Reconstruction of Dynamical Systems using Constructive Neural Networks

Burkhard v. Stackelberg, Viktor Avrutin, Paul Levi,
Michael Schanz, Georg Wackenhut
Institute of Parallel and Distributed Systems,
University of Stuttgart,
Universitätstrasse 38, D-70569 Stuttgart,
Germany

Abstract

In this work, we present a modern neural network construction method able to build approximations to dynamical systems efficiently while choosing the model size, here the number of neurons, automatically. There is no upper bound as in pruning methods. We present a rather simple approach combining features from Dynamic Node Creation, Cascade Correlation, and Maximum Covariance, and show their capabilities in reconstructing the dynamical behavior of chaotic systems such as the Lorenz and the Rössler system.

Keywords: Neural networks, Dynamical systems, Vector field reconstruction, Dynamic node creation.

I. Introduction

The reconstruction of vector fields of deterministic dynamical systems discrete or continuous in time from time series data is important whenever dynamical processes are observed, where the underlying mathematical description on the microscopic scale is incomplete or even lacking [KKM98], [GL94]. Here, one can distinguish mainly between two cases: On the one hand, there are pure phenomenological models on the macroscopic scale, such as EEG, MEG or ECG models in physiology ([FKH92], [Nun81]), stock index models in finance theory or for instance net traffic models in the field of computer science. On the other hand there are models, where the underlying mathematical description on the microscopic scale is partially known, like some problems in geophysics, meteorology or biology. In both cases, a construction or at least approximation of the corresponding vector field is useful, because the reconstructed vector field can be investigated further in more detail, either by numerical simulation or in some cases even analytically. One possible approach for vector field approximation is given by the theory of neural networks.

The use of neural networks in computer science rose with the abstract analysis of neural information processing through the biologist D. O. Hebb [Heb49] in 1949. In the early years, they were mainly seen as adaptable logic elements and pattern classifiers, as logic seemed the principal issue of human intelligence in these years, and so most neural networks were based on binary neurons.

Neural networks as universal approximations to continuous-valued functions are in use since 1974 [Wer74], and became popular in the 1980s [RHW86]. For a proof of approximation universality see for instance [HSW89]. Some efforts are done towards using neural networks in combination with systems discrete ([Jor86], [Elm90]) and continuous ([Pin87], [Pea89]) in time, whereas other researchers focused on constructive ([Ash89], [FL90a], [Pre97], [Leh99]) and destructive ([LDS90], [HS93], [MS89], [LLM94]) topology optimization. In this work, we present a combination of a constructive approach with time-series prediction using maps and ordinary differential equations. The constructive approach of dynamical insertion of neurons into a network has been investigated by a notable number of researchers. Some of their work is used here.

Since T. Ash [Ash89] developed a method of Multilayer Perceptron (MLP) construction through Dynamic Node Creation (DNC), various MLP construction methods have been proposed, among them the popular Cascade Correlation Learning Architecture (CasCor) by Fahlman and Scott [FL90a], [FL90b], which is most appropriate to classification problems. By the time more methods based on dynamical creation of neurons are developed. Some significant examples on the field of approximation are Cascade/Cand architecture [Pre97], Constructive Back-Propagation (CBP) [Leh99] as well as Brain Construction [HvS01]). The common family structure, which underlies all these methods, is revealed in the last time ([KY97a], [Sta03]). Therefore the modern construction methods are expected to gain more and more popularity.

In this article, a construction strategy is presented built up of elements of DNC, CasCor and CBP. Thereby, DNC determines the way how neurons are inserted into the network and trained, whereas CasCor and CBP are used for initialization of the neurons. It can be shown, that a short covariance training is sufficient to improve the network's performance. DNC is used, as it adapts the whole network for best training and generalization results.

This paper is organized as follows: In section II, we describe the task of reconstruction of dynamical systems from time series using neural networks. Section III describes the developed neural approach in detail, while section IV presents the experimental setup and discusses the results. In sections V and VI we close our article by summarizing and giving an outlook of what enhancements to our approach are possible.

