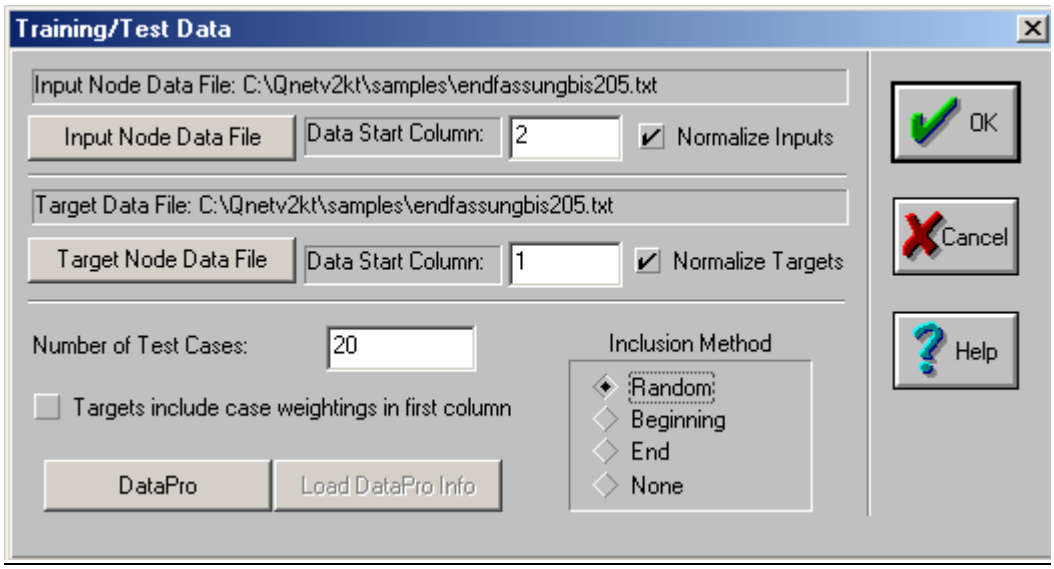


Abbildung 3: Screenshot Training/Test Data

- Zuerst mussten wir nun das Netzwerk trainieren. Hierzu mussten wir sowohl den Eingabe- als auch den Ausgabevektor in das Programm eingeben. Backpropagation-Netzwerke benötigen Zielwerte, die zwischen 0 und 1 liegen. Hierbei ist Qnet besonders hilfreich, da diese Daten vom Programm automatisch normalverteilt werden. Da wir diese Option auswählten, wurden die Daten sowohl für die Input- als auch für die Output-Zielwerte zwischen den Schranken 0,15 und 0,85 in eine Normalverteilung konvertiert.

- Anschließend stellten wir die Anzahl der **Test Cases** ein. In der Literatur wird oft erwähnt, dass eine optimale Anzahl 30 wäre, mindestens aber 3. Als Faustregel kann gesagt werden, dass etwa 1/3 bis ¼ der Testdaten als Evaluationsmenge verwendet werden sollten. Wir hatten 205 Datensätze und einigten uns darauf, 20 von diesen als Test Cases zu verwenden
  (*Training-Set*: 1-185 = April 1981 – August1996;
  Test-Set: 186-205 = September 1996 – April 1998).

- Der Menüpunkt „**Targets Include Case Weightings in First Column**“ im diesem Untermenü („Training/Test Data“) wurde von uns nicht aktiviert, weil wir alle Datensätze anfangs gleich gewichten wollten.

- **Test Set Inclusion Method**
  Hier mussten wir auswählen, welche Datensätze als Testsatz beiseite gelegt werden. Da die Daten in einer systematischen Ordnung, nämlich nach Jahren, geordnet sind, wählten wir die Random-Methode aus.Es wird empfohlen, bei chronologisch ablaufenden Datensätzen die Random-Methode zu wählen, da sie zu den aussagekräftigsten Ergebnissen führt.

Training Parameters

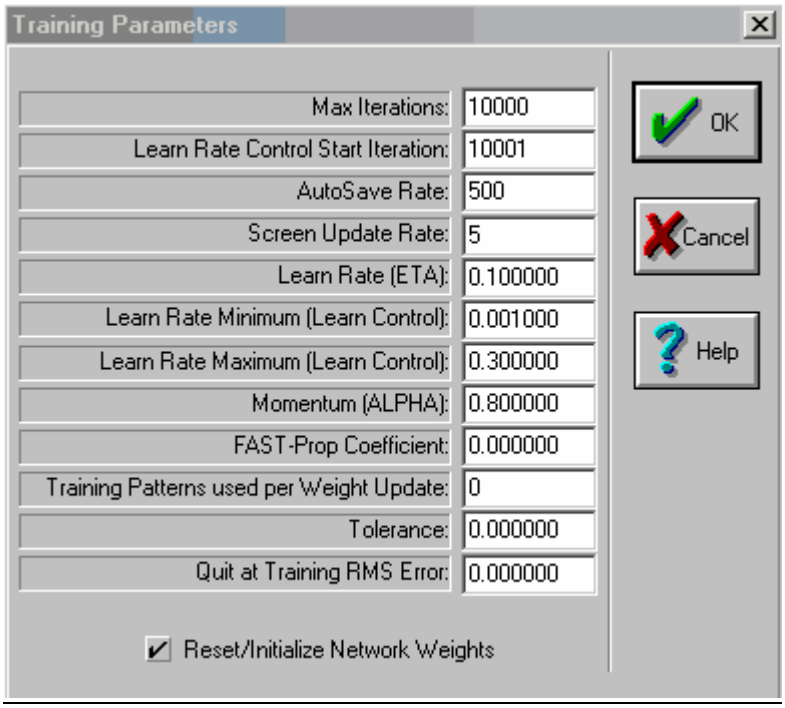


Abbildung 4: Screenshot Training Parameters

Parameter des Netzwerks

- **Maximale Iterationen**

  Wir begannen und beließen es schließlich auch mit 10000 Wiederholungen. Es gibt keine absolute Anzahl an erforderlichen Iterationen für eine bestimmte Datenmenge. Man kann diese Anzahl nur durch Versuch und Irrtum herausfinden. Nachdem sich dann bei der von uns gewählten Wiederholungszahl in unserem speziellen Netzwerk keine großartigen Verbesserungen mehr registrieren ließen, nahmen übernahmen wir eben auch die Standardeinstellung.

