TECHNISCHE UNIVERSITÄT MÜNCHEN

Sinus-Netzwerk

D. Egger

IAPG / FESG No. 22

Institut für Astronomische und Physikalische Geodäsie
Forschungseinrichtung Satellitengeodäsie

München 2006

Sinus-Netzwerk

D. Egger

IAPG / FESG No. 22

München 2006

ISSN 1437-8280
ISBN 3-934205-21-6

Hinweis: Eine PDF-Version dieser Arbeit mit farbigen Abbildungen ist erhältlich unter
http://tau.fesg.tu-muenchen.de/~iapg/web/veroeffentlichung/schriftenreihe/schriftenreihe.php

Adressen:
Institut für Astronomische und Physikalische Geodäsie
Technische Universität München
Arcisstrasse 21
D-80290 München
Germany
Telefon: +49-89-289-23190
Telefax: +49-89-289-23178
http://tau.fesg.tu-muenchen.de/~iapg/

Forschungseinrichtung Satellitengeodäsie
Technische Universität München
Arcisstrasse 21
D-80290 München
Germany
Telefon: +49-89-289-23191
Telefax: +49-89-289-23178
http://tau.fesg.tu-muenchen.de/~fesg/

Dieter Egger

Sinus-Netzwerk

Vorwort

Beeindruckt von der immensen Leistungsfähigkeit biologischer neuronaler Netze versucht man schon seit vielen Jahren hinter deren „Geheimnis“ zu kommen. Man möchte gerne von ihnen profitieren und entwirft dazu Modelle und Programme für einen Computer. Das gelingt aber nur zum Teil. Schließlich hat das Vorbild um die 100 Milliarden Neuronen, von denen jedes bis zu tausend Verbindungen zu anderen aufrecht erhält.

Mindestens eine interesssante Anregung kann man diesem Bestreben entnehmen: Einfache Elemente können im Verbund erheblich komplexere Aufgabenstellungen erledigen als einzelne für sich genommen.

Bei der Simulation auf einem Rechner sprechen wir aber besser nicht von neuronalen Netzen, denn solche gibt es nur in der biologischen Vielfalt unserer Natur, sondern von Systemen miteinander kommunizierender Einzelelemente, sogenannten Netzwerken.

Möchte man nicht-lineare Zusammenhänge erfassen, so müssen auch die Elemente nicht-lineare Übergänge aufweisen. Und wenn periodische Vorgänge im Spiel sind, könnten periodische Übertragungsfunktionen von besonderem Nutzen sein.

In der vorliegenden Arbeit werden die Elemente mit einer Sinus-Funktion ausgestattet. Durch passende Verschaltungen der Sinus-Elemente zu einem Sinus-Netzwerk können dann vielfältige Geschehnisse aus der Realität, beispielsweise Zeitreihen, abgebildet werden. Mit dem „Begreifen“ des vorgelegten Datenmaterials geht zum Glück auch die Fähigkeit einher, Daten außerhalb des bekannten („gelernten“) Bereiches wiederzugeben.

Vor allem die Prognosefähigkeit stellt eine besonders hervorzuhebende Stärke dar. Interesssanterweise genügt oft eine geringe Anzahl von Elementen (wenige hundert) um bemerkenswerte Leistungen zu vollbringen. Würde man diese Systeme nicht nur auf einem herkömmlichen Rechner simulieren, sondern auf Schaltkreis-Ebene hardwaretechnisch realisieren, so könnte das „Begreifen“ womöglich gar in Echtzeit ablaufen.

Bemerkungen bitte an: dieter.egger@bv.tum.de

WWW des Autors: www.dieteregger.de

Sinus-Netzwerk-Simulator/Beispiele:

http://tau.fesg.tu-muenchen.de/~dieter/enn/netzwerk/netzwerk.php

Falls Sie den Sinus-Netzwerk-Simulator selbst einsetzen möchten, steht dem nichts im Wege. Allerdings bitte ich bei Publikationen um einen entsprechenden Vermerk.

Dieter Egger, München, den 11.11.05

Inhalt

Theorie

Die Idee, einzelne Elemente zu verbinden und dadurch zu einem größeren Ganzen werden zu lassen, ist nicht neu. Es handelt sich vielmehr um eines der erfolgreichsten Konzepte der Evolution. Zellen bilden Zellverbände (Organe) und diese wiederum Organismen, also Lebewesen. Wobei Zellen selbst aus kleineren Funktionseinheiten bestehen, sodaß in jeder Hierarchiestufe Systeme von kommunizierenden Elementen vorliegen. Diese Elemente können dabei durchaus verschiedene Eigenschaften besitzen.

Eines der komplexesten (Teil-)Systeme, die von der Natur hervorgebracht worden sind, ist das menschliche Gehirn. Viele hundert Milliarden Neuronen sind hier jeweils tausendfach verschaltet. Jedes Neuron bildet quasi ein Element mit einer nicht-linearen Übertragungsfunktion, das mit vielen anderen Neuronen kommuniziert.

Auch Naturwissenschaften und Technik profitieren allenthalben von diesem Konzept. Die Elemente heißen dann vielleicht Komponenten oder Bestandteile, aber auch sie kommunizieren miteinander weil sie verschraubt, verklebt oder verschweisst sind. Die Kommunikation ist dann vielleicht nur die Übertragung von Kräften oder Schwingungen, aber sie findet statt. Sie kann auch indirekt über Rechner oder Regelkreise erfolgen.

In der Mathematik gibt es beispielsweise eine Theorie der Funktionenräume. Man wählt mit Bedacht passende Basisfunktionen aus und verknüpft sie um damit beliebige andere Funktionen darzustellen.

In der Automobilindustrie werden in Fließbandarbeit Komponenten eines Fahrzeugs zusammengefügt um schlußendlich ein fahrtüchtiges Vehikel zu produzieren. Auch hier findet letztlich eine vielfältige Kommunikation statt, damit das Fahrzeug auch fährt.

Beim modernen Software-Engineering fügt man Softwarekomponenten (Module, Objekte) zusammen um eine gestellte Aufgabe zu lösen. Zwischen den Objekten werden Botschaften hin und hergeschickt. Darin liegt ja gerade die Stärke der objektorientierten Programmierung.

In der (Elementarteilchen-) Physik spricht man geradezu von Bausteinen, die miteinander wechselwirken, also kommunizieren. Hier wurde der Systembegriff zu einer der tragenden Säulen. Aber nicht nur dort, auch in den Geowissenschaften ist er nicht mehr wegzudenken („System Erde“).

Allgemeines

Wegen der Vielzahl (bio-)chemischer Prozesse sind „reale“ neuronale Netze nicht einfach modellier- und simulierbar, wenn überhaupt. Zudem ist der elektrische Signalfluß pulsförmig.

Damit wir trotzdem so etwas ähnliches wie ein neuronales Netz auf einem Rechner simulieren können, beschränken wir uns auf einfach zu realisierende Elemente. Jedes Element empfängt Signale von anderen Elementen, gewichtet und addiert sie auf und speist die Summe einer nicht-linearen Übertragungsfunktion ein. Deren Output wiederum wird an andere Elemente geschickt.

$$O_i = f\left(\sum_{j=0}^{j=i-1} O_j g_{ji}\right)$$

Der Output des i-ten Elementes hat eine mathematisch fassbare Übertragungscharakteristik f und die Vernetzung erfolgt rückkopplungsfrei ($j<i$).

Elemente

Auch wenn alle möglichen nichtlinearen Funktionen als Übertragungsfunktionen in Frage kommen, wählen wir eine einfache, die zudem periodisch ist. Sie ist überall stetig, stetig differenzierbar und im Wertebereich (trotz unendlichem Definitionsbereich) auf ein kleines Intervall beschränkt. Unser Netz wird diese Wahl mit Stabilität und „Gutmütigkeit“ honorieren.

Auswahl

In der Literatur über neuronale Netze ist oft von der Sigmoid-Funktion die Rede. Ihr Verlauf könnte durchaus eine plausible Antwort eines Neurons auf eine steigende Anregung widerspiegeln, vorausgesetzt man übersetzt die Impulsfrequenzen in bedeutungsgleiche Amplituden.

Setzt man die Sigmoid-Funktion $s(t)=\frac{1}{1+e^{-ct}}$ als Übertragungsfunktion ein, wobei der Parameter c die Steilheit des Übergangs von 0 nach +1 bestimmt, so erhält man zwar funktionierende Netzwerke, aber zu einem hohen „Preis“. Denn zum einen werden sehr viele Elemente benötigt und zum anderen ist die Lerngeschwindigkeit sehr niedrig.

Es hat sich als günstiger erwiesen, die Elemente mit etwas mehr „Funktionalität“ auszustatten, insbesondere bereits die Periodizität mit hinein zu packen.

Aus der Fülle der in Frage kommenden Funktionen greifen wir die allgemeine Sinus-Funktion heraus, also

$$y=a\sin(\omega t+\phi)$$

mit der Amplitude a, der (Kreis-) Frequenz $\omega=2\pi f$ und der Phase ϕ und erfassen damit auch die Kosinus-Funktion. t suggeriert hier zwar die Bedeutung Zeit, repräsentiert aber letztlich nur die gewichtete Summe der Input-Werte.

Netzwerk

Die Verschaltung der Elemente führt uns zum Netzwerk. Jedes Element könnte zumindest theoretisch mit jedem anderen Element verbunden werden. Allerdings gibt es dann auch jede Menge von Rückkopplungen. Und diese können das gesamte Netzwerk schnell in „tiefe Abgründe“ führen. Man denke nur an unerwünschte Resonanzen und chaotische Verhaltensweisen. Wegen dieser Unwägbarkeiten werden wir auf Rückkopplungen verzichten, auch wenn ihr gezielter Einsatz durchaus reizvoll sein kann.

Struktur

Man kann feste Strukturen für die Vernetzung vorschreiben oder aber vollkommen dynamische Strukturen erlauben, die sich erst im Verlauf eines Lernverfahrens herauskristallisieren.

Einige Elemente werden Informationen von der „Außenwelt“ ins Netz einbringen und einige werden Ergebnisse des Netzes an die „Außenwelt“ zurückgeben. Das ist im Prinzip nur eine Formsache, liefert aber wohldefinierte Schnittstellen für das Netz.

Die beiden folgenden Beispiele für ein „paralleles“ und ein „serielles“ Netz stecken die Grenzen ab, innerhalb derer wir die Netzwerkstruktur für ein dynamisch resultierendes „gemischtes“ Netz wiederfinden können.

Parallel

Von den n Elementen fungiert eines als lineares Eingabe-Element, das n-2 nicht-lineare Elemente „versorgt“. Diese n-2 Elemente sind alle mit einem weiteren linearen Element, dem Ausgabe-Element verbunden. Sonst gibt es keine Verknüpfungen, insbesondere sind die nicht-linearen Elemente nicht untereinander verknüpft. Dann erhalten wir

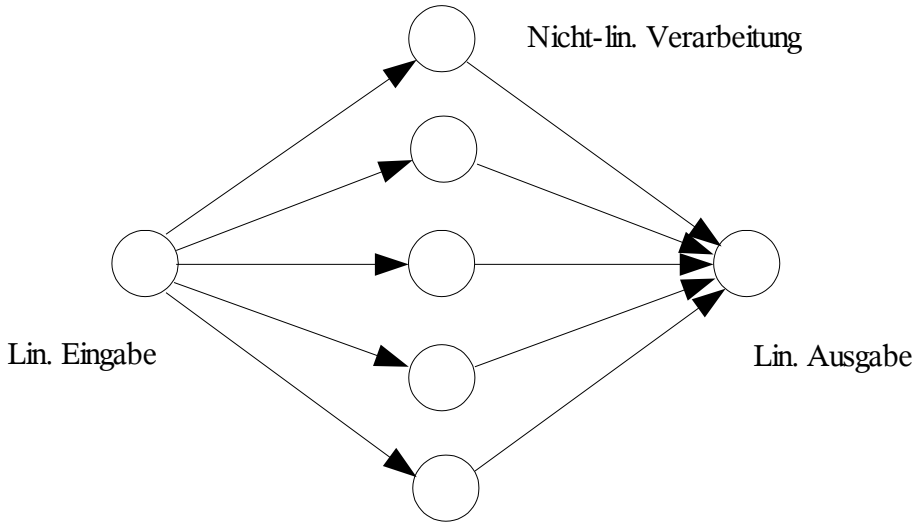


Abbildung 1: Eine Elementschicht zur parallelen Verarbeitung des Eingabewertes könnte eine Fourier-Reihe nachbilden.

als Input für das letzte Element (Ausgabeelement)

$$i_{n-1}=\sum_{i=1}^{n-2} a_i \sin(\omega_i O_0+\phi_i)$$

Darin steckt als Spezialfall die Fourier-Reihe, wodurch offensichtlich wird, daß der Ansatz für die Darstellung von Zeitreihen geeignet sein wird. Es handelt sich um die Superposition von Basisfunktionen, wobei der Output des Eingabe-Elements

$O_0=a_0 t+\phi_0$ als lineare Funktion der Zeit selbst als Zeit aufgefasst werden darf. Das Ausgabeelement kann noch eine lineare Abbildung des Inputs i_{n-1} durchführen:

$O_{n-1}=a_{n-1} i_{n-1}+\phi_{n-1}$ um das Endergebnis abzuliefern.