II. Task Definition

Many dynamical systems come as discrete maps governed by a rule

$$\vec{q}(t+1) = \vec{f}(\vec{q}(t)) = \vec{q'}(t), \tag{1}$$

or as ordinary differential equations (ODEs) governed by

$$\dot{\vec{q}}(t) = \vec{f}(\vec{q}(t)) = \vec{q'}(t), \tag{2}$$

whereby $\vec{q}(t)$ denotes the state vector of the system at time t and the target vector $\vec{q'}(t) = \vec{f}(\vec{q}(t))$ represents its time evolution, meaning the next state in (1) and the state vector derivative with respect to time in (2). Concerning the vector function in the equation of motion, there is not much difference between these two classes of dynamical systems, so both classes can be investigated by almost the same methods. Therefore, we focus our studies to continuous systems, because the methods needed for discrete systems are included within. Note that, the values $\vec{q'}(t)$ can be easily constructed from time-series data $\vec{q}(t)$, be it discrete or continuous in time [ASSW02].
The main task of our work is the development of an approach, which fits a neural model function $\vec{f}(\vec{q}(t))$ to a given set of time-series data. Related work is considered in [Gou91], [Gou92], and [GL94]. In former works [ASSW02], we investigated neural time series reconstruction using a fixed neural topology with polynomial input neurons. The advantage of this approach is, that the approximation always has a simple analytical form, and it is most appropriate, where the transitional rule is known or reasonably supposed to be polynomial. But this approach has its limitations: the order of the polynomial has to be known, and the number of neurons grows exponentially with the polynomial order. Furthermore, the network cannot be built using a single type nonlinear neuron, as functions additively composed of polynomials never exceed the highest polynomial order of its components.
Therefore, we present in this work an alternate approach using a single type nonlinear neuron to build a layered feed-forward network including one nonlinear hidden layer. Here, we have used neurons with a tanh transfer function, although others may be used also. This network is dynamically created by stepwise insertion of hidden neurons, so that the size of the network is chosen according to the needs. As a result of the insertion method, the number of neurons only grows linearly in each step.

III. Description of the Method

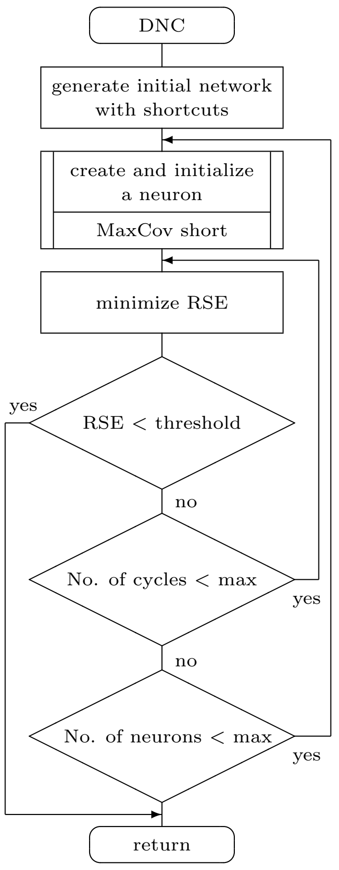
A. Construction of the Network

The algorithm for reconstruction of dynamical systems is based on DNC [Ash89] and consists of the following steps:
1. Generate a network consisting only of input and output neurons and shortcut links, which are initialized as described in section III-B.
2. Insert a new hidden neuron, connecting it backwards to the input, connecting it forwards to the output. The neuron will be initialized using the Maximum Covariance short candidate training as described in section III-C.
3. Train the whole network until training abort criteria are met. We train the network until a given number of cycles is reached or the residual squared error (RSE) is falling below a given threshold, as described in further detail in section III-D.
4. Repeat from 2, until at least one construction abort criterion is fulfilled. Here we repeat until a certain number of hidden neurons is created or the RSE is falling below the threshold given above.
The algorithm of the construction process is presented in Fig. 1, whereas the network topology is illustrated in Figs. 2 and 3. Shown are in-to-hidden, hidden-to-out and shortcut connections. As not in use in our experiments, hidden to hidden connections are left out. Network links are shown as edges of the graph in Fig. 2 and, for better visibility, as link knots on the line crossings in Fig. 3, whose direction is defined through the arrows. In both figures, one sees a new neuron being added from a pool of candidates. A neuron is held from this pool via the initialization procedure described below.