- Die **Auto-Save Rate** haben wir nicht verändert, sondern sie bei standardmäßigen 500 belassen. Diese gibt an, wie oft das Programm die Gewichte speichert, um das Netzwerk im Falle eines Übertrainierens zu helfen. Übertraining passiert, wenn das Netzwerk von der Generalisierung zum Auswendiglernen der Daten wechselt. Ein typisches Symptom von Übertrainieren ist ein fallender Training Set-Fehler, während der Test Set-Fehler ansteigt.

- Bezüglich der **Lernrate** ist zu sagen, dass eine Zahl zwischen 0 und 1 zu wählen ist, mit welcher das Netzwerk seine Fehler bzgl. Den Verbindungsgewichten zwischen den Netzwerkneuronen anpasst. Man kann zwischen sehr hohen und sehr niedrigen Lernraten unterscheiden. Der Vorteil hoher Lernraten besteht darin, dass das Netzwerk schnell trainiert wird. Allerdings kann dies dann auch instabil werden. Dies kann man daran erkennen, dass es zu einem plötzlichen Sprung oder zu starken Fluktuationen im RMS-Error kommen kann. Es wird empfohlen, eine niedrige maximale Lernrate von 0,01 bis 0,001 einzustellen. Im Lernverlauf stimmt die tatsächliche Lernrate aber mit der maximalen Lernrate nicht überein, sondern kann sich auch bedeutend darunter bewegen. Wir verwendeten maximale Lernraten von 0,01 und 0,1. Schlussendlich beließen wir die Lernrate bei relativ hohen 0,1, da wir bei diesem Wert die niedrigsten Fehlerwerte erreichen konnten.

- **Momentum** oder **α**

  Dies ist ebenfalls eine Zahl zwischen 0 und 1, die der Rate der Gewichtsveränderung Stabilität verleihen soll. Typische Momentum-Zahlen reichen von 0,2 bis 0,8. Hier führen wiederum höhere Zahlen zu einer höheren Trainingsgeschwindigkeit, was aber wieder zu Instabilität führen kann. Der Momentumfaktor wird auch im Trainingsalgorithmus verwendet. Einige

  Netzwerke trainieren allerdings sogar ohne Momentum. Wir experimentierten mit 0,2 und mit 0,8, und verwendeten dann schlussendlich 0,8 für unser fertiges Netzwerk.

5.3.2.2 Network

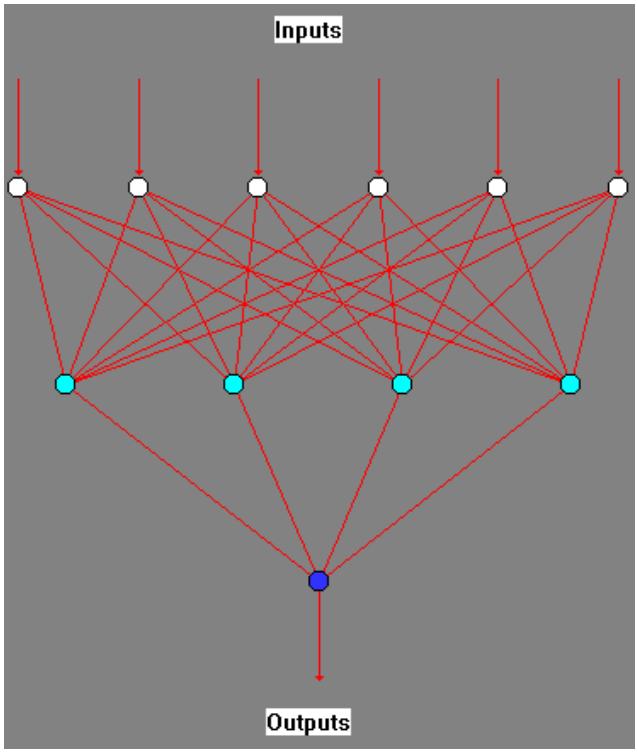


Abbildung 5: Design unseres Netzwerks

In diesem Netzwerkprogramm war die Mindestanzahl für die einzelnen Schichten (wie bei uns verwendet) 3 und die höchste erlaubte Anzahl 10, also 8 verdeckte Schichten.

Bei der Auswahl der Anzahl an Zwischenschichten sind folgende Punkte zu beachten:

- Je mehr verdeckte Schichten, desto besser kann ein Netz auch mit wenig verdeckten Neuronen lernen.
- Allerdings verlangsamt eine höhere Anzahl die Trainingsgeschwindigkeit.
- Es wird empfohlen, bei den meisten Netzen 3 bis 6 Schichten insgesamt zu verwenden, da dies vollkommen ausreichend ist.

- Weiters ist noch zu erwähnen, dass eine zu hohe Anzahl an verdeckten Neuronen zu schlechten Modellen führen kann, die sich dann eben in einem schlechten „Lernprozess“ und auch schlechten Ergebnissen wiederspiegeln (Stichwort: „Auswendiglernen“).
- Eine zu niedrige Anzahl wird allerdings auch zu ungenauen Resultaten führen.

Wir kamen nach mehreren Experimenten zu dem Schluss, bei unserem „Problem“ ein 3-schichtiges Netz mit 6 Input-Neuronen, 4 verdeckten und einem Output-Neuron (SPX) zu verwenden.

Natürlich versuchten wir auch andere Kombinationen bezüglich Schichtenanzahl und Neuronenanzahl je Schicht. Allerdings kam es öfters zu schwankenden Lernverläufen und heftigen Ausschlägen, die man dann auch im Lernfenster beobachten konnte.