Für eine gegebene Zeitreihe könnte man die Parameter der Übertragungsfunktionen auch noch analytisch ermitteln (Fourier-Zerlegung) und bräuchte dann gar kein „Lernverfahren“.

Seriell

Die andere Struktur-Grenze ergibt sich durch die bloße Hintereinanderschaltung der Elemente.

Als Resultat erhält man eine (n-2)-fache Verschachtelung der Übertragungsfunktionen, deren Parameter nun nicht mehr trivial zu bestimmen sind.

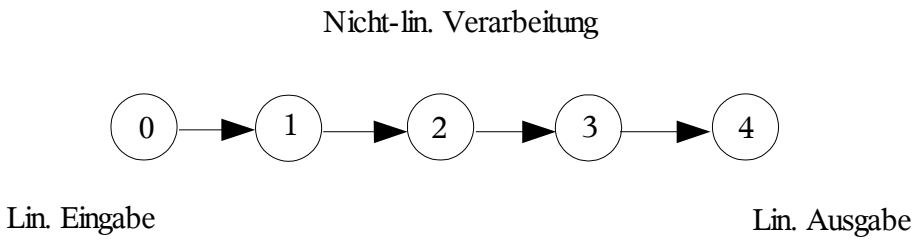


Abbildung 2: Die serielle Verkettung der Elemente führt zu nahezu undurchschaubaren nichtlinearen Abhängigkeiten.

$$i_4 = a_3 \sin(\omega_3 a_2 \sin(\omega_2 a_1 \sin(\omega_1 o_0 + \phi_1) + \phi_2) + \phi_3)$$

Nun ist keine Superposition mehr gegeben und außer den beiden Elementen 0 und 1 erhält auch kein weiteres mehr die lineare Zeit als Input-Wert.

Gemischt

Irgendwo zwischen dem parallen und dem seriellen Fall wird sich eine dynamische Netzstruktur entwickeln. Bei einer vorgegebenn Maximalanzahl von Elementen wird der Aufbau des Netzes zu einem wichtigen Bestandteil des Lernverfahrens.

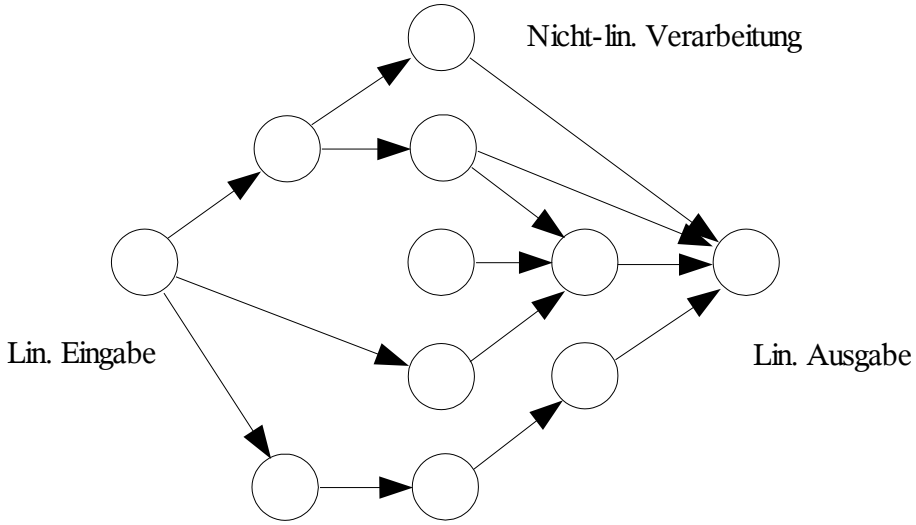


Abbildung 3: Beispiel für eine gemischte Netzstruktur

Auf die formelmäßige Darstellung des Inputs für das Ausgabeelement sei verzichtet. Das soll aber nicht heissen, daß sie nicht möglich wäre. Ganz im Gegenteil: Mit einem vom Simulationsprogramm selbst erstellten (C-) Programm kann das „gelernte“ Wissen des Netzwerks auch an andere Anwendungen übermittelt werden.

Die „gemischte“ Vorgehensweise führt aufgrund zahlreich durchgeführter Versuche am schnellsten zum Erfolg und erhält daher den Zuschlag für die Untersuchung weiterer Zeitreihen. Weder die Parameter der Übertragungsfunktionen, noch die Netzstruktur selbst, können direkt berechnet werden. Daher ist ein geeignetes Lernverfahren zu durchlaufen, das im Sinne einer Optimierung die Netzparameter schrittweise verbessert.

Lernverfahren

Unter Lernverfahren wollen wir eine schrittweise Anpassung des Netzwerks an die vorgelegten („zu lernenden“) Daten verstehen. Als Maß für die Annäherung soll dabei die „Methode der kleinsten Quadrate“ dienen. Es werden jeweils alle Abweichungen berücksichtigt:

$$RMS = \sqrt{\sum_{i=0}^{m-1} \sum_{j=0}^{k-1} (O_{i,n-k+j} - d_{i,j+l})^2}$$

mit *m* zu lernenden Datensätzen *d*, die jeweils *l* Eingabewerte und *k* Ausgabewerte aufweisen, das Ganze für ein Netz mit maximal *n* Elementen (die letzten *k* sind Ausgabeelemente).

In zufälliger Weise werden nun dem Netz Elemente und Verbindungen hinzugefügt oder entnommen, sowie die Gewichte von Verbindungen und Parameter der Elemente so lange verändert, bis die gewünschte Lerngenauigkeit (z.B. 5% des anfänglichen RMS) erreicht ist oder eine vorgegebene Lernzeit (z.B. 12 Stunden) überschritten wird.

Um ein zügiges „Lernen“ zu fördern, wird versucht, bereits durchgeführte positive Schritte (Verringerung des RMS) ab und an zu wiederholen. Man könnte also von einem erfolgsorientierten „Monte-Carlo“ Verfahren sprechen.

Welche Eigenschaft des Netzwerks gerade zum Ziel einer Anpassung werden soll, bleibt ebenfalls weitestgehend dem Zufall überlassen. Einige systematische „Besuchsstrategien“ erweisen sich aber dennoch als vorteilhaft und werden zusätzlich eingebracht.

Der Zufall

Nachdem offensichtlich alles irgendwie vom Zufall gesteuert werden soll, ist diesem besondere Aufmerksamkeit zu schenken. Denn im Rechner gibt es keinen wirklichen Zufall. Man setzt vielmehr auf sogenannte Pseudo-Zufallszahlen-Generatoren, die nach einem mathematischen Verfahren und dem gezielt provozierten numerischen Überlauf, sogenannte Pseudo-Zufallszahlen produzieren.

Und hier ist Vorsicht geboten !

Ausgehend von einem bestimmten Startwert wird jeweils dieselbe Zahlenfolge durchlaufen. Und es ist keineswegs sichergestellt, dass alle möglichen Zahlen der vorgegebenen maximalen Zahlenlänge durchlaufen werden. Zwar erfüllen praktisch alle RNGs (Random Number Generators) die Forderung nach globaler Gleichverteilung (beispielsweise zwischen 1 und n, was auch schon die blosse Folge 1,2,3,...,n-1,n tut), jedoch nicht ohne weiteres die Forderung nach maximaler Periodenlänge oder partieller Gleichverteilung.

Betrachtet man die „einfachen“ RNGs gängiger Programmiersysteme, so werden oft gerade mal 16 Bit für die Zahlenlänge eingesetzt, womit sich theoretisch eine Reihe von maximal 65536 Pseudo-Zufallszahlen erzeugen lässt. Es ist wohlgemerkt eine einzige Reihe, die abhängig vom Startwert an einer bestimmten Stelle begonnen wird. Und dass tatsächlich erst nach 65536 Werten sich der Startwert wieder einstellt, ist der ideale, aber fast immer nicht der reale Fall.

Benötigt man mehr Zufallszahlen als die reale Periodenlänge hergibt, so bringen all diese „Zufallsgeneratoren“ letztlich nur systematische Effekte ein, die alles andere als erwünscht sind und das ganze Simulationssystem versagen lassen. In der Tat konnte beispielsweise der beim Borland Builder mitgelieferte RNG nicht die Anforderungen des Sinus-Netzwerk-Simulators erfüllen.

Zum Einsatz kam deshalb ein eigens entworfener Zufallszahlengenerator mit 48 Bit Wortlänge (Anregung durch einen Artikel aus BYTE 3/87, S. 127-128).

Simulator

Mangels biologischer Hardware wird das vorgestellte Konzept auf einem herkömmlichen Rechner realisiert. Das dazu notwendige Programmsystem ist der (Sinus-Netzwerk-) Simulator und wird nun näher betrachtet.

Aufbau

Zum einen ist ein „Arbeitskern“ vorgesehen und zum anderen eine graphische Bediener-Oberfläche. Letztere ist zwangsläufig objektorientiert (Borland Builder) gestaltet, während das Kernprogramm aus historischen Gründen klassisch strukturiert programmiert ist.

Programm

Das Netzwerk enthält Neuronen (Sinus-Elemente), die wiederum aus Synapsen (den Inputs) und einem Axon mit den Anschlüssen an andere Neuronen (den Outputs) bestehen.

Maximal n Elemente werden über ein Pointer-Feld zugelassen. Das hat den Vorteil, dass jedes später tatsächlich erzeugte Element über einen Index angesprochen werden kann und dadurch eine Ordnung festgelegt ist (bspw. darf bei einem reinen Feed-Forward Netz Element [i] nur dann mit Element [k] verknüpft werden, falls i<k).

Die Datenstruktur für das Netz lautet nun:

```
typedef struct neuronetz
{
   char name [80];                          // Bezeichnung des Netzes
   int anz_neuronen;                        // max. Anzahl von Elementen
   int ins;                                 // Anzahl der Eingabe-Elemente
   int outs;                                // Anzahl der Ausgabe-Elemente
   double *minima;                          // Minima der Eingabe-Werte
   double *breite;                          // Umfang der Eingabe-Werte
   double lf_g;                             // Steuerparameter für Gewichte
   double lf_a;                             // Steuerparameter für Aktivitäten
   int anz_transfers;                       // Anzahl der Übertragungsfunktionen
   void *transfers [ANZ_TRANSFERS];         // die Übertragungsfunktionen
   NEURON *netz;                            // Zeiger auf Liste der Elemente
}
NEURONETZ;
```

mit den Neuronen (Elementen)

```
typedef struct neuron
{
  double activity;                          // Aktivität des Neurons (Output)
  double amplitude;                         // Skalierungsparameter für Ü-Fkt.
  double lastAmplitude;                     // vorhergehende Aktivität
  double frequency;                         // Frequenzparameter r der Ü-Fkt
  double lastFrequency;                     // vorhergehende Frequenz
  double phase;                             // Phasenparameter v der Ü-Fkt.
  double lastPhase;                         // vorgehende Phase
  double (* tf) (double x, double r, double v);      // Ü-Fkt
  int anz_in;                               // Anzahl der Synapsen
  SYNAPSE *in;                              // Liste der Synapsen
  int anz_out;                              // Anzahl der Neuronen am Axon
  AXON *out;                                // Liste der Neuronen am Axon
}
NEURON;
```

An jeder Synapse hat ein Neuron „angedockt“, das das jeweils aktuelle Neuron mit Signalen versorgt.

```
typedef struct synapse
{
   struct neuron *woher;                        // das „angedockte“ Neuron
   double lage;                                 // letzter Wert für Gewicht
   double gewicht;           // die Gewichtung des eintreffenden Signals
   struct synapse *next;                        // nächste Synapse
}
SYNAPSE;
```

Über das Axon wird die Aktivität des Neurons an angeschlossene Neuronen weitergeleitet:

```
typedef struct axon
{
   struct neuron *wohin;                  // das „angeschlossene“ Neuron
   struct axon *next;                           // der nächste Anschluss
}
AXON;
```

Frühere Werte werden deshalb zwischengespeichert, um eine erfoglreiche Variation festzuhalten und beim nächsten Test dieses Parameters erneut zu versuchen. Ansonsten verbergen sich keine Besonderheiten hinter diesem datentechnischen Ansatz.