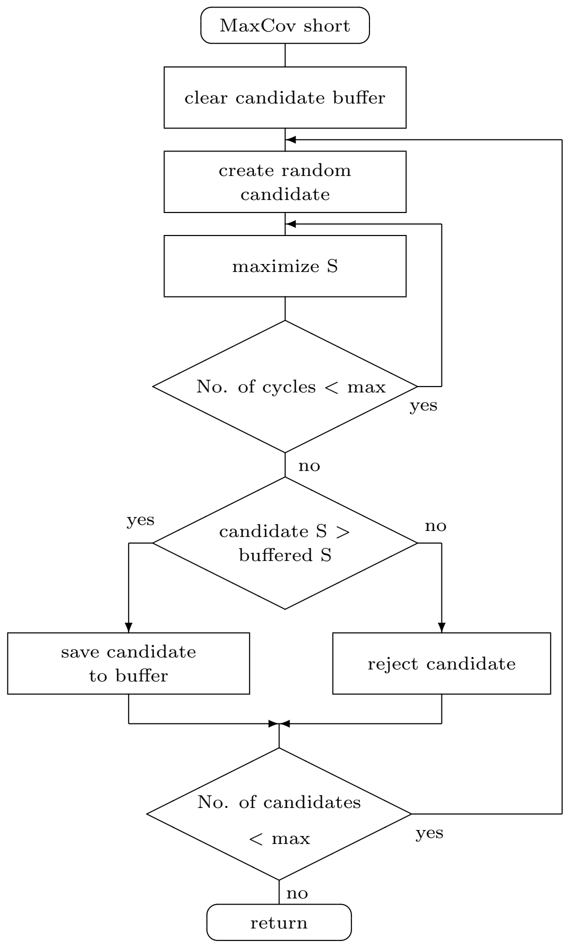
B. Initialization

As mentioned above, the network consisting of input and output neurons and direct links between them is initialized. As we choose the output neurons as linear, direct connections may be "trained" analytically solving the linear equation

$$\frac{\partial E(w_{ij})}{\partial w_{ij}} = 0 \tag{3}$$



(a) Modified Dynamic Node Creation (DNC) as general construction procedure.



(b) Maximum Covariance short candidate training (MaxCov short) as a part of the general construction procedure.

Fig. 1. The used network construction algorithm

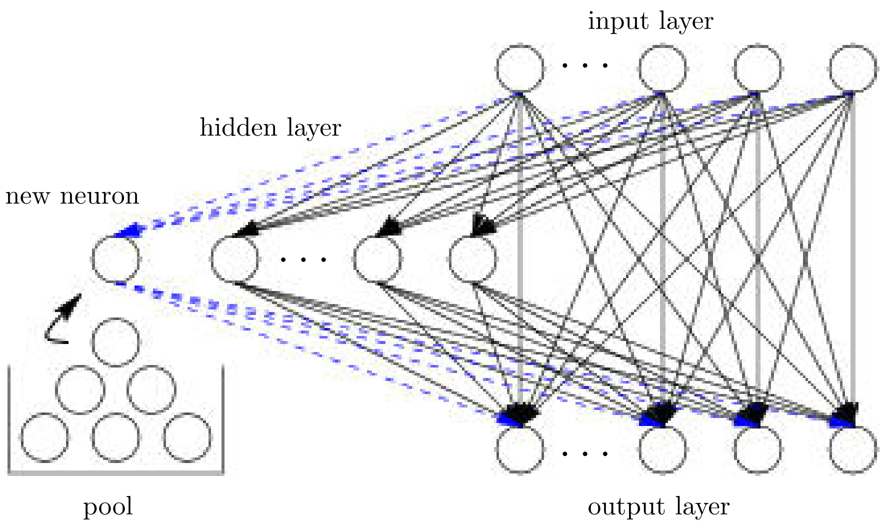


Fig. 2. Network topology and node insertion according to the Dynamic Node Creation procedure in its usual graph representation. For details see text.