Exemplarisch einige Ergebnisse anhand des root-mean-square error (RMS) zwischen den training sets, den targets und den tatsächlichen Netzwerk outputs bei je 10000 Wiederholungen:

z.B: 2 verdeckte Schichten (2 Neuronen in der zweiten Schicht / 2 in der dritten Schicht)

LearnRate			RMS	LR	RMS
0,02				0,1	
Momen.	Training	->	0,015537	MOM	0,015
0,8	Test	->	0,018780	0,8	0,011

2 verdeckten Schichten (3 Neuronen in der zweiten Schicht / 2 in der dritten Schicht)

LearnRate		RMS
0,1		
Momen	Training	0,013976
0,8	Test	0,014969

Abbildung 6: Vergleich Lernrate, Momentum mit Error

Nachdem wir nun unser endgültiges Netz gefunden hatten, es dann auch gelernt hatte, konnten wir dann mithilfe einiger Menüs dann auch noch die Effizienz, bzw, Genauigkeit der Test-Outputs mit den tatsächlichen Outputs vergleichen.

So ist zum Beispiel nach vollendetem Training im Untermenü **„Info“** der Punkt **„Agreement Statistics“** aufrufbar:

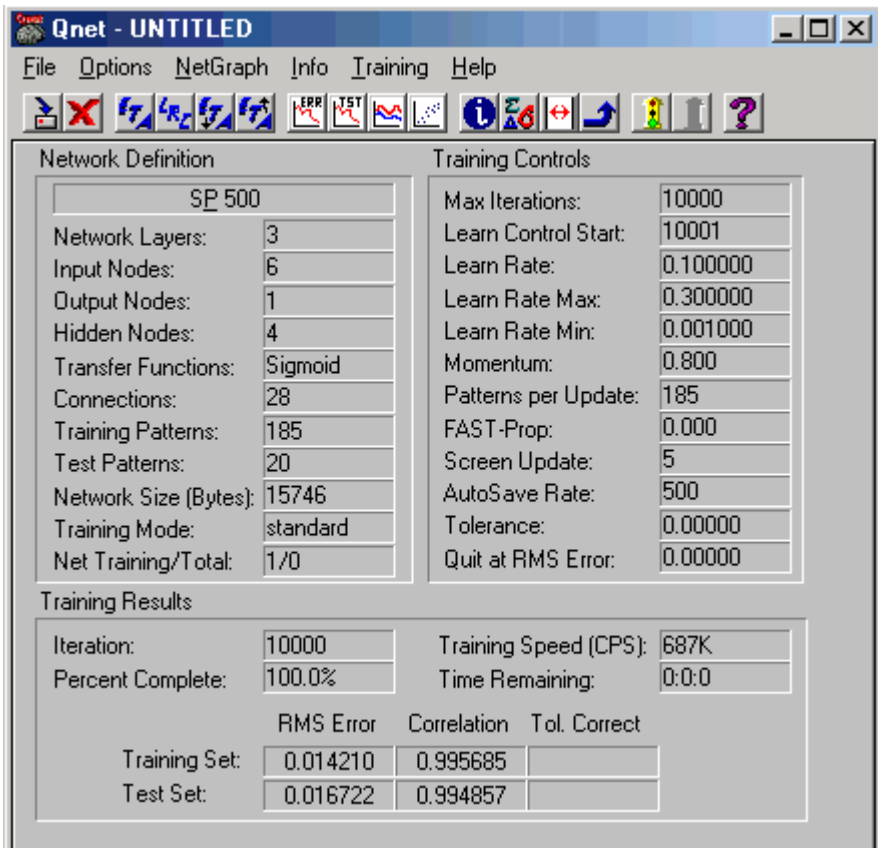


Abbildung 7: Anzeige nach vollendetem Training; 185 Datansätze (training patterns) wurden verwendet

„Agreement Statistics“

Hier kann man sich zum Beispiel die Standarabweichung sowohl des Training-Data-Sets als auch des Test-Data-Sets anschauen. Ebenso auch die maximale Abweichung, die im Laufe des Trainingsdurchlaufs zw. Output und vorgegebenen Daten registriert wurde.

„Tolerance Checking“

Ein weiterer hilfreicher Punkt; hier kann man einen beliebigen Wert einsetzen (TOLERANZWERT), um die die Outputs von den „wahren Werten“ abweichen dürfen. Anschliessend erhält man dann eine Auswertung, welche Anzahl der Werte in dieses Intervall fallen, bzw. welche nicht. Man kann dies als Wahrscheinlichkeits-Intervall annehmen.

„Input Node Interrogator“

Interessant, da man hier die Information bekommt, welchen Anteil welcher Inputfaktor an dem Outputfaktor bei dem Trainingsdurchgang hatte. Dies ist auch insofern hilfreich, um eben die geeignetesten Faktoren, die man doch selber auswählen muss, für das zu lernende endgültige Netzwerk zu bestimmen. So haben wir anfangs auch andere Faktoren dem Netzwerk zugrunde gelegt, diese Ideen dann aber nach der Auswertung wieder verworfen.

Weitere Punkte:

- Die eingeteilten Gewichtung der Neuronen untereinander sind ebenso abfragbar wie das BIAS-NEURON
- Man kann sich (wie bei den meisten Punkten) neben den Zahlen alleine auch die dazugehörigen Grafiken anzeigen lassen. Dies ist zum Beispiel bei dem RMS-Minimierung im Verlauf des Trainings möglich. Hier wird insbesondere veranschaulicht, dass der grösst Lernerfolg immer zu Beginn eines jeden Trainings erreicht wir und dann am Ende zunehmends abnimmt, bzw. sich sogar wieder verkehrt.
- Weiters ist es auch möglich mittels dem Menüpunkt „Color Contours“ den Einfluss von 2 Input-Faktoren auf den Output in einem Farbschema anzeigen zu lassen. Man kann dies auch als eine Form der grafischen Anzeige der Gewichtung, bzw. Einfluss einzelner fundamentaler Faktoren, zum Beispiel Geldmenge und Ölpreis auf den S&P 500 bewerten.