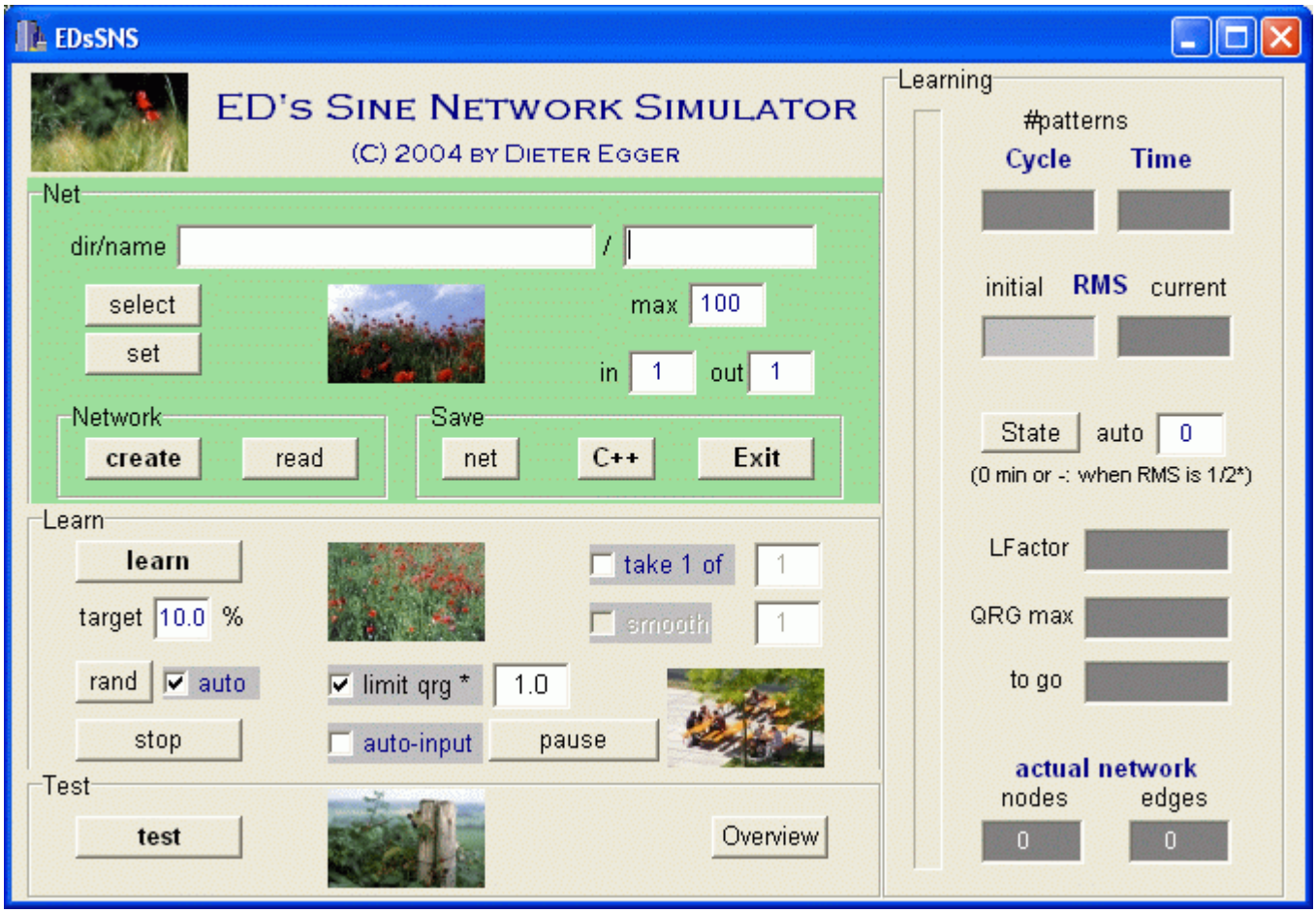
Der wichtigste Vorgang besteht aus dem Lernen der angelegten Muster, die aus Input- und Outputwerten zusammengesetzt sind. Dazu wird in der Tat ein Neuron nach dem anderen und in diesem Neuron ein Parameter nach dem anderen getestet, d.h. eine kleine Variation vorgenommen. Welches Neuron und welcher Parameter tatsächlich an der Reihe ist, entscheidet der Zufall, der auch die Stärke und die Richtung der Variation bestimmt.

Am Anfang gibt es nur die Eingabe- und die Ausgabe-Elemente ohne Verbindungen. Während des Lernens werden die Neuronen und die Verbindungen nach dem Zufallsprinzip hinzugefügt oder wieder entfernt. Falls der Gesamtfehler geringer ausfällt, bleibt die Massnahme bestehen, ansonsten wird sie zurückgenommen.

Und das ist Alles ! Mehr passiert nicht. Das System organisiert sich quasi selbst. Und zwar so, dass die Vorgaben (das was zu lernen ist) optimal erfüllt werden. Das eigentlich Verwunderliche an neuronalen Netzen ist gerade diese Fähigkeit zur Selbstorganisation.

Graphische Oberfläche

Zur bequemen Steuerung des Simulators bedient man eine einfache graphische Oberfläche. Die Bildchen sind Shortcuts für „Netz laden“ (links oben), „Netz kreieren“, „Netz lernen“ und „Netz testen“ (Mitte, von oben nach unten), sowie „Lernen Pause“ (rechts unten). Sie erfüllen hauptsächlich den Zweck der optischen Auflockerung. Sobald man mit der Maus „darüberfährt“ erscheint ein kurzer erklärender Text.



Wie schon in der Einführung erwähnt, wird nicht ein neuronales Netz simuliert, sondern ein Netzwerk aus einfachen (Sinus-) Elementen.

Ein bestehendes Netzwerk kann aus einer Datei gelesen werden, genauso wie ein vorhandenes Netzwerk in eine Datei abgespeichert werden kann. Liegt noch kein abgespeichertes Netzwerk vor, legt man fest, wieviele Elemente maximal im Netz sein dürfen, wieviele davon für die Eingabe und wieviele für die Ausgabe zuständig sein sollen. Dann vergibt man einen Namen und klickt auf „create“ und kann anschliessend mit dem Lernen beginnen.

Am besten beginnt man die Bearbeitung eines „neuen Problems“ mit wenigen Elementen und erlaubt erst bei weiteren Versuchen mehr, denn viele Elemente bedeuten auch viele Freiheitsgrade und das wiederum heisst, dass die Werte einzeln gelernt werden können und damit keine funktionalen Abhängigkeiten der Daten erkannt werden müssen.

Die Lernmuster werden in der Datei mit dem Namen des Netzes und der Endung „.lrn“ erwartet. Pro Zeile stehen zuerst die Eingabe- und dann die Ausgabewerte Das Netz selbst bekommt die Endung „.net“ und die Testdaten für ein trainiertes Netz sollten in der Datei mit der Endung „.tst“ stehen. Alle Dateien sind ASCII-Dateien.

Das Lernziel ist mit 10% des anfänglichen RMS voreingestellt, die Initialisierung des RNG's erfolgt automatisch (mit der Rechnerzeit) und der Frequenzparameter der Elemente des Netzes ist auf die maximale Eingabefrequenz der Lernmuster limitiert. Letzteres kann hilfreich sein, um Netze zu finden, die frei von „Aliasing-Effekten“ sind.

„Auto-Input“ wäre dann abzuhaken, wenn pro Lernmuster pro forma nur ein einziger laufender Inputwert vorhanden ist, dieser Wert aber nicht explizit in den Mustern auftaucht. Er wird dann automatisch erzeugt. Die angelegten Lernmustern können auf Wunsch ausgedünnt werden (take 1 of m), d.h. sie werden in Gruppen von m (m ungerade) Mustern aufgeteilt, von denen entweder das mittlere Muster oder das aus <smooth> Mustern gebildete Mittel genommen wird (<smooth> kleiner oder gleich m).

Im rechten Fensterbereich lässt sich der Lernvorgang mitverfolgen. Er wird automatisch abgeschlossen, wenn das Lernziel erreicht ist oder manuell durch einen Mausklick auf den Stop-Button. Dann kann beispielsweise eine C++ Datei erzeugt werden, die das Netzwerk als C-Funktion darstellt. Dieser C-Code kann beliebig in anderen C-Programmen weiterverwendet werden. Mit geringem Aufwand lässt sich die C-Funktion auch javatauglich machen oder etwa nach Fortran oder Basic übertragen.

Obwohl der Lernprozess in einem eigenen Prozess-Thread abläuft, ist er doch so rechenintensiv, dass er womöglich andere Programme auf dem Rechner beeinträchtigt. Zur Abhilfe kann auf den „Pause“-Button gedrückt werden, der den Lernprozess anhält, wenn kurzfristig andere Programme bevorzugt abgearbeitet werden sollen. Nach getaner Arbeit lässt sich der Lernprozess schadlos fortsetzen.

Sobald ein Netzwerk „ausreichend“ trainiert ist, kann es Testmuster verarbeiten. Mit einem Mausklick auf die „Overview“-Schaltfläche wird nach dem Lernen für alle Lernmuster und für jedes Neuron der jeweilige Neuron-Output in eine Datei mit der Endung „.ovw“ geschrieben.

Beispiele

Anhand von zwei Beispielen aus dem Reich der „Geodäsie und Geoinformation“ soll die Leistungsfähigkeit des Sinus-Netzwerks aufgezeigt werden. Es werden generell jeweils alle vorhandenen Daten als Ganzes zum Lernen vorgelegt. Ist das Lernziel erreicht, werden die Testdaten „eingespeist“, die dem Netzwerk zum einen Zwischenwerte (im Sinne einer Interpolation) und zum anderen Aussenwerte (im Sinne einer Extrapolation) abfordern. Fast alle Lernversuche verlaufen erfolgreich, was die zu lernenden Werte anbelangt. Erst bei den Zwischen- und vor allem bei den Aussenwerten scheiden sich die Netze. Daher sind für ein und dieselbe Problemstellung stets mehrere Versuche durchzuführen. Je stochastischer die Daten, desto groesser die Gefahr des Aliasing und der dadurch mangelnden Brauchbarkeit für einen weitergehenden Einsatz. Zwar werden die vorgelegten Werte sehr schnell und genau gelernt, jedoch mit einer sehr hohen Frequenz, so dass die Werte zwischen zwei Stuetzpunkten regelrecht oszillieren. Um diesen Effekt wenigstens einzudämmen wurde „limit qrg“ in der graphischen Steuerung eingeführt.

Erdrotation

15402 Lernmuster mit je einem Zeitpunkt (MJD 37665 bis MJD 53066, entsprechend Anfang 1962 bis März 2004 im Tagesabstand) und drei Kennwerten (x, y, LOD) dienten als Lernvorlage. Das bedeutete für das Netzwerk ein gesamtes Lernpensum von 61608 Werten, das innerhalb von 18 Stunden zufriedenstellend bewältigt worden ist (AMD64 3500+). Die Netzabfrage erfolgte im Tagesabstand von MJD 20000 bis MJD 70000, also von August 1913 bis etwa Mitte Juli 2050).

Die x- und die y-Koordinate des Erdrotationspols, bezogen auf den mittleren Pol am Nordpol, verhalten sich relativ „gutmütig“, nicht jedoch die Exzess-Tageslänge LOD, die starken stochastischen Schwankungen im Millisekundenbereich unterliegt. Dennoch hat es sich als vorteilhaft erwiesen alle drei Werte auf einmal zu lernen.