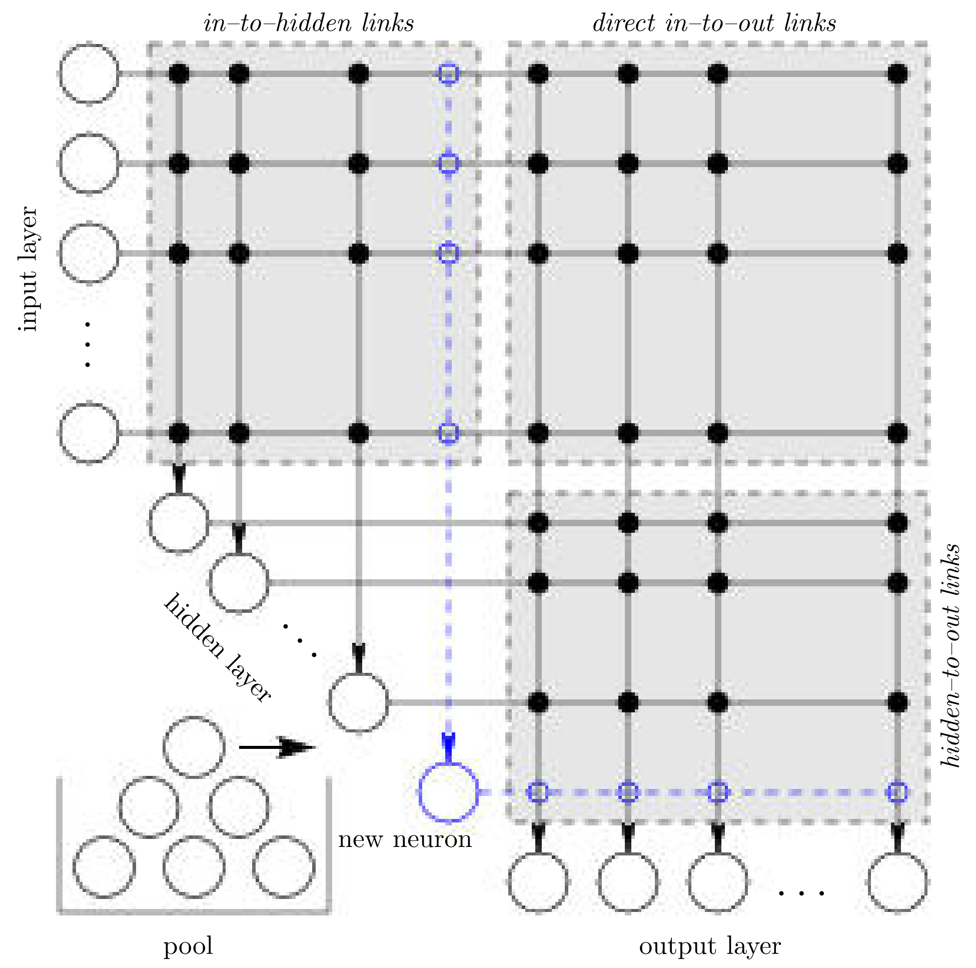


Fig. 3. Network topology and node insertion according to the Dynamic Node Creation procedure, in grid representation. For details see text.

for all network weights w_{ij}, as long as the objective function $E(w_{ij})$ is of second order in the weights, but iterative methods, as used during fixations, also apply. For this reason, and as implementation is simpler, we decided for the iterative approach using Resilient Back-propagation (Rprop) [RB92].

C. Maximum Covariance Short Candidate Training

After the network's initial creation, the construction cycles through creation, initialization and fixation of hidden neurons, until the whole construction process stops.
For this reason, a pool of identical neurons is created, which only differ by their initialization, done by random. Each of these neurons is shortly trained to maximize the sum S of absolute covariances

$$S = \sum_k \left| \sum_p (y^h_{j,p} - \bar{y}^h_j)(e^o_{k,p} - \bar{e}^o_k) \right| \tag{4}$$

where $y^h_{j,p}$ is the candidate's j output at pattern p, $e^o_{k,p}$ is the output error at output neuron k at the same pattern and the barred values are the mean over all patterns. The selection of one neuron from this pool is done by choosing the candidate in order to maximize S. Candidate neurons are connected only backwards at this stage for they do not influence the output and each other yet. After covariance training, the neuron yielding maximal covariance, is selected for permanent insertion.
Instead of this approach, alternatives may be used also. While CasCor trains candidates until convergence, which is inappropriate for continuous approximation tasks, MaxCov as applied to CBP shrinks to a pure selection without previously training the neurons. Here, a short training of a few cycles is done, for this seems to be sufficient to improve selection results noticeably with low cost.

D. Fixation (Main Training)

After insertion and initialization of a new neuron, the network is trained until abort criteria are met, using the main objective function E, which is the Relative Squared Error RSE as described below. The maximal number of cycles should be considerably large to assure a good approximation and convergent behavior.
The natural quality measure of a neural network is the objective function it is trained upon. Here, the squared error is in use, as it is easy to calculate and optimize. If we normalize the squared error, which happens to be the residual variance of the neural model, by the variance of the measured data, we obtain a quality measure ranging from 0 to 1 on the training set, as a sensitive data model fits the data at least as good as the data's mean whose residual variance equals the data variance. The normalized, relative square error RSE may then be expressed as

$$E = RSE = \frac{ResidualVariance}{DataVariance} = \frac{\sum_{k,p}(t_{k,p} - y^o_{k,p})^2}{\sum_{k,p}(t_{k,p} - \bar{t})^2} \tag{5}$$

where $t_{k,p}$ are the measured data, used as teacher input, and $y^o_{k,p}$ the neural model values, k ranging through all output neurons and p ranging through all patterns (norming factors of the variances cancel out). The value $\bar{t}$ denotes the mean of all measured data $t_{k,p}$. If one wants to take individual variances of individual outputs into account, equation (5) changes to

$$E = RSE = \frac{1}{d}\sum_{k=1}^{d} \frac{\sum_p (t_{k,p} - y^o_{k,p})^2}{\sum_p (t_{k,p} - \bar{t}_k)^2} \tag{6}$$

where d is the output layer's dimension, k identifies individual output neurons and p the individual patterns, and $\bar{t}_k$ denotes the mean of all measured data corresponding to a given output neuron k.
On test data, the RSE may take values larger than 1, as the test data may differ in their statistics from the training data in use.