Targets/Outputs vs. Pattern Number

Dieser Punkt gibt eben ein Bild über die Differenzen zwischen targets & tatsächlichen Outputs:

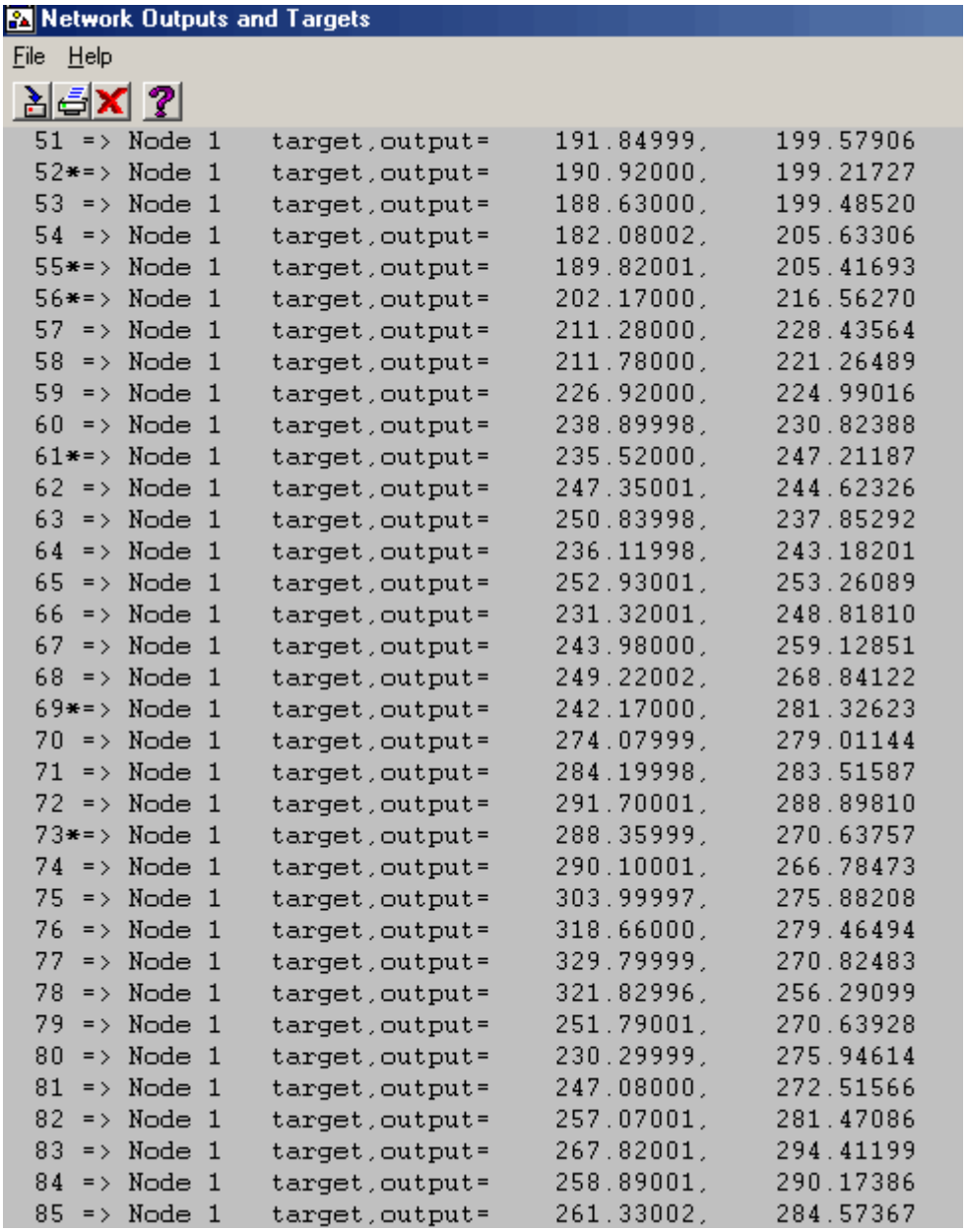


Abbildung 8: Ausschnitt der Datensätze inklusive der Test-Sätze

Wie bereits erwähnt ist aber auch eine Ausgabe dieser Daten als Grafik möglich, daher sieht man auch auf dem folgenden Bild den Verlauf der einzelnen Werte.