Auszug der zu lernenden Daten („eop.lrn“):

```
37665   -0.012700   0.213000   0.0017230
37666   -0.015900   0.214100   0.0016690
37667   -0.019000   0.215200   0.0015820
37668   -0.021999   0.216301   0.0014960
37669   -0.024799   0.217301   0.0014160
...
53062   -0.103975   0.233441   0.0004160
53063   -0.105958   0.235866   0.0002460
53064   -0.108304   0.237969   0.0000731
53065   -0.110488   0.239810  -0.0001210
53066   -0.112337   0.241966  -0.0001287
```

und die gelernten Werte:

```
 3.766500e+04   -6.465345e-02    2.291913e-01    1.388461e-03
 3.766600e+04   -6.410331e-02    2.293418e-01    1.381317e-03
 3.766700e+04   -6.354063e-02    2.294695e-01    1.374438e-03
 3.766800e+04   -6.296691e-02    2.295811e-01    1.367826e-03
 3.766900e+04   -6.238410e-02    2.296833e-01    1.361479e-03
...
 5.306200e+04   -9.673296e-02    2.227969e-01    6.660730e-04
 5.306300e+04   -9.804578e-02    2.247121e-01    6.906290e-04
 5.306400e+04   -9.932179e-02    2.266308e-01    7.162860e-04
 5.306500e+04   -1.005526e-01    2.285596e-01    7.424362e-04
 5.306600e+04   -1.017261e-01    2.305099e-01    7.682004e-04
```

die im einzelnen zwar deutliche Abweichungen zeigen, jedoch im Gesamt-Überblick eine erstaunliche Übereinstimmung signalisieren, wie die folgenden Abbildungen zeigen.

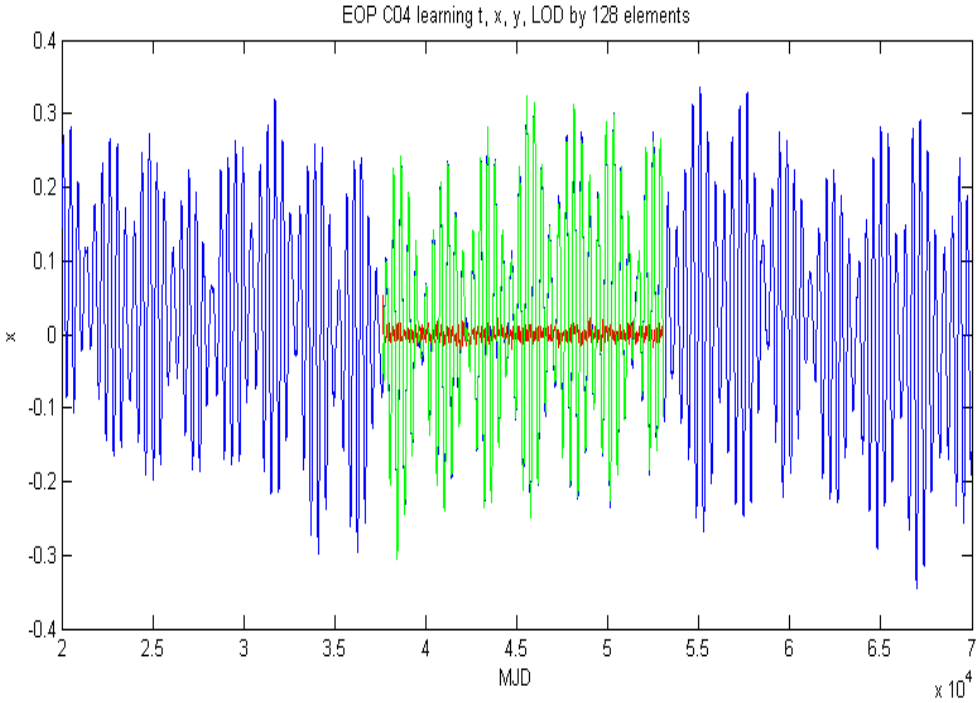


Abbildung 4: Die x-Koordinate der Rotationsachse der Erde in Bogensekunden. Der grüne Bereich in der Mitte spiegelt die Original-Werte wider, die rote Linie kennzeichnet die Lern-Abweichungen von den Originalwerten. Links und rechts finden sich in blau die extrapolierten Werte wieder. Im mittleren Teil werden die Netzdaten nahezu vollständig von den Originaldaten überdeckt, was für einen sehr guten Lernerfolg spricht.

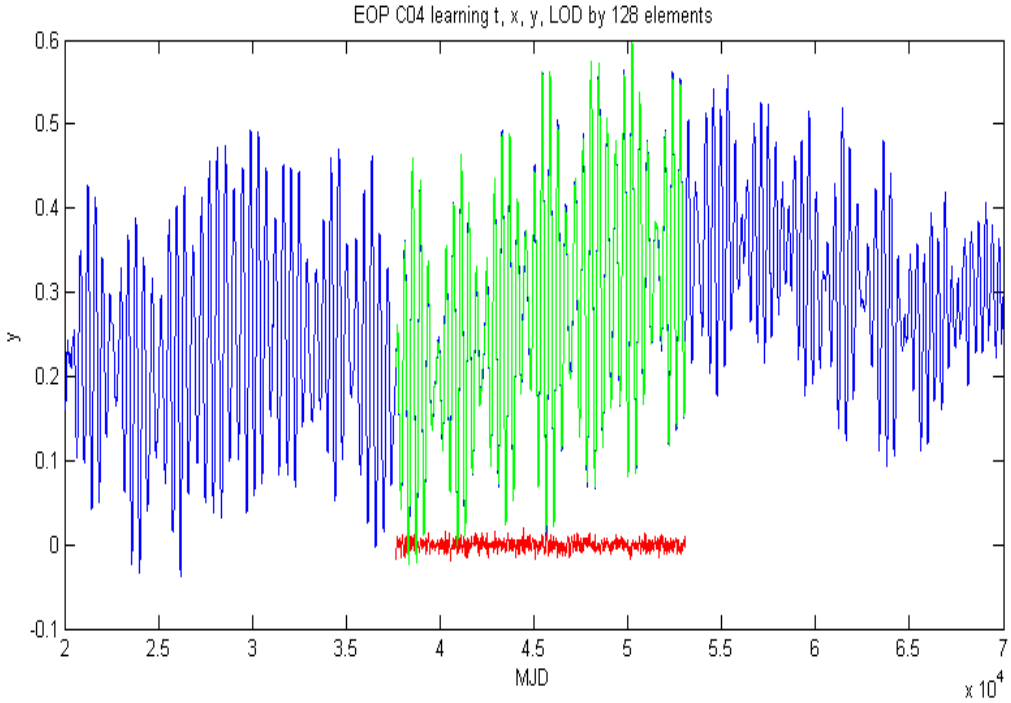


Abbildung 5: Die y-Koordinate der Erdrotationsachse (vgl. Erklärung der vorhergehenden Abb.)

Als offensichtlich schwierig gestaltet sich das Mit-Lernen der stochastisch behafteten LOD-Datenmenge:

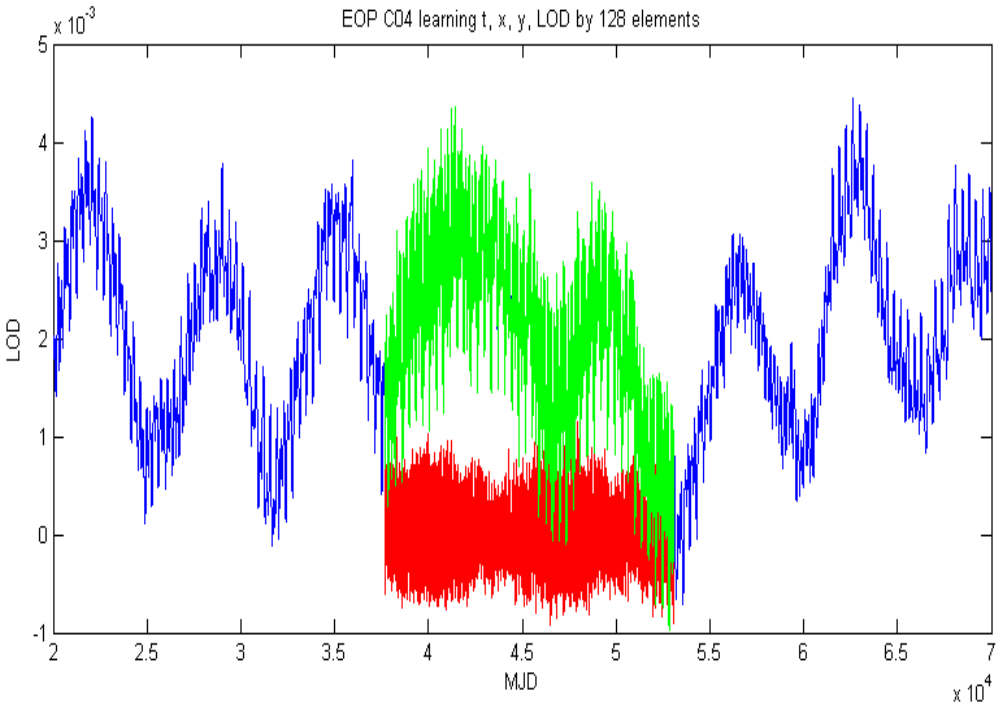


Abbildung 6: Die Variation LOD der Tageslänge in Sekunden (vgl. Erklärung bei Abb. 1). Wegen der starken Oszillationen verbleiben grössere Residuen als bei der x- und y-Polkoordinate. Im Netzwerk findet bereits eine Glättung der Werte statt, da die Anzahl der Elemente (Neuronen) viel zu klein ist, um jeden Wert einzeln für sich zu lernen.

Zur besseren Veranschaulichung wird aus den vorgehenden Abbildungen jeweils der Zeitraum von etwa 1995 bis 2010 (MJD 50000 bis 55000) herausgezoomt:

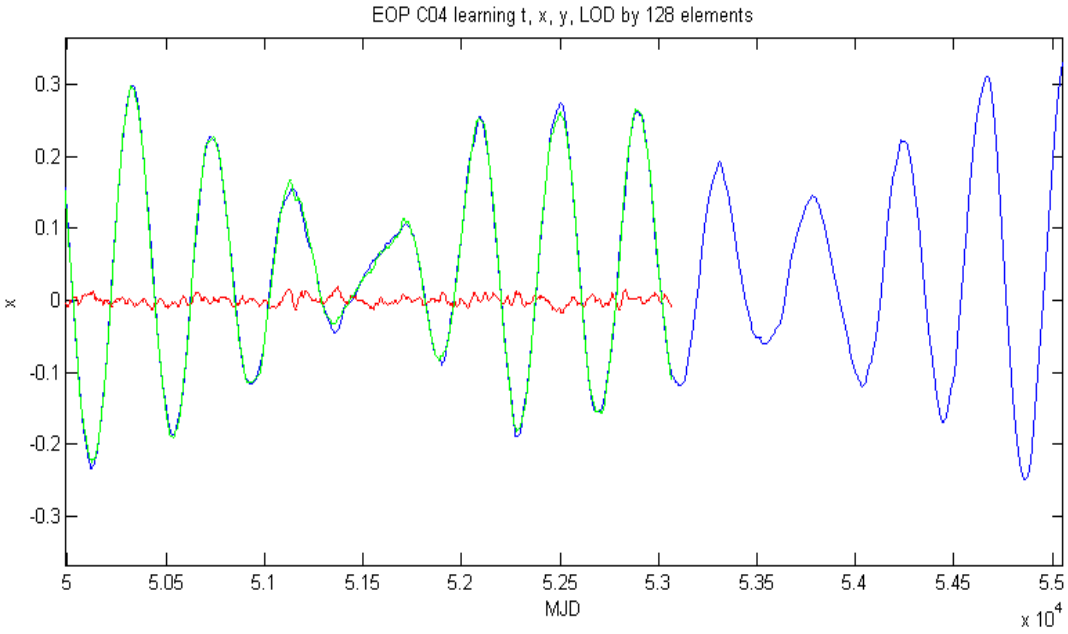


Abbildung 7: Veranschaulichung des x-Pols im Zeitraum 1995 bis 2010. Die Lernabweichungen (rote Kurve) sind sehr gering. Plausibel erscheint die Vorhersage von 2004 bis 2010, wenn auch keinerlei Aussage über ihre Zuverlässigkeit getroffen werden kann.