E. Integrating Dynamics

The neural networks described above can be converted to "recurrent" networks by embedding them into a dynamical system as the transition function that governs the dynamic. In the case of ODEs, the dynamical system then may be integrated numerically. The target vector $\vec{q}'(t)$ is given in this case by the derivative $\dot{\vec{q}}(t)$. In the case of discrete dynamics, i. e. maps, the network has just to calculate the system's state at the next time step, where the target vector $\vec{q}'(t)$ is given by the next state in the time evolution $\vec{q}(t+1)$. In both cases, it is trained to approximate the original system function which is known numerically on the time series.
The unknown system function f is calculated using the neural network by identifying the network's input vector $\vec{x}$ with the state vector $\vec{q}$ and the network's output vector $\vec{y^o}$ with the target vector $\vec{q}'$. During the training instead, not the network output vector, but the teacher input, as in equation (5), is identified by a numerical estimate of $\dot{\vec{q}}'$.
The training may be done here without the need of Back-Propagation Through Time (BPTT) [RHW86] or Real Time Recurrent Learning (RTRL) [WZ89], as the whole time series is known. So, the network can be trained using non-recurrent methods as if in non-recurrent environments.

IV. Experiments

A. Setup

In our experiments, the vector field is reconstructed by a neural network with up to 40 hidden neurons, whose hidden layer size is restricted by an error stop criterion: construction is stopped when training error falls below a given level. The levels of the relative squared error are given as 0.1, 0.01, 0.003, 10^{-3}, 10^{-4} and 10^{-5}. Construction is also stopped whenever a construction step does not succeed lowering the network's training error.
The network topology consists of an input layer, an output layer and a single hidden layer; connections exist from input layer to hidden and output layer (shortcut connections) and from hidden to output layer. The input and output transfer functions are identity, whereas the hidden layer transfer function is tanh.
First, linear direct connections are pre-trained obtaining a linear model in the beginning. The network is then trained using Rprop [RB92] through 1000 cycles to optimize the network after insertion of each neuron selected from 100 pool candidates in each construction step. Candidates are trained and selected using Conjugate Gradient Descent (CGD) [PFTV88] training to maximize covariance as described above, using 3 training steps.
The networks have been trained on 1000 equidistant samples, taken from data of the Lorenz 63 system [Lor63] holding 10^4 points with a time step of $dt = 10^{-2}$, and from data of the Rössler system [Rös76] holding 10^5 points with a time step of $dt = 10^{-3}$. In both cases, the data stems from the chaotic regime of these systems, whereby the transient dynamics is not included.

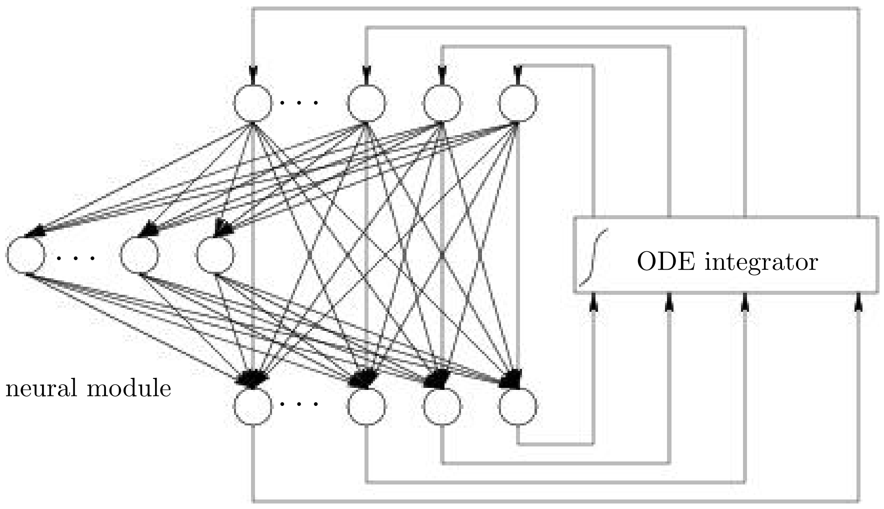


Fig. 4. The neural network as kernel function in a differential equation connecting the output to the input through an ODE integrator

The Lorenz system [Lor63] is given through the equations

$$\begin{aligned}\dot{x} &= \sigma(y-x)\\ \dot{y} &= x(r-z)-y\\ \dot{z} &= xy-bz\end{aligned} \tag{7}$$

where $\sigma = 16$, $r = 45.92$ and $b = 4$.
The Rössler system [Rös76] is given through the equations