ROT = vorgegebene Outputwerte

GRÜN= Trainingswerte

BLAU = Testwerte

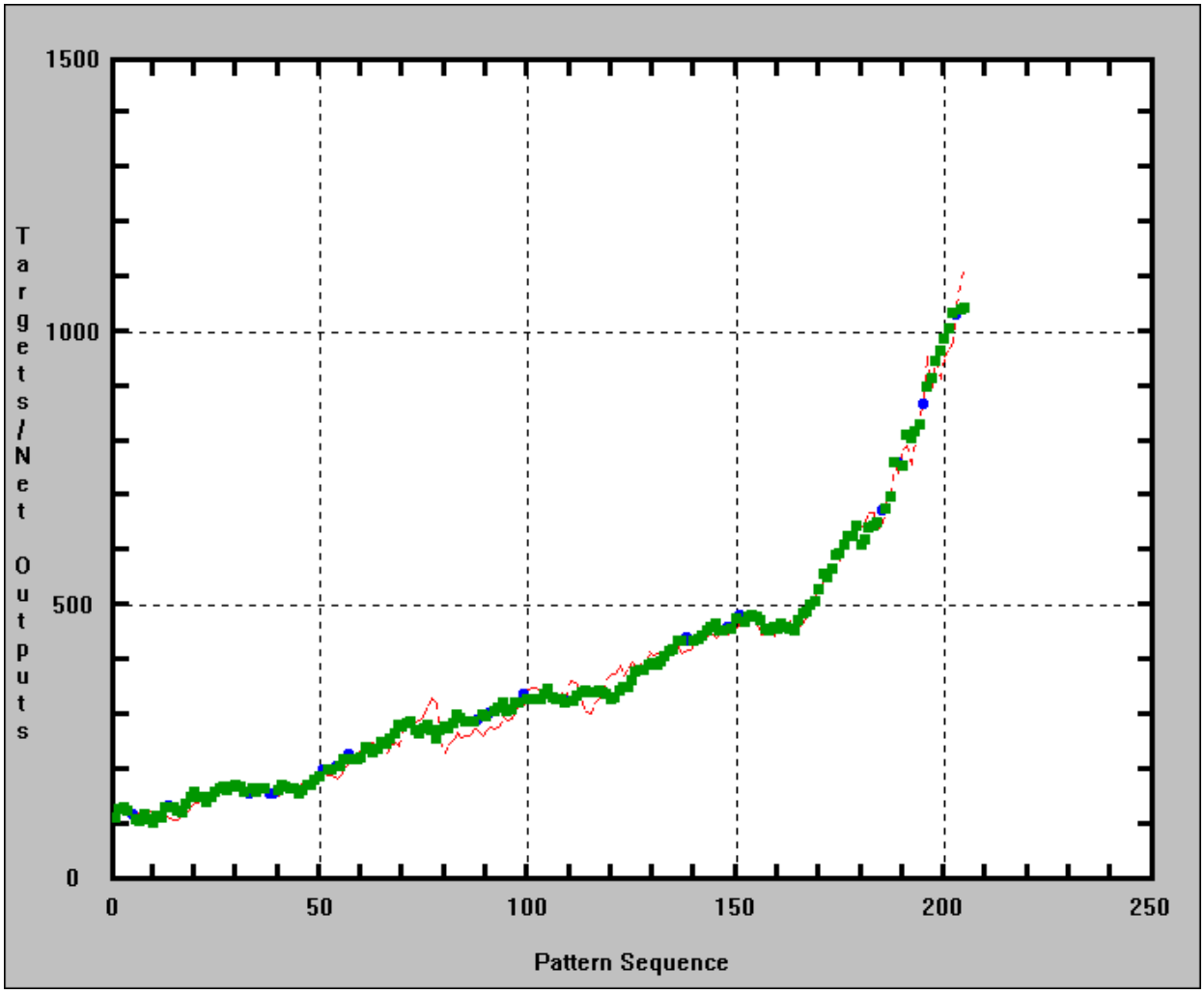


Abbildung 9: Differenzen zwischen targets & tatsächlichen Outputs als Grafik

5.3.3 Experimente

Hier ist anzumerken, dass man tatsächlich nur durch Versuch & Irrtum an ein annähernd besseres Ergebnis kommt; die einzig wahre Formel gibt es nicht.

1. Versuch

Begonnen haben wir mit einem einfachen Netzwerk, bestehend aus einer Eingabe-, einer verdeckten und einer Ausgabeschicht. In der verdeckten Schicht befanden sich 2 Neuronen. Wir verwendeten die Standardeinstellungen des Programms mit 6 Eingabeneuronen. Bei den nächsten Versuchen änderten wir verschiedene Trainingsparameter.

2. Versuch

Nun erhöhten wir die Lernrate unter Beibehaltung der Schichtenanzahl von 0,01 auf 0,1. Die restlichen Parameter blieben unverändert.

x. Versuch

Nachdem wir beispielsweise in einigen vorangegangenen Versuchen mit 4 Schichten, 10000 Iterationen (beliessen wir jedes Mal einfachtheishalber unverändert) und je 2 und 2 Neuronen in den verdeckten Schichten experimentiert hatten, was zu einem viel zu hohem RMS-Fehler führte, entschlossen wir uns, ein Netzwerk mit 4 Schichten und jeweils 3 bzw. 2 Neuronen in den verdeckten Schichten zu entwickeln.

x. Versuch

Wir versuchten auch das Momentum von 0,8 bis auf 0,2 zu reduzieren, was allerdings zu einem zu großen RMS-Fehler sowohl im Training als auch im Test Set führte.

x. Versuch

In unserem vorletzten Versuch wählten wir eine erhöhte Lernrate von 0,5 und verringerten das Momentum von 0,8 auf 0,6, um ein stabileres Netzwerk zu erreichen. Allerdings befand sich der RMS-Error sowohl im Training als auch im Test Set noch immer über den erreichten Werten.

5.3.4 7. Versuch – Optimum

Hier verwendeten wir , wie bereits erwähnt 3 Schichten mit 10000 Iterationen (Standardeinstellung), Lernrate 0,1 und das Momentum setzten wir auf 0,8. Dies erzielte das beste Ergebnis aller von uns durchgeführten Versuche mit einem niedrigen RMS-Fehler.

5.3.5 Evaluierung, Recall-Modus

Vergleich der Trainings- und Testoutputs mit den Trainingszielen (= S&P 500, Spalte 1)

Wir führten anschließend im „Recall-Modus“ mit Hilfe unseres erstellten Netzwerkes anhand der vorhandenen Parameter (bis auf den S&P 500) einen Versuch durch, die jeweiligen Zielwerte des S&P 500 für die letzen 36 Monate (Zeitraum Mai 1998 bis April 2001, ab Datensatz 206) vorherzusagen (besser = nachzuprüfen): Das Ergebnis wich von den tatsächlichen Werten ab, ***allerdings fielen die letzten beiden Monate des Zeitraumes der Targets als auch der Outputs wieder fast aufeinander***. Man könnte dies als Rückkehr zur Normalität deuten, bzw. Rückbesinnung auf fundamentale Einflussfaktoren, die den Indexstand bestimmen („sollten“).

Hier ist anzumerken, dass die Jahre zuvor der Überschwang an Euphorie (Stichwort: Internet, irrational exuberance) nicht vorhanden war, bzw. nicht in regelmäßigen Mustern, daher auch nicht in dem Maß „miteingelernt“ wurde.

Resumé: Die Vorhersage hätte also noch für einige Monate einen ansteigenden Trend vorhergesagt, allerdings dann eine Seitwärtsbewegung, entgegen dem tatsächlichen Verlauf, welcher auch trotz der Warnungen vor der „Spekulationsblase“ im Aktienmarkt durch den amerikanischen Notenbankchef Alan Greenspan STARK NACH OBEN zeigte.