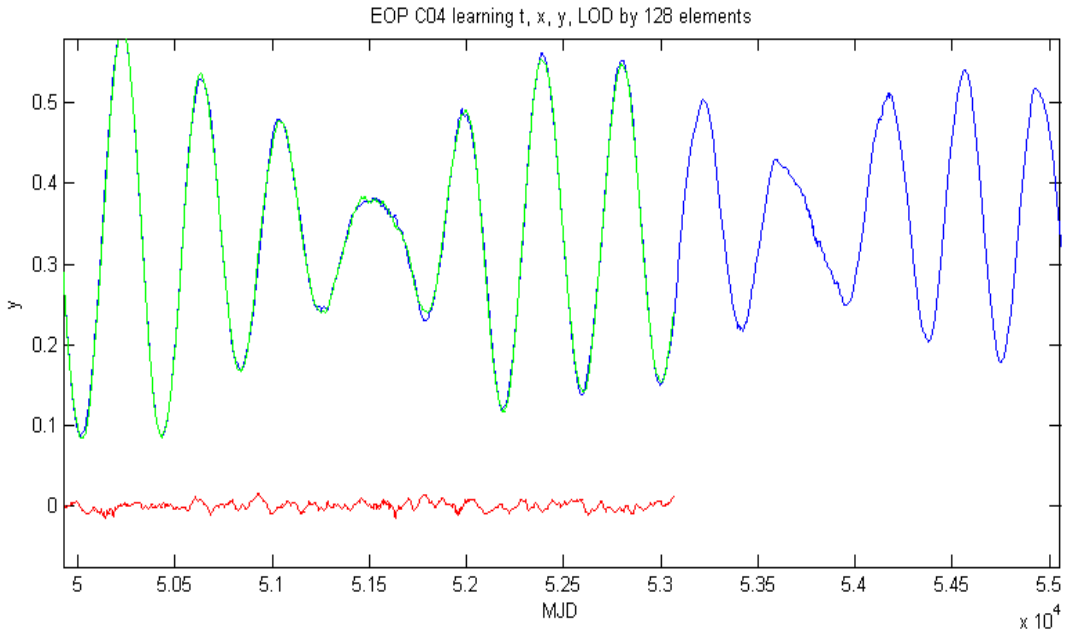


Abbildung 8: Die y-Koordinate der Erdrotationsachse wurde ebenfalls sehr gut „gelernt“.

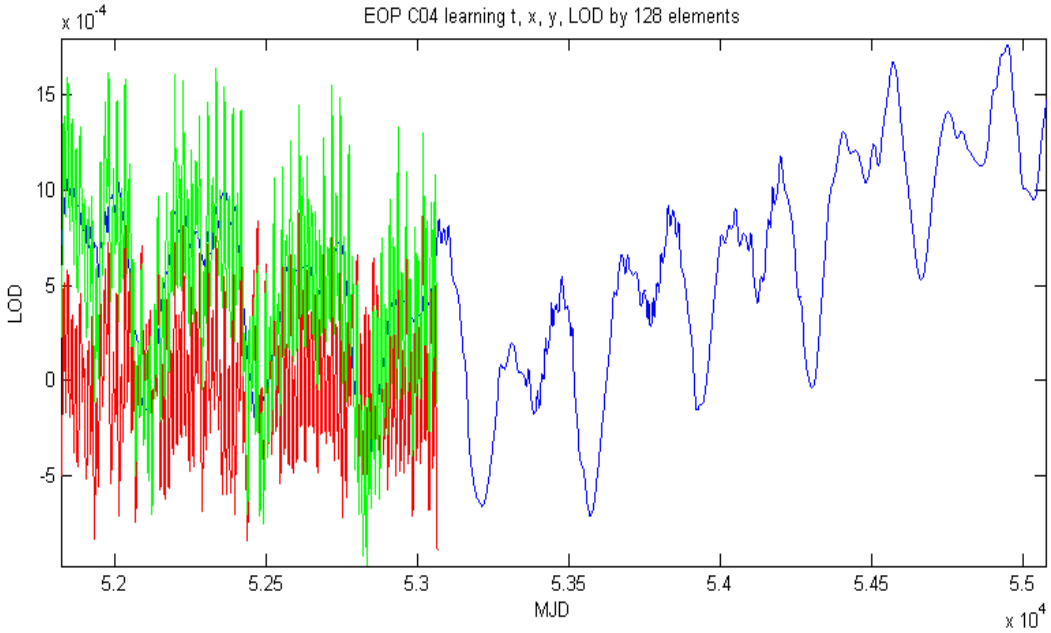


Abbildung 9: Variation der Tageslänge in Sekunden während der Jahre 1995 bis 2010. Die grüne Kurve gibt die Originalwerte wieder, die rote Kurve zeigt die Lernabweichungen. Vom Netzwerk selbst stammt die blaue Kurve, die im extrapolierten Bereich ab 2004 recht gut die jährlichen und halbjährlichen Schwankungen erfasst und die schnellen Oszillationen schön ausfiltert.

Wie schon erwähnt, erhält man nicht unbedingt schon beim ersten Lernversuch des Sinus-Netzwerks den gewünschten Erfolg. Zwar werden die Originalwerte fast immer hervorragend gelernt, jedoch fallen die Extrapolationen doch recht unterschiedlich aus. Vor allem, wenn es sich um Daten mit hohem stochastischem Anteil handelt, wie es bei der Tageslängenvariation der Fall ist. Daher wird man günstigerweise mehrere Prognosen erstellen und dann später die Zutreffendste auswählen.

Wiederverwendbarkeit

Dem Sinus-Netzwerk zum Lernen der Erdrotationsparameter wurden zunächst maximal 128 Elemente („Neuronen“) zugestanden, von denen tatsächlich dann 125 zum Einsatz kamen, die über 906 Verbindungen miteinander kommunizieren. Jede Verbindung trägt ein bestimmtes Gewicht und jedes Element 3 Übertragungsparameter.

Damit die gewonnene Lösung möglichst allgemein genutzt werden kann, steht eine C++ Datei bereit, die auf Knopfdruck vom trainierten Netz ausgegeben wurde. Sie enthält ein sehr einfaches C-Programm zum Testen der Lösung, die selbst als C-Prozedur formuliert ist. Nach dem Eintippen der Eingabewerte erfolgt die Ausgabe der Netzantwort.

Hier ein Auszug (vollständige Datei unter

http://tau.fesg.tu-muenchen.de/~dieter/enn/netzwerk/netzwerk.php

abrufbar; dort finden sich auch die Lerndaten „eop.lrn“, die gelernten Daten „eop.txt“, die Lernresiduen „eop.res“, das Netz „eop.net“, die Testdaten „eop.tst“ und das zugehörige Testergebnis „eop.aus“):

```
#include <conio.h>
#include <stdio.h>
#include <math.h>

// neuroValue erwartet:
// 1 Eingabewert(e) in xein
// 3 Ausgabewert(e) in xaus

double x [128];
double t, tq;
const int ins = 1;
const int outs = 3;

void neuroValue (double *xein, double *xaus) {
   x [0] = (xein [0]-(  3.7665000000000000e+04))/  1.5401000000000000e+04 * 2.0
- 1.0;
   x [0] = (  1.0993628215672893e+00)*(x [0] * (  1.0645965974269969e+00) +
(  1.6413897432825789e-01));

   t = ( -3.0421612595275382e-02 * x [0]);
   x [1] =   1.9227786210652131e-02 * sin  (  5.3753777623301642e+02 * t +
(  8.5694731648449984e-02));

   t = ( -1.1224671391306652e-01 * x [1]) + (  1.0179778964486746e-02 * x [0]);
   x [2] =   6.7652076351804855e-02 * sin  (  5.4338217283521681e+02 * t +
(  8.5009816654483497e-02));

   t = (  6.9489569560231709e-02 * x [1]) + ( -2.0904109687877302e-03 * x [0]);
   x [3] =  -4.4953618237023638e-02 * sin  ( -3.2657339637801709e+02 * t +
( -1.8725522911285122e+00));

   t = ( -7.5704474987647499e-02 * x [0]) + ( -1.0466353138237854e-01 * x [1]);
   x [4] =   5.1578359553833684e-02 * sin  ( -4.1261439118936153e+01 * t +
(  1.1473921404249239e+00));

   t = ( -1.4962111778477050e-02 * x [0]) + (  3.0589538546974228e-03 * x [2])
+ ( -4.6772662255706136e-03 * x [3]);
   x [5] =   2.1100298487959947e-02 * sin  ( -4.1662479739943012e+02 * t +
(  4.4396991550705706e-01));
...
...
...
   t = (  5.0711189782244481e-01 * x [27]) + (  3.2996514082098582e-01 * x
[96]) + (  3.0767164575332617e-01 * x [43])
```

```
      + ( -3.3564245330239456e-01 * x [15]) + ( -5.3825203066460575e-02 * x
[100]) + (  3.8803100016066938e-01 * x [4]) + (  4.8496809876668678e-01 * x
[83])
      + (  5.3004561333074775e-02 * x [30]) + ( -1.3989215907738198e-01 * x
[108]) + (  3.4760132667210758e-01 * x [68]) + (  4.3884966376310908e-01 * x
[48])
      + (  4.1386658058006209e-01 * x [81]) + ( -1.9857153783174375e-01 * x
[47]) + (  4.8717188023534592e-02 * x [107]) + ( -3.2564116073644910e-04 * x
[125])
      + (  2.2040703847539478e-02 * x [102]) + ( -2.1639564558165500e-01 * x
[36]) + (  3.2336620666919475e-03 * x [24]) + ( -1.7102949952024250e-04 * x
[45])
      + ( -1.4337000755714888e-01 * x [58]) + (  1.3377412300483622e-02 * x
[113]) + (  7.9951634069334102e-02 * x [64]) + (  2.7554872494340243e-03 * x
[116])
      + ( -9.0235441671039890e-02 * x [115]) + (  4.3547850074666085e-02 * x
[93]) + ( -8.3151592157436980e-03 * x [104]) + (  9.1220312596265521e-02 * x
[119])
      + (  5.9813708046085714e-03 * x [56]) + (  5.5783494935927096e-02 * x
[72]) + (  2.9221320691071575e-03 * x [29]) + (  1.9703223629050218e-01 * x
[98])
      + ( -2.5136189103922100e-03 * x [88]) + ( -6.8220976507771976e-05 * x
[38]) + (  5.1589784923038538e-03 * x [73]) + (  1.4162795175701114e-02 * x
[76])
      + (  6.6241013146827496e-02 * x [111]) + ( -1.1626000696357118e-01 * x
[69]) + ( -2.8577757330073880e-02 * x [8]) + ( -1.5897543355408800e-02 * x
[51])
      + (  4.8008641470723779e-03 * x [32]) + (  3.6106187132179482e-06 * x
[75]) + ( -6.0981201302636755e-02 * x [110]) + (  6.3637639427653728e-02 * x
[124])
      + (  1.4887856690898612e-02 * x [46]);
   x [127] =   1.6073616461654296e+00 *  (  1.1347188112807487e+00 * t +
(  6.3920361376561791e-02));
   xaus [0] = (x [125] + 1.0)/2.0 *   6.3061000000000000e-01 +
( -3.0619000000000002e-01);
   xaus [1] = (x [126] + 1.0)/2.0 *   6.1973000000000000e-01 +
( -2.2919999999999999e-02);
   xaus [2] = (x [127] + 1.0)/2.0 *   5.3218000000000007e-03 +
( -9.6679999999999997e-04);
}

main () {
   double ein [ins];
   double aus [outs];
   printf ("Bitte 1 Eingabewert(e) --> ");
   for (int i=0; i<ins; i++) scanf ("%lf", ein+i);
   neuroValue (ein, aus);
   for (int i=0; i<outs; i++) printf ("%24.16le\n", aus[i]);
   while (!kbhit());
   return 0;
}
```

Die Funktion kann herausgelöst und in andere Programme eingesetzt werden, wenn man prädizierte oder auch nur interpolierte Werte für die Erdrotation benötigt. Als Eingabe dient die Zeit als MJD, die Ausgabe erfolgt in Bogensekunden für die x- und y-Koordinate des Rotationspols und in Sekunden für die Tageslängenveränderung.