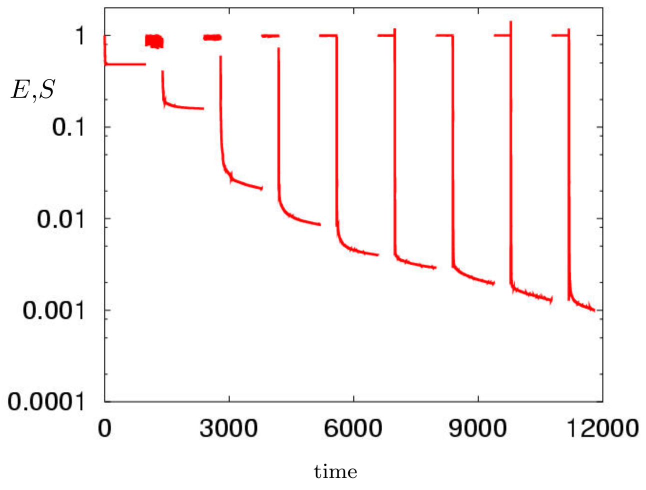
$$\begin{aligned}\dot{x} &= -(y+z)\\ \dot{y} &= x+ay\\ \dot{z} &= b+z(x-c)\end{aligned} \tag{8}$$

where $a = 0.15$, $b = 0.2$ and $c = 10$.
Prediction is done for 10^5 points from the first point of the training example using a time step of $dt = 10^{-3}$.

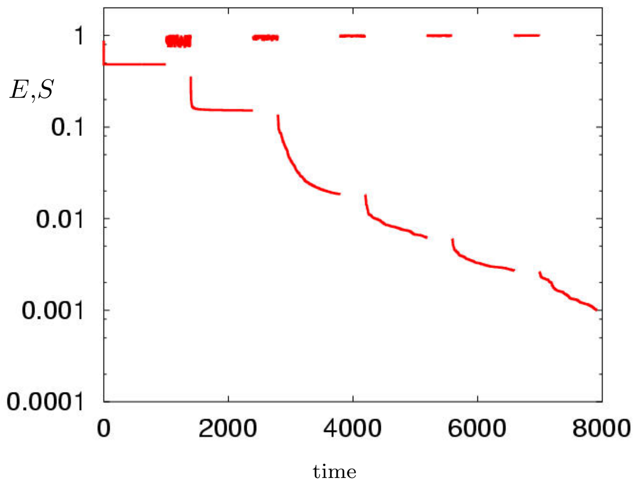
B. Example Runs

The construction method can be more clearly understood by examining the evolution of the error functions in example runs as in Figs. 5(a) and 5(b). In both runs, CGD is used as training method during candidate training. However, they differ in the training method used during the neuron's fixation. While the run in Fig. 5(a) is using Rprop, the run in Fig. 5(b) is using CGD here too. While CGD, as described in [PFTV88], has a fast and stable convergence in every step, the individual CGD step takes a long time while doing a line minimum search, which evaluates the objective function several times. In contrast, Rprop takes no long time in each step, as the objective function is calculated only once, but converging reasonably well, it is not assured to converge at its individual step.
The first L-shaped part of the graph in both figures represents the RSE, short E in the graphs, while shortcut training. The fast step convergence of CGD is clearly seen. Not seen in the graph is the total training cost.
The seemingly stochastic part in the graph following the first L is the first candidate phase of the training. Here, not RSE but a transformation S_T of S is measured with $S_T = \frac{1}{1+S}$, for the "error" is decreasing during the training, whilst S is increasing. Further, 1 is added to the denominator, so that S_T never is infinite and resides in the interval of $[0, 1]$, just as the RSE value. This part of the graph consists of the short MaxCov training S_T (S in the graphs) values of each candidate neuron to be tested for insertion. Candidates are trained sequentially, as the computer on which the calculation is performed is serial. Each candidate differs from the others by its pseudo-random initialization, hence the different values of their training curves.
Consequently, a further L-shaped part of the graph represents the fixation RSE of the network after introduction of its first candidate. Starting with a good candidate initialization, the network will continue just near its last RSE value and soon fall below.

Initialization and fixation phases follow in alteration until construction abort criteria are met. As one can see, in the case of Rprop (Fig. 5(a)), at low errors do the assumptions to good initial step-widths, valid at the beginning of the construction, not hold any more: The first step takes the network parameters far from the optimum, and many steps following are needed to return to the previously found (or even another) optimum. At even lower error levels, the construction may be abandoned, due to the fact, that within a fixed number of training cycles the RSE value falls not below the RSE value of the predecessor. Obviously, a modification of the original Rprop procedure or the construction method is needed.



(a) using Rprop



(b) using CGD

Fig. 5. Typical graphs of the objective functions through time. Explanation see text.

C. Results

C.1 Network Sizes

As shown in the left parts of Tabs. I and II, the network sizes cover a large range beginning with 2 hidden neurons at the highest error level of $RSE = 0.1$ up to over 20 neurons at the lowest error levels. At the lowest levels, some of the networks do not reach the scheduled RSE due to the dynamics of Rprop training which jumps off the optimum reached at the last construction step when training error already has become small. The runs not reaching their nominal error level are displayed in the right parts of Tabs. I and II. Still, most networks at this point represent reasonable approximations to the dynamics. Network sizes for successful and abandoned constructions are displayed in combination in the Figs. 6 and 7.