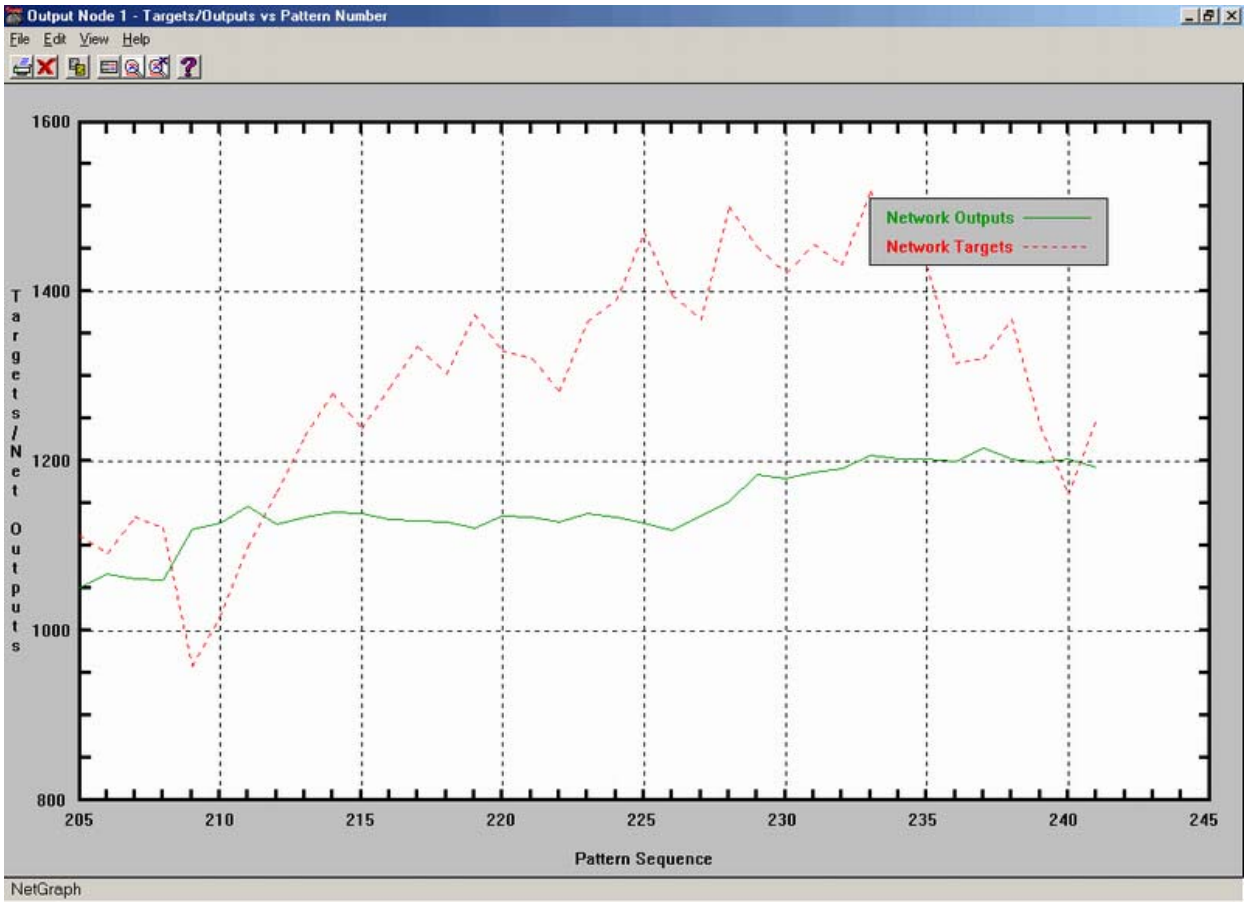


Abbildung 10: Evaluierung im Recall-Modus, Grafikvariante

6 Zusammenfassung

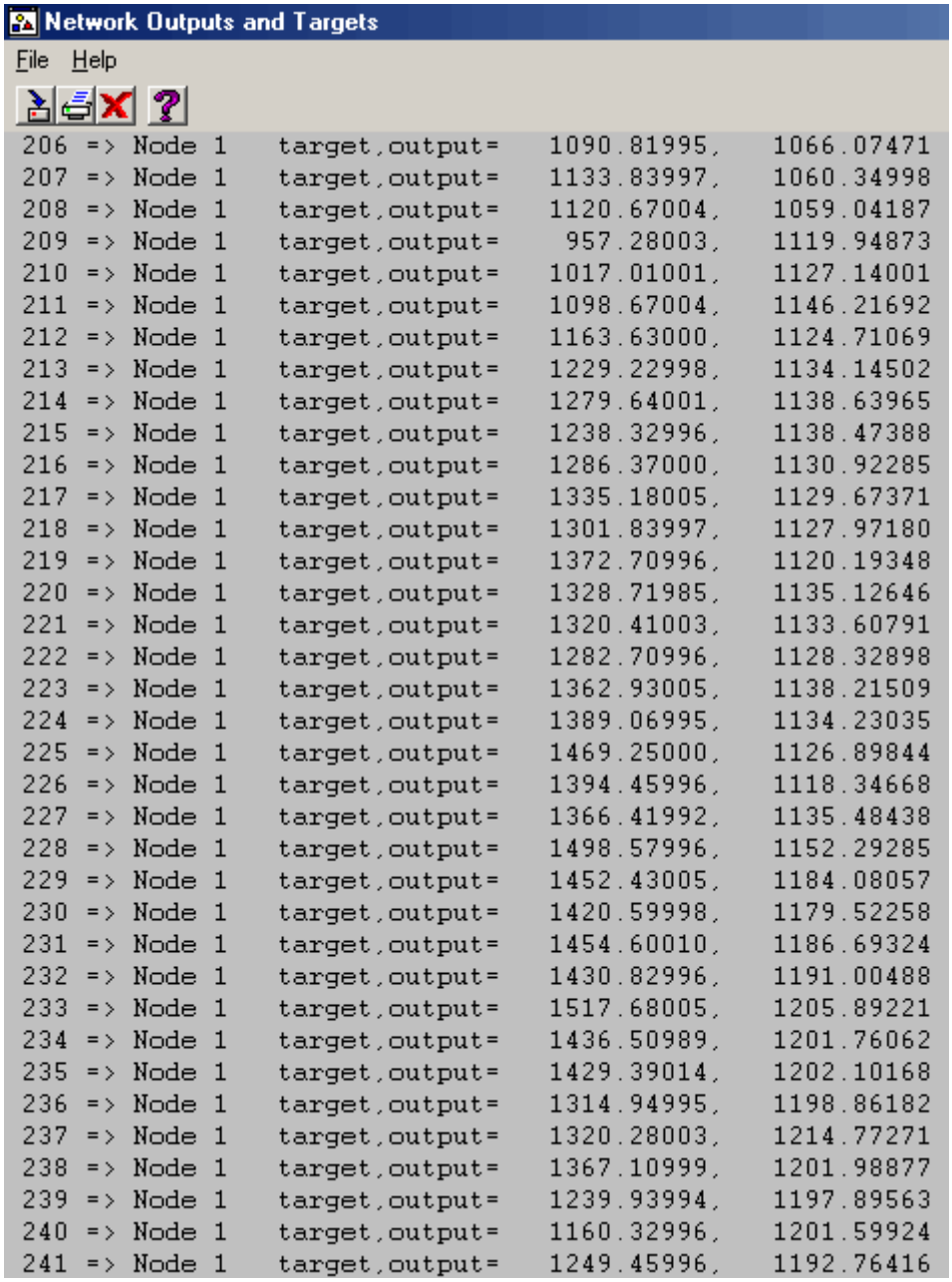
6.1 Neuronale Netze und Prognosen

Prinzipiell können Neuronale Netze für beliebige Aufgabenstellungen eingesetzt werden. Voraussetzung für einen erfolgreichen Einsatz dieser Technik ist aber in jedem Fall, dass die Daten oder Muster, auf die das Neuronale Netz angesetzt wird, nicht bloße Zufallswerte (so zum Beispiel im LOTTO) ohne irgendeinen Sinn oder Zusammenhang darstellen. Es kann mit anderen Worten nur dann von einem Neuronalen Netz erwartet werden, dass es eine Entwicklung oder einen Zusammenhang entdeckt, wenn in den zur Verfügung stehenden Mustern eine solche Entwicklung oder ein solcher Zusammenhang überhaupt ablesbar ist.

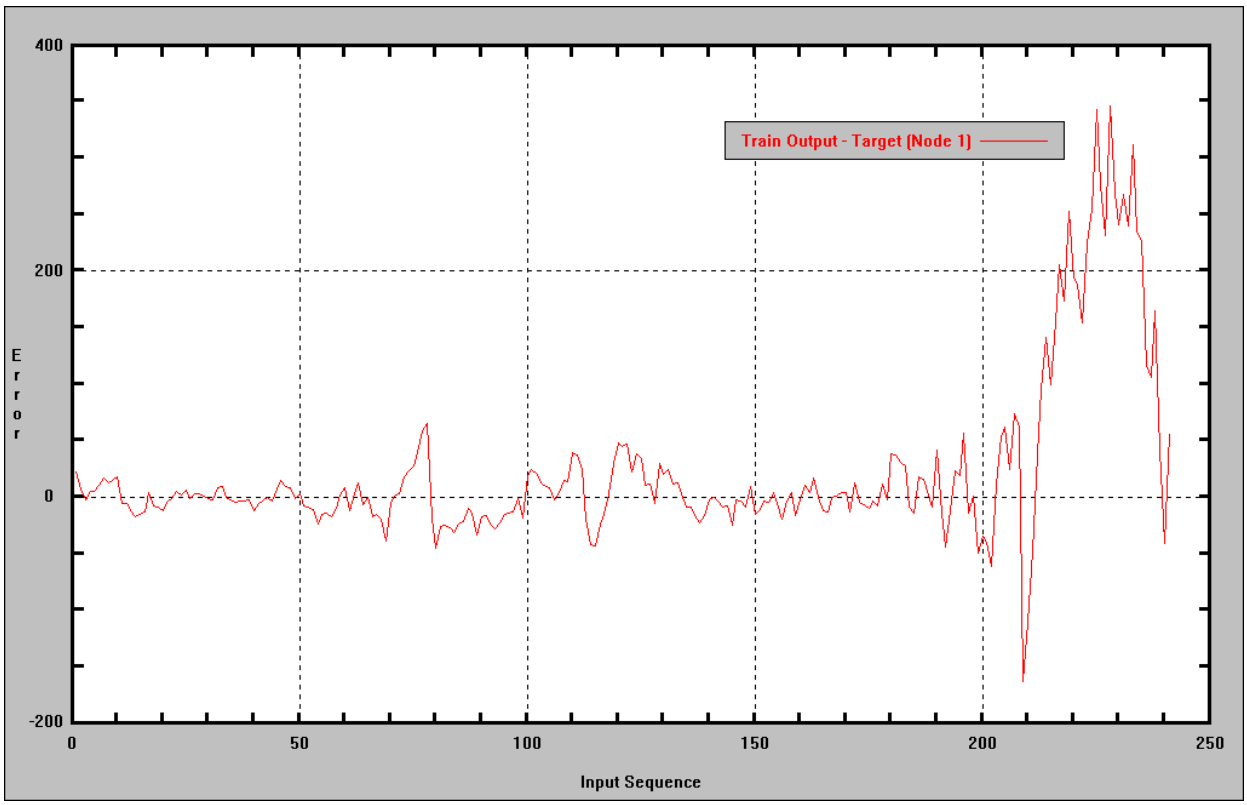
6.2 Neuronales Netz prüft Theorien

Umgekehrt kann ein Neuronales Netz auch dazu eingesetzt werden festzustellen, ob bestimmte Daten oder Analyse-Ansätze prinzipiell aussagekräftig sind: Wenn man zum Beispiel annimmt, dass sich die Kursentwicklung eines Titels aufgrund von bestimmten Indikatoren vorhersagen lässt, kann man mit einem Neuronalen Netz feststellen, ob oder wie gut diese Vermutung zutrifft. Abhängig davon, wie gut oder wie schlecht ein Neuronales Netz eine bestimmte Entwicklung auf der Basis eines bestimmten Inputs prognostizieren kann, lässt sich sagen, welche Aussagekraft diese Inputs für die Entwicklung besitzen. Der Prozess der Entwicklung von Neuronalen-Netz-Modellen hat insofern zwei Richtungen. Zum einen lernen die Netze etwas über den Zusammenhang der Daten zum Erstellen möglichst guter Prognosen. Zum anderen lernt aber auch der Entwickler des Modells etwas darüber, welche Daten, Indikatoren und Analyseverfahren für einen bestimmten Prognosezweck etwas bringen und welche nicht. In unserem Fall haben wir uns auf diese Theorievariante beschränkt; aufgrund der Software war es uns auch nicht anders möglich.