Sonnenaktivität

Als besonders „hartnäckig“ erwiesen sich die relativen Sonnenfleckenzahlen zum Lernen durch ein Sinus-Netzwerk. Obwohl fast alle Netze sich die vorgelegten Werte zügig einverleibten, unterschieden sie sich doch deutlich in den Prognosen.

Die Daten

Seit etwa 1750 liegen Sonnenbeobachtungen vor. Die relativen Sonnenfleckenzahlen geben Auskunft über die jeweilige Sonnenaktivität, die sich so etwa alle 11 Jahre in ihrem Rhythmus wiederholt. Das heisst, sie schwankt zwischen sehr geringer Aktivität (wenige oder gar keine Sonnenflecken) und ziemlich heftiger Aktivität (um die 200 Sonnenflecken und darüber). In Zeiten hoher Aktivität weht ein stärkerer Sonnenwind, der die hohen Atmosphärenschichten stärker ionisiert und dadurch beispielsweise die Kurzwellenausbreitung rund um die Erde begünstigt. In Zeiten niedriger Sonnenaktivität ist der weltweite Funkverkehr auf Kurzwelle dagegen stark eingeschränkt.

Als Lernvorlage dienten die monatlichen Daten von Anfang 1749 bis April 2004. Hier ein kleiner Auszug der 3064 Wertepaare:

```
     1749.042    58.0
     1749.125    62.6
     1749.208    70.0
     1749.292    55.7
     1749.375    85.0
     1749.458    83.5
     1749.542    94.8
     1749.625    66.3
     1749.708    75.9
     1749.792    75.5
     1749.875   158.6
     1749.958    85.2
...
...
     2003.375    55.2
     2003.458    77.4
     2003.542    85.0
     2003.625    72.7
     2003.708    48.8
     2003.792    65.6
     2003.875    67.2
     2003.958    47.0
     2004.042    37.2
     2004.125    45.9
     2004.208    48.9
     2004.292    39.3
```

Das Netzwerk

Die maximal zugestandenen 128 Sinus-Elemente wurden alle verwendet und mit 1499 Verbindungen verknüpft. Die anfängliche RMS-Abweichung von knapp 39 (normiert) sank innerhalb von 4 Lern-Minuten bereits auf 9 um dann erst nach knapp 6 Stunden auf 4.8 und weiteren knapp 10 Stunden die 10 Prozent-Marke von 3.9 nach unten zu durchstossen. Dieses asymptotische Lernverhalten ist durchweg typisch für (neuronale) Netzwerke und zeigt an, dass eine weitere Verbesserung immer mehr Zeit in Anspruch nehmen würde. Sinnvollerweise bricht man spätestens in diesem Stadium den Lernvorgang ab.

Betrachten wir nun die Resultate:

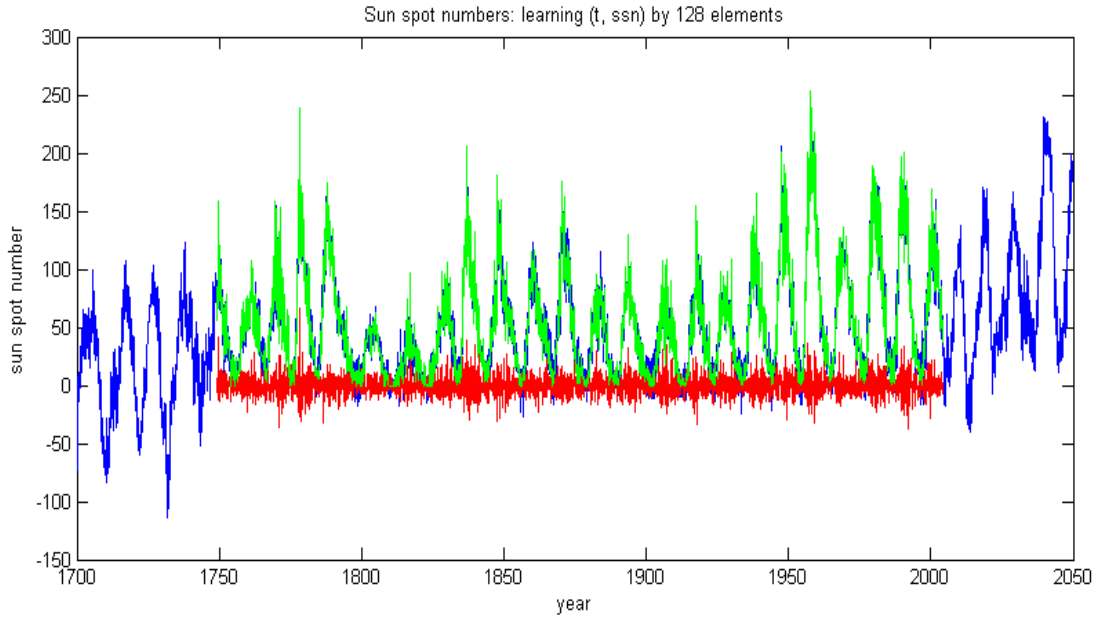


Abbildung 10: Relative Sonnenfleckenzahlen von 1700 bis 2050. Der etwa 11-jährige Zyklus ist gut zu erkennen und bleibt auch in der Zukunftsprognose des Netzwerks erhalten. Grün sind wieder die Originalwerte eingefärbt, rot die Lern-Abweichungen und blau die Netzantwort.

Da insbesondere der Zeitraum „heute +/- 20 Jahre“ von aktuellem Interesse ist, sei dieser aus der vorhergehenden Abbildung herausgezoomt.

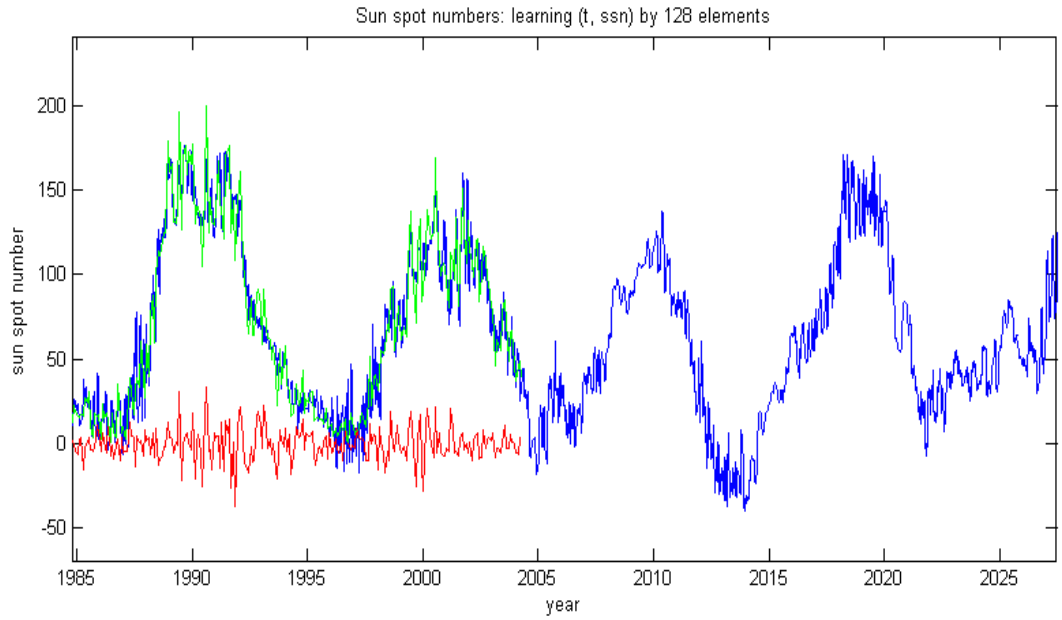


Abbildung 11: Der gezoomte Bereich von 1985 bis 2025. Der etwa 11-jährige Zyklus bleibt auch in der Zukunftsprognose des Netzwerks erhalten. Grün sind die Originalwerte eingefärbt, rot die Lern-Abweichungen und blau die Netzantwort.

Viele der getesteten Netzwerke approximierten den zu lernenden Bereich (grüne Kurven der beiden Abbildungen) recht gut, aber nur einige wenige behielten den offensichtlich vorhandenen Zyklus in der Prognosezone bei.

Wiederverwendbarkeit

Da die gesamte Netzwerkstruktur als C-Funktion formuliert werden kann, ist ein bestmöglicher Einsatz der gewonnenen Lösung möglich.

Auszug aus dem C-Testprogramm (die vollständige Datei „spot.cpp“ ist unter

http://tau.fesg.tu-muenchen.de/~dieter/enn/netzwerk/netzwerk.php

einzusehen; dort finden sich auch die anderen Dateien, die zu diesem Netzwerk gehören):

```
#include <conio.h>
#include <stdio.h>
#include <math.h>

// neuroValue erwartet:
// 1 Eingabewert(e) in xein
// 1 Ausgabewert(e) in xaus

double x [128];
double t, tq;
const int ins = 1;
const int outs = 1;

void neuroValue (double *xein, double *xaus) {
  x [0] = (xein [0]-(  1.7490419999999999e+03))/  2.5525000000000000e+02 * 2.0
- 1.0;
  x [0] = (  1.0999795892475925e+00)*(x [0] * (  1.0648104306765851e+00) +
(  9.3774388775064443e-02));

  t = ( -3.6911404855005020e-01 * x [0]);
  x [1] =   3.7371965175693483e-01 * sin  (  1.1387280393049621e+00 * t +
(  3.5729617041007460e-02));

  t = (  3.9160549456093890e-01 * x [1]) + (  4.3073159732587962e-02 * x [0]);
  x [2] =  -7.1161964744500513e-03 * sin  (  4.2616902518605406e+02 * t +
( -2.0456400425998691e-01));

  t = ( -8.5993855991758103e-04 * x [2]) + ( -6.0835048844705615e-04 * x [0]);
  x [3] =  -4.6359122804532769e-02 * sin  ( -5.6763043939849399e+02 * t +
( -3.7530937385216193e-02));
...
...
...
  t = (  3.8550215209954641e-01 * x [72]) + ( -1.0386525324729967e-01 * x
[101]) + ( -6.2424870805958799e+00 * x [41])
     + ( -4.4116869476501008e-02 * x [52]) + ( -1.0568840816840208e-01 * x
[5]) + ( -9.3180645249514277e-02 * x [65]) + (  1.4381752415043839e-01 * x
[124])
     + ( -1.2388486994167604e-01 * x [53]) + (  1.1473731246087940e-01 * x
[38]) + ( -1.5924364569878019e-01 * x [123]) + ( -1.9829409070048900e-02 * x
[89])
     + ( -2.9762210064712624e-01 * x [2]) + (  6.7285963361274098e-02 * x [6])
+ (  8.1380442879452741e-01 * x [126]) + ( -2.7915554604371354e-02 * x [59])
     + ( -4.0426621546436386e-01 * x [71]) + ( -3.6224671199601453e-02 * x
[39]) + (  1.2277135169993153e-01 * x [46]) + (  3.4148288148588418e-01 * x
[90])
     + (  6.8008718762277831e-01 * x [113]) + ( -9.5103244384729571e-02 * x
[4]) + ( -1.3205135328565540e-01 * x [108]) + (  9.4295612981932050e-02 * x
[96])
     + ( -8.7019414618644014e-02 * x [77]) + (  9.0890471254771163e-02 * x
[34]) + ( -7.5329581684579275e-02 * x [17]) + (  1.0246141227935596e-02 * x
[57])
     + (  1.7114307592405863e-02 * x [42]) + ( -5.5060826115991846e-01 * x
[67]) + ( -3.6734746500589197e+00 * x [19]) + ( -7.0607168959169455e-03 * x
[44])
```

```
      + ( -6.9227748718924023e-02 * x [14]) + (  3.0791200940411519e-03 * x
[111]) + ( -1.2482538465379011e-10 * x [102]) + ( -5.6475303925062441e-02 * x
[122])
      + (  4.1617938094700692e-02 * x [70]) + ( -4.2137913699299295e-01 * x
[119]) + (  1.0853086093206261e-01 * x [56]) + ( -4.7367311134501686e-03 * x
[93])
      + ( -1.7910667825484039e-01 * x [82]) + (  3.7958041458885268e-02 * x
[80]) + ( -2.0632546857046147e-02 * x [31]) + ( -2.8979686468077887e-02 * x
[68])
      + ( -3.1244271852602021e-02 * x [21]) + (  1.8898300267909587e-01 * x
[36]) + ( -7.1296796442277433e-02 * x [66]) + ( -3.8766777463919816e-05 * x
[18])
      + ( -1.3263232012344948e-02 * x [86]) + (  1.1677614072558115e-06 * x
[76]) + (  3.1906874268294118e-02 * x [98]) + (  3.5039456450447190e-02 * x
[115])
      + ( -2.0706379622693477e-02 * x [97]) + (  5.2723934814176169e-03 * x
[103]) + ( -3.6280321340972768e-07 * x [75]) + (  2.0794349012290912e-02 * x
[112])
      + ( -1.9497218663776125e-02 * x [106]) + ( -2.3698371025783541e-02 * x
[117]) + (  1.1208618935516832e-02 * x [83]) + ( -1.0209056421331501e-01 * x
[118])
      + (  6.7754769187831548e-02 * x [116]) + ( -1.6256011893958647e-01 * x
[120]) + (  2.3594792669970263e-02 * x [99]) + ( -3.8779040692846606e-03 * x
[100])
      + (  2.2708569359957461e-02 * x [49]) + (  1.5278989625259334e-01 * x
[81]) + ( -6.9688403361665130e-01 * x [45]) + (  1.7667999172360191e-01 * x
[110])
      + (  4.0094178457290488e-03 * x [91]) + (  1.6435062207185948e-04 * x
[125]) + (  4.8839005549630367e-02 * x [64]);
   x [127] =   4.3821555081732928e+00 *  (  1.2208663315192791e+00 * t +
( -1.3582390988948206e-01));
   xaus [0] = (x [127] + 1.0)/2.0 *   2.5380000000000001e+02 +
(  0.0000000000000000e+00);
}

main () {
   double ein [ins];
   double aus [outs];
   printf ("Bitte 1 Eingabewert(e) --> ");
   for (int i=0; i<ins; i++) scanf ("%lf", ein+i);
   neuroValue (ein, aus);
   for (int i=0; i<outs; i++) printf ("%24.16le\n", aus[i]);
   while (!kbhit());
   return 0;
}
```