With the help of linear shortcuts, a single hidden neurons with tanh transfer function may approximate multivariate second-order polynomials of rank 1 asymptotically. Hereby, the rank of the multivariate polynomial is defined as the rank of the coefficient matrix associated with the second order terms. Higher-rank multivariate second-order polynomials may be composed from rank 1 polynomials successively, resulting in as much hidden neurons as the rank of the coefficient matrix. Higher-order polynomials may be approximated also by tanh neurons in a similar way. As the Lorenz field contains two rank 2 polynomials, ideally 4 hidden neurons should suffice to approximate the field, while the Rössler field contains only one rank 2 polynomial, thus 2 hidden neurons should do the task.

This number is reached in practice only at the highest RSE levels. The number of hidden neurons depends mostly on experimental settings as the approximation method with its parameters and convergence rate and on training time. The optimum is never reached, as it does not exist for finite parameters of the network. This is due to the used function system which can approximate a polynomial function only asymptotically. In contrast to that, for instance, a neural network with piecewise linear neurons or neurons which converge to polynomials at finite parameters, maybe through re-parameterizing the tanh function, can do so.

C.2 System Reconstructions

Figs. 8 to 11 and 12 to 15 show examples of dynamics reconstructed from data of the Lorenz and Rössler system at several error levels and coming from different network initializations. It turned out, that the Lorenz system is more tolerant to training error than the Rössler system. We also found out, that prediction quality not only depends on the squared error where the network converged to, but also on the network initialization which was done by random.

The Lorenz attractor, for example, is approximated well at an error level of 10^{-4} to 10^{-5} by nearly all training runs, as seen in Fig. 8. At higher levels, more and more runs fail to approximate well, and trajectories typically break down to more simple attractors such as the two-banded attractor in Fig. 11 or even stable points as in Fig. 10 which originally were unstable. But still, some attractors reveal a more or less close approximation to the original Lorenz attractor, as in Fig. 9.
For the Rössler attractor, similar observations can be made. At a nominal error level of 10^{-5}, although not reached by the runs, most trajectories reveal the typical Rössler dynamics, as seen in Fig. 12. Up to a level of $3 \cdot 10^{-3}$, matching trajectories may be found, as in Fig. 13. At the higher level, deviations from the original dynamics such as periodical attractors (Fig. 14) or rather strange dynamics (Fig. 15) may be observed.
Obviously, dynamical reconstruction quality cannot be identified exactly with regression quality – at a given regression quality, reconstruction quality may vary strongly depending on the network's initialization. Therefore a reconstruction quality measure is to be developed in further research. For instance, one could measure the maximum distance of points of one attractor to the other attractor, or compare the power spectra and Lyapunov exponents of the attractors. It is due to the fact that regression quality is only a measure of how good the data are fit in the points where they are given. But, once the reconstructed trajectory leaves the original attractor topology, it is not guaranteed to ever return, as long as the model itself is not guaranteed forcing such trajectories.

V. Summary

In this paper, we presented a method for reconstruction of dynamical systems both continuous and discrete in time using a modern neural network construction method. We showed that this can be done modularly combining methods from several fields, i. e. combining the construction method of Dynamic Node Creation (DNC) with neuron initialization and selection from Maximum Covariance short training (MaxCov short). We also showed that this method is able to construct a model of a chaotic dynamical system, which reconstructs its behavior reasonably well at a given residual error level, while the error level left alone gives no definitive measure of the reconstruction quality. The model complexity, given through the number of hidden neurons used, was adapted according to the needs of approximating at a given error level.

VI. Outlook

One may wish to improve the method further. As the system is constructed modularly, many of the decisions made are not fixed. Especially one could test more modern momentum based objective functions [KY97b] instead of covariance to improve the hidden neuron initialization. One could change the linking and freezing behavior according to a BCA stylish scheme [HvS01], [Sta03], where in-to-hidden links are only trained for the actual candidate units. Other types of nonlinear hidden neurons such as neurons with Gauss and abs transfer functions may be used, or even a set of several neural transfer functions may be tested and used during one construction.
The problem of our approach is, that there is no information about the investigated dynamics accessible from the network beyond its function system, neither is there an access to a unique description, e. g. in terms of polynomial components. The solution of this problem will be presented in a further article, where we will describe the extraction of a polynomial model from a neural network using integral transforms. As one may assume the system's behavior to be polynomial, this assumption is confirmed if the transformation contains only a finite number of relevant terms.

References

[Ash89] T. Ash. Dynamic node creation in backpropagation networks. *Connection Science*, 1(4):365–375, 1989.