Evaluierung der Ergebnisse im Recall-Modus (Daten-Variante):



Schließlich führten wir auch eine Auswertung der Abweichungen zwischen „errechnetem“ Ergebnis und tatsächlichem Indexstand des S&P 500 im Zeitraum Mai 1998 (*206*) bis April 2001 (*241*) durch. Die Ergebnisse haben wir in Grafikform durch das Untermenü **„Targets – Network Output Error vs Input Sequence Number“** folgend abgebildet:



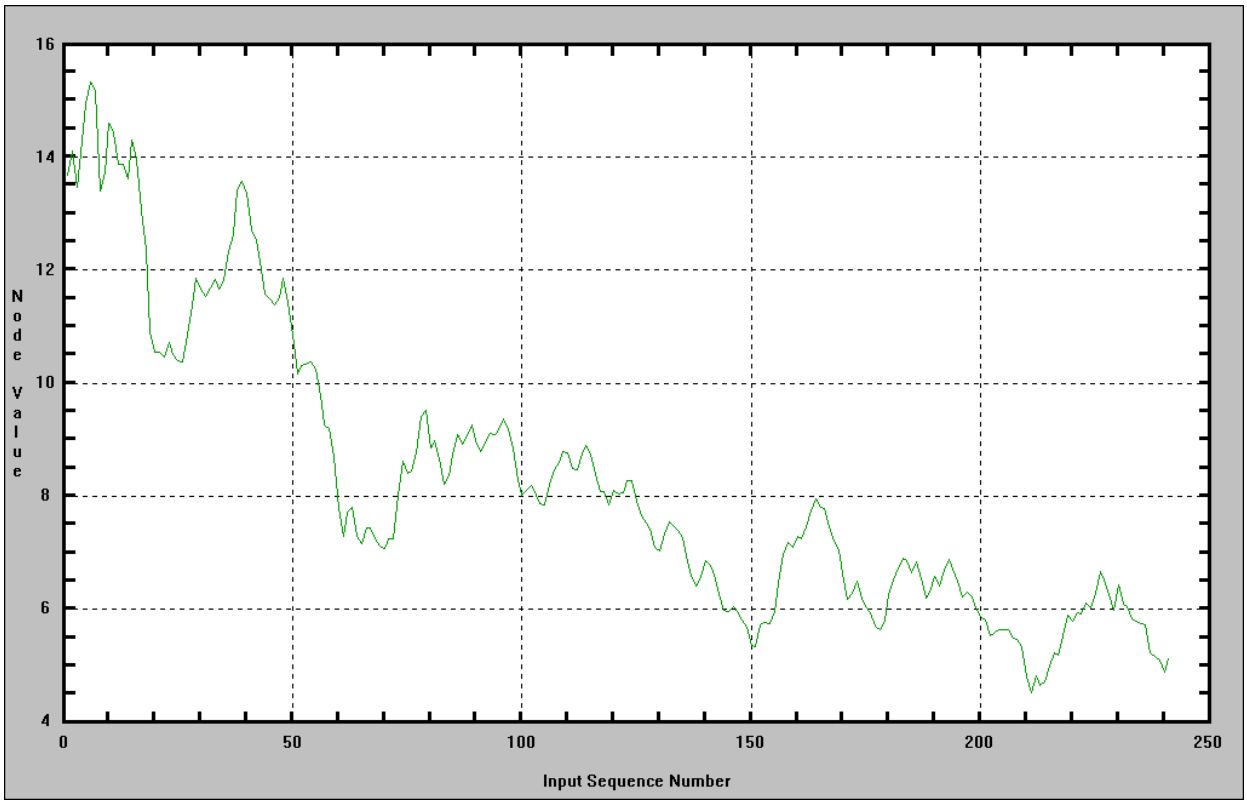
Hier sind die „Fehler“ pro Datensatz aufgelistet. Man kann gut erkennen, dass das Kursverlaufmuster im Recall-Modus sprunghaft angestiegen ist. Man KÖNNTE dies naürlich auf ein fehlerhaftes Lernen zurückführen. Unserer Meinung nach wird hier allerdings die starke Abweichung von dem bisherigen Zusammenspiel der einzelnen Faktoren erklärt. In der Recall-Phase (206-241) ist zuerst sowohl eine „Untertreibung“ als auch in weiterer Folge eine „Übertreibung“ basierend auf den gelernten Mustern in der Lernphase zu erkennen.

In Zahlen gefasst konnte man dies auch durch das Untermenü **„Network Statistics“** abrufen:

So belief sich die Standardabweichung, mit welcher man sich dann auch ein Konfidenzintervall errechnen kann, auf 77,30463 (Punkte), also in 68% der Fälle traf das System im *Intervall von [x +/- 1 Std.abw.] 154,6 ins Schwarze.*

Allerdings ist natürlich auch der **„Max Error“** ersichtlich gewesen, so ist zum Höchstpunkt ein „Fehler“ von 346,28711 Punkten registriert worden.

Abschließend ist es auch noch möglich, sich die einzelnen Input-Faktoren je Datansatz insgesamt anzuzeigen. Hierzu ist im Recall-Modus unter dem Punkt **„NetGraph“** (wie bei jeder Grafikanzeige) das Untermenü **„Input Nodes vs. Pattern Sequence“** anzuklicken. Als Beispiel haben wir hier die „10-jährigen US-Schatzanleihen“ (10y-t-bill) gewählt:



Hier könnte man schon beginnen zu argumentieren, dass fundamental geshehen ein starker Anstieg im S&P 500 mit einem starken Anstieg von 10y-t-bill-notes nicht einhergehen kann, bzw. normalerweise eher nicht anzunehmen ist.

Ebenso zeigte unser Netz ca. bei Datensatz 210 einen Anstieg des Indexstand an, was auch mit einem einhergehenden Fall der US-Schatzanleihen verständlich gewesen wäre. Tatsächlich kam es aber zu diesem Zeitpunkt zu einem Fallen des S&P 500 und nachher zu einer monatelangen Hausse. Allein aus dieser Sichtweise heraus kann man schon erkennen, wie ein Neuronales Netz bei der Beurteilung von Sachverhalten behilflich sein kann.