Als Eingabe wird das Datum als Jahreszahl + Jahresbruchteil erwartet, also beispielsweise 2005.375 für Mitte Mai 2005. Danach steht die zugehörige Sonnenflecken-Relativzahl als Ergebnis an.

Zusammenfassung

Mit einfachen Elementen lassen sich im Verbund erstaunliche Leistungen erzielen. Fast wie von selbst wird eine vorgelegte Datenmenge „gelernt“. Aber es sind offensichtlich nicht nur die diskreten Werte allein, die gelernt werden, sondern Funktionalitäten, die diese Daten und darüberhinaus auch dazwischen und ausserhalb liegende Daten (re-)produzieren können.

Man muss dabei gar nicht einmal wissen, woher die Daten kommen, wer sie also in der „realen“ Welt erzeugt hat. Genauso wenig muss man wissen, wie die Daten im (physikalischen) Detail zustande gekommen sind. Und dennoch lassen sich bei Zeitreihen beispielsweise zukünftige Werte vorhersagen. Zunächst bleibt aber völlig im Verborgenen, wie zuverlässig solche Vorhersagen sein können. Was macht man daher am besten ? Ganz einfach, wie im richtigen Leben, andere Experten fragen, mithin also mehrere Lernansätze verfolgen, die alle zu unterschiedlichen Netzwerken führen (solange der Zufallszahlengenerator die entsprechende Vielfalt bereitstellen kann).

Jedes Netzwerk wird die vorgelegten Daten in etwa gleich gut „lernen“, sicherlich verschieden schnell, aber unterschiedliche Funktionalitäten herausfinden. Somit lauten auch die Prognosen anders. Letztlich kann natürlich nur die reale zukünftige Entwicklung darüber entscheiden, welches der Netzwerke am nächsten liegt.

In der vorliegenden Arbeit wurde gezeigt, wie sogenannte Sinus-Netzwerke, also Netzwerke mit Elementen, die die Sinus-Funktion beherrschen, für „alltägliche“ Aufgaben eingesetzt werden können.

Wenn früher stets die gesamte Netzwerk-Software erforderlich war, um ein trainiertes Netz abzufragen, wurde nun die Netz-Funktionalität auf eine C-Funktion abgebildet, die leicht in anderen Programmsystemen wiederverwendet werden kann. Somit kann auch mal schnell eine Approximationsfunktion für irgendwelche vorgelegte diskrete Werte ermittelt und genutzt werden.

Falls Sie den Sinus-Netzwerk-Simulator selbst einsetzen möchten, steht dem nichts im Wege (siehe Download-Hinweis im Vorwort). Allerdings bitte ich bei Publikationen um einen entsprechenden Vermerk.

Anhang

Zum Kennenlernen

Ein erster Versuch

Wenn Sie das Programm und die Testdaten von

http://tau.fesg.tu-muenchen.de/~dieter/enn/netzwerk/netzwerk.php

auf Ihren Rechner geladen haben, können Sie mit einem ersten Test beginnen.

Kreieren Sie ein Directory mit dem Namen „test“ und platzieren Sie dort die beiden Dateien „test.lrn“ und „test.tst“. Erstere enthält die Lernmuster, die eine einfache Zeitreihe mit 9 Wertepaaren wiedergibt:

```
0  1
1  1
2  1
3  1
4  0
5  1
6  1
7  1
8  1
```

Die Null in der Mitte wird von jeweils 4 Einsen rechts und links flankiert. Wir können die erste Spalte auch weglassen und sie später automatisch erzeugen lassen (auto-input wird dann abgehakt). Wir müssen die Input-Spalte aber immer dann angeben, wenn sie nicht bei Null beginnt und nicht in Einserschritten wächst.

„test.tst“ enthält die Testmuster, also nur die Input-Spalte(n), die entweder ausgeschrieben

```
-5
-4
-3
-2
-1
0
1
2
3
4
5
6
7
8
9
10
11
12
13
```

oder in Kurzform gehalten werden kann

```
-5
1
13
```

mit dem ersten Wert, der Schrittweite und dem letzten Wert, der zu abzufragen ist.

Dann starten Sie das Programm und klicken der Reihe nach die Bildchen von links oben und dann in der Mitte weiter nach unten an. Sie stehen für

1) Bild links oben anklicken -> Netz über File-Auswahlbox auswählen (also „test.lrn“ im Directory „test“)

2) wenn nicht schon geschehen: max. Anzahl der Elemente festlegen (z.B. 32)

3) erstes Bildchen in der Mitte oben anklicken -> neues Netz wird erzeugt

4) mittleres Bildchen in der Mitte anklicken -> Lernvorgang starten (da er weniger als eine Sekunde dauert, wird praktisch sofort der untere Testbereich markiert)

5) unterstes Bildchen in der Mitte anklicken -> Testmuster anlegen und vom Netz anfordern

Sie können vor dem Lernen noch die gewünschte Genauigkeit auf ein Prozent oder weniger setzen und die Limitierung der max. Netzfrequenz aufheben.

Um ein anderes Netz für den Test auszuprobieren, wiederholen Sie die Schritte 2) bis 5)

Sie finden in der gezippten Programmdatei auch eine Datei „show.m“, mit deren und MATLAB's Hilfe sie die Lern- und Testergebnisse visualisieren können.

Während des Lernens wird eine Protokoll-Datei erstellt. Voreinstellung ist, dass immer dann der aktuelle Status ausgegeben wird, wenn sich die RMS-Abweichung halbiert hat. Es lässt sich aber auch ein von 0 verschiedener Wert in Minuten eintragen. Ist er negativ, werden *zusätzlich* alle diese Minuten die Statusdateien (“n01“,“.r01“,“.n02“,“.r02“,...) ausgegeben, ansonsten *nur* dann. Ausserdem bewirkt ein Klick auf den Button „Status“ die sofortige Ausgabe des momentanen Netz-Zustands.

Testergebnisse

Und so könnte es aussehen (32 Elemente, 10 % Lerngenauigkeit, Abfrage in Einserschritten):

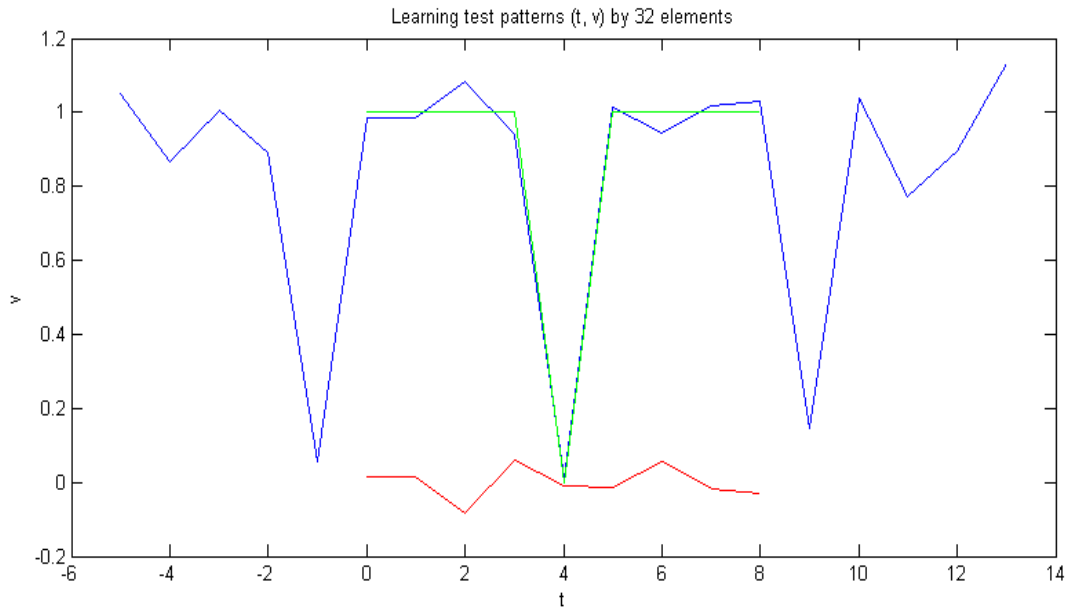


Abbildung 12: Testergebnis für ein SInus-Netz mit 32 Elementen und 10 % Lerngenauigkeit. Die Netzantwort (blaue Kurve von ganz links nach ganz rechts) nähert die zu lernenden Werte (grün) schon ganz gut an (Abweichungen: rote untere Kurve).

Unmittelbar nach dem Klick auf das Lernen-Bildchen steht auch bei 1 % geforderter Genauigkeit das Ergebnis sofort an:

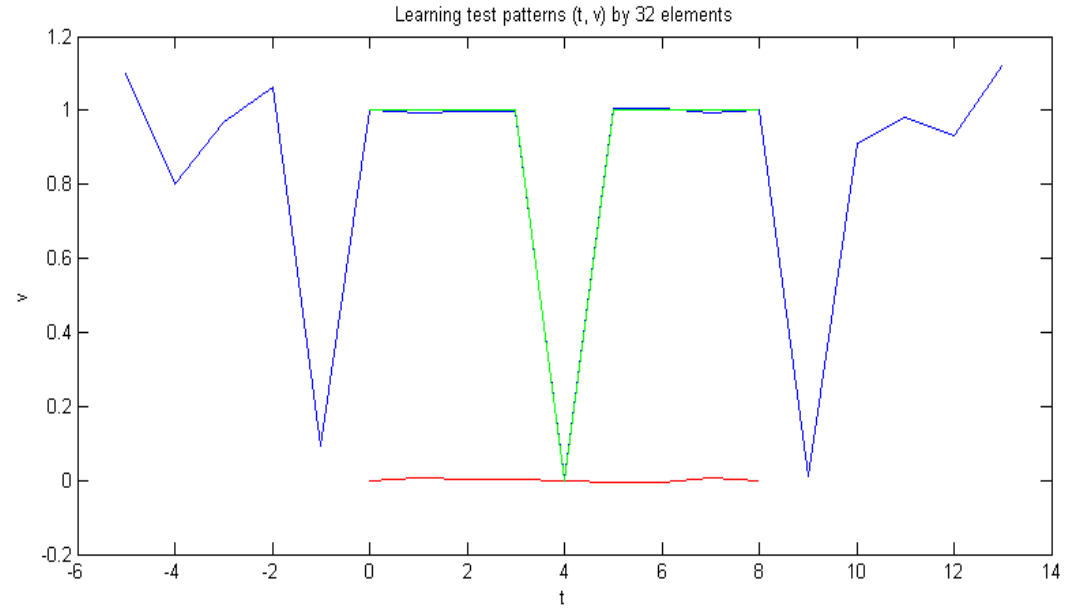


Abbildung 13: Bei einem Prozent Lerngenauigkeit wird der zu lerndende Bereich nahezu perfekt vom Netz repräsentiert. Die Extrapolationen in die Vergangenheit und in die Zukunft könnten plausibel sein, müssen aber keineswegs genau so aussehen. Abgefragt wurde das Netz genau an den Lern-Stützstellen.

Und nun folgt eine bemerkenswerte Eigenschaft des Sinus-Netzwerks (Aliasing). Frägt man nämlich auch nach den Zwischenwerten mit ¼ der Schrittweite, so offenbart sich der Sinus-Kern:

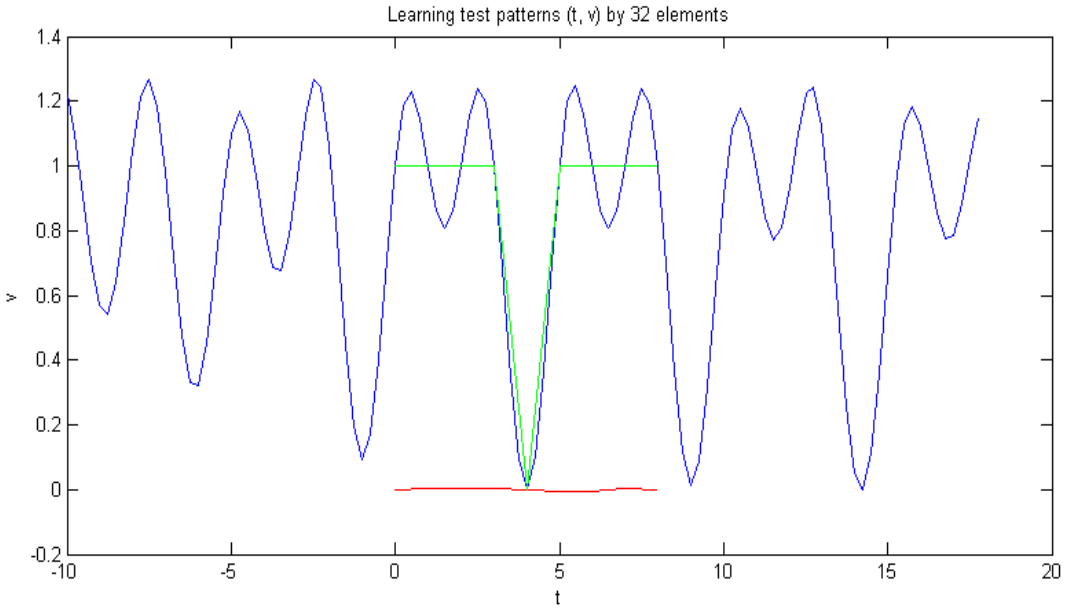


Abbildung 14: Um die Werte schnell und genau zu lernen, oszilliert das Netz einfach zwischen den zu lernenden Werten hin und her, repräsentiert diese selbst hervorragend, erfindet aber eigene Zwischenwerte.

Wegen diesem Aliasing-Effekt ist es stets ratsam, ein trainiertes Netz auch nach den Zwischenwerten zu fragen. Und wenn Oszillationen auftreten, die nicht unbedingt plausibel erscheinen bzw. real nicht vorhanden sind, empfiehlt es sich, ein neues Netz zu trainieren. Je mehr Elemente dem Netz zugestanden werden, desto „freizügiger“ wird es mit den Daten umgehen:

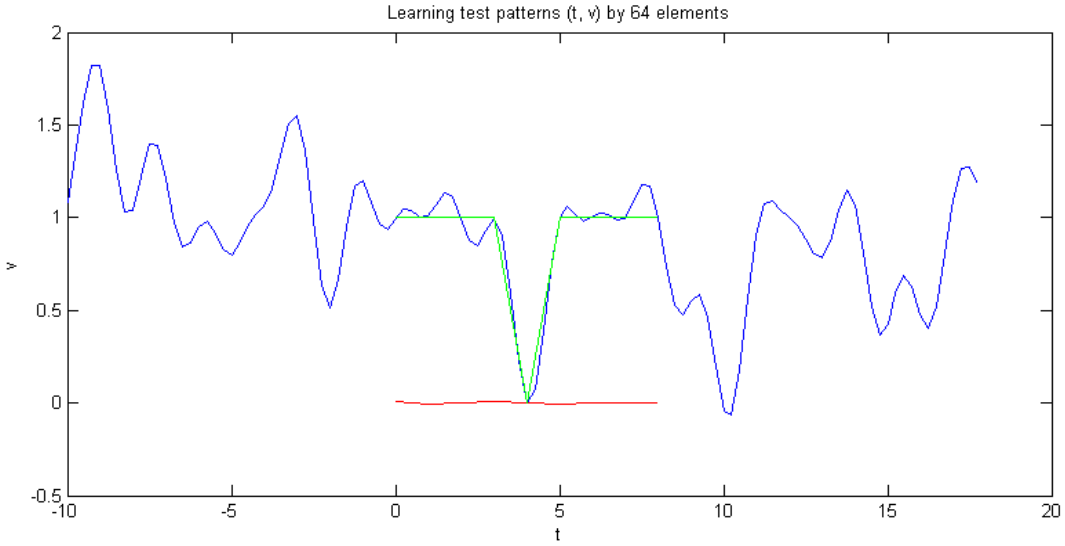


Abbildung 15: Auch wenn von den maximal 64 zur Verfügung gestellten Elementen nicht alle benötigt worden sind, sieht man doch schon den etwas lockereren Umgang mit den zu lernenden Daten. Für die Extrapolationen wird dadurch ebenfalls mehr Spielraum geschaffen.

Anmerkungen

Die Lernmuster müssen nicht gleichabständig sein, auch wenn dies bei den Beispielen und den Testmustern suggeriert wird. Bei Zeitreihen dürfen also durchaus beliebige Zeitpunkte (mit Unterbrechungen) vorgelegt werden.

Falls unerwünschte Aliasing-Effekte auftreten, trainiert man am Besten ein neues Netz mit weniger Elementen und/oder reduzierter zulässiger Netzfrequenz („limit qrg“ abhaken und Faktor kleiner als 1.0 wählen). Führt dies nicht zu glatteren Kurven oder lässt sich die erforderliche Genauigkeit beim Lernen nicht erreichen, handelt es sich womöglich um stark stochastisches Datenmaterial.

In diesem Fall sind viele Elemente erforderlich und es werden sehr viele Verbindungen (mindestens 10 mal so viel wie Elemente vorhanden sind) aufgebaut. Dann werden auch die Zwischenwerte und die extrapolierten Werte stochastische Merkmale widerspiegeln.

Literatur

Black I.: *Symbole, Synapsen und Systeme,* Die molekulare Biologie des Geistes, Spektrum Akademischer Verlag, 1993

Churchland P., Sejnowski T.: *The Computational Brain,* MIT Press Paperback, 1994

Egger D.: *Neuronales Netz prädiziert Erdrotationsparameter,* Allgemeine Vermessungsnachrichten (AVN), 11/12, 517-524, 1992

Egger D., Fröhlich H.: *Prädiktion von Erdrotationsdaten – klassisch und neuronal,* Allgemeine Vermessungsnachrichten (AVN), 10, 366-375, 1993

Rojas R.: *Theorie der neuronalen Netze,* Eine systematische Einführung, Springer-Verlag, 1993

Spies M.: *Unsicheres Wissen,* Wahrscheinlichkeit, Fuzzy-Logik, neuronale Netze und menschliches Denken, Spektrum Akademischer Verlag, 1993

Ulrich M.: *Vorhersage der Erdrotationsparameter mit Hilfe Neuronaler Netze,* IAPG/FESG No. 9, München 2000

Index

Veröffentlichungen in der Schriftenreihe IAPG / FESG (ISSN 1437-8280):
Reports in the series IAPG / FESG (ISSN 1437-8280):

No. 1: Müller J., Oberndorfer H. (1999). *Validation of GOCE Simulation.* ISBN 3-934205-00-3.

No. 2: Nitschke M. (1999). SATLAB – *Ein Werkzeug zur Visualisierung von Satellitenbahnen.* ISBN 3-934205-01-1.

No. 3: Tsoulis D. (1999). *Spherical harmonic computations with topographic/isostatic coefficients.* ISBN 3-934205-02-X.

No. 4: Dorobantu R. (1999). *Gravitationsdrehwaage.* ISBN 3-934205-03-8.

No. 5: Schmidt R. (1999). *Numerische Integration gestörter Satellitenbahnen mit* MATLAB. ISBN 3-934205-04-6.

No. 6: Dorobantu R. (1999). *Simulation des Verhaltens einer low-cost Strapdown-IMU unter Laborbedingungen.* ISBN 3-934205-05-4.

No. 7: Bauch A., Rothacher M., Rummel R. (2000). *Bezugssysteme in Lage und Höhe. Tutorial zum Kursus INGENIEURVERMESSUNG 2000.* ISBN 3-934205-06-2.

No. 8: Rothacher M., Zebhauser B. (2000). *Einführung in GPS. Tutorial zum 3. SAPOS-Symposium 2000 in München.* ISBN 3-934205-07-0.

No. 9: Ulrich M. (2000). *Vorhersage der Erdrotationsparameter mit Hilfe Neuronaler Netze.* ISBN 3-934205-08-9.

No. 10: Seitz F. (2000). *Charakterisierung eines bistatischen Rayleigh- und Raman-Lidars zur Bestimmung von höhenaufgelösten Wasserdampfprofilen.* ISBN 3-934205-09-7.

No. 11: Meyer F. (2000). *Messung von höhenaufgelösten Wasserdampfprofilen unter Verwendung eines bistatischen Raman-Lidars.* ISBN 3-934205-10-0.

No. 12: Peters T. (2001). *Zeitliche Variationen des Gravitationsfeldes der Erde.* ISBN 3-934205-11-9.

No. 13: Egger D. (2001). *Astronomie und Java – Objekte der Astronomie.* ISBN 3-934205-12-7.

No. 14: Steigenberger P. (2002). MATLAB-*Toolbox zur TOPEX/POSEIDON Altimeterdatenverarbeitung.* ISBN 3-934205-13-5.

No. 15: Schneider M. (2002). *Zur Methodik der Gravitationsfeldbestimmung mit Erdsatelliten.* ISBN 3-934205-14-3.

No. 16: Dorobantu R., Gerlach C. (2004). *Investigation of a Navigation–Grade RLG SIMU type iNAV–RQH.* ISBN 3-934205-15-1.

No. 17: Schneider M. (2004). *Beiträge zur Bahnbestimmung und Gravitationsfeldbestimmung mit Erdsatelliten sowie zur Orientierung von Rotationssensoren.* ISBN 3-934205-16-X.

No. 18: Egger D. (2004). *Astro-Toolbox, Theorie.* ISBN 3-934205-17-8.

No. 19: Egger D. (2004). *Astro-Toolbox, Praxis.* ISBN 3-934205-18-6.

No. 20: Fackler U. (2005). *GRACE - Analyse von Beschleunigungsmessungen.* ISBN 3-934205-19-4.

No. 21: Schneider M. (2005). *Beiträge zur Gravitationsfeldbestimmung mit Erdsatelliten.* ISBN 3-934205-20-8.

No. 22: Egger D. (2006). *Sinus-Netzwerk.* ISBN 3-934205-21-6.

Weitere Exemplare können bezogen werden unter:
Copies are available from:
Institut für Astronomische und Physikalische Geodäsie
Technische Universität München
Arcisstrasse 21
D-80290 München
Germany
Telefon: +49-89-289-23190
Telefax: +49-89-289-23178
Email: rechel@bv.tum.de

Oder im Internet:
Or via Internet:
http://tau.fesg.tu-muenchen.de/~iapg/web/veroeffentlichung/schriftenreihe/schriftenreihe.php

ISSN 1437-8280, ISBN 3-934205-21-6