[ASSW02] V. Avrutin, M. Schanz, F. Schreiber, and G. Wackenhut. Reconstruction of vector fields of dynamical systems from time series data: A neural network approach. In *Proceedings of the 3rd WSEAS International Conference on: Neural Networks and Applications NNA '02*, pages 3601–3606, 2002.

[Elm90] J. L. Elman. Finding structure in time. *Cognitive Science*, 14:179–211, 1990.

[FKH92] A. Fuchs, J.A.S. Kelso, and H. Haken. Phase Transitions in the Human Brain: Spatial Mode Dynamics. *Int. J. of Bifurcation and Chaos*, 2(4):917, 1992.

[FL90a] S. E. Fahlman and C. Lebiere. The cascade-correlation learning architecture. Technical Report CMU-CS-90-100, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1990.

[FL90b] S. E. Fahlman and C. Lebiere. The cascade-correlation learning architecture. In D. S. Touretzky, editor, *Advances in Neural information processing systems*, volume 2, pages 524–532. Morgan Kaufmann, San Mateo, CA, 1990.

[GL94] G. Gouesbet and C. Letellier. Global vector-field reconstruction by using a multivariate polynomial l_2 approximation on nets. *Phys. Rev. E*, 49(6):4955–4972, 1994.

[Gou91] G. Gouesbet. Reconstruction of standard and inverse vector fields equivalent to a Rössler system. *Phys. Rev. A*, 44(10):6264–6280, 1991.

[Gou92] G. Gouesbet. Reconstruction of vector fields: The case of the Lorenz system. *Phys. Rev. A*, 46(4):1784–1796, 1992.

[Heb49] D. O. Hebb. *The Organization of Behavior*, chapter Introduction and 4: The first stage of perception: growth of an assembly, pages xi–xix, 60–78. Wiley, New York, 1949.

[HS93] B. Hassibi and D. G. Stork. Second order derivatives for network pruning: Optimal brain surgeon. In J. D. Cowan S. J. Hanson and C. L. Giles, editors, *Advances in Neural Information Processing Systems*, volume 5, pages 164–171. Morgan Kaufman, San Mateo, CA, 1993.

[HSW89] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366, 1989.

[HvS01] G. Haag and B. v. Stackelberg. Aufbau und Test des optimierten und selbstgenerierenden neuronalen Netzwerks. In *[vorläufiger] Endbericht Verbundprojekt: Selbstgenerierende neuronale Netze zur automatischen Qualitätssicherung und Fehlerdiagnose beim Spritzgießen, AQF*, chapter 7, pages 21–50. Steinbeis-Transferzentrum angewandte Systemanalyse, Stuttgart, Januar 2001.

[Jor86] M. I. Jordan. Attractor dynamics and parallelism in a connectionist sequential machine. In *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*, pages 531–546, Hillsdale NJ, 1986. Erlbaum.

[KKM98] Y. Kuroe, H. Kawakami, and T. Mori. Neural Network Learning for Vector Field Approximation from Sparse Data. In Sh. Kitamura and K. Uchida, editors, *Proc. of the* 8^{th} *Japanese–German Seminar on Nonlinear Problems in Dynamical Systems – Theory and Applications –*, pages 217–228, 1998.

[KY97a] T. Kwok and D. Yeung. Constructive algorithms for structure learning in feedforward neural networks for regression problems. *IEEE Transactions on Neural Networks*, 8(3):630–645, 1997.

[KY97b] T. Kwok and D. Yeung. Objective functions for training new hidden units in constructive neural networks. *IEEE Transactions on Neural Networks*, 8(5):1131–1148, 1997.

[LDS90] Y. LeCun, J. S. Denker, and S. A. Solla. Optimal brain damage. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 2, pages 598–605. Morgan Kaufman, San Mateo, CA, 1990.

[Leh99] M Lehtokangas. Modelling with constructive backpropagation. *Neural Networks*, 12(4?):707–716, 1999.

[LLM94] A. U. Levin, T. K. Leen, and J. E. Moody. Fast pruning using principal components. In G. Tesauro J. D. Cowan and J. Alspector, editors, *Advances in Neural Information Processing Systems*, volume 6, pages 35–42. Morgan Kaufmann, San Mateo, CA, 1994.

[Lor63] E.N. Lorenz. Deterministic non-periodic flows. *J. Atmos. Sci.*, 20:130, 1963.

[MS89] M. Z. Mozer and P. Smolensky. Skeletonization: A technique for trimming the fat from a network via relevance assessment. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 1*, pages 107–115. Morgan Kaufmann, San Mateo, CA, 1989.

[Nun81] P.L. Nunez. *Electric fields of the brain.* Oxford University Press, Oxford, New York, 1981.

[Pea89] B. Pearlmutter. Learning state space trajectories in recurrent networks. *Neural Computation*, 1(2):263–269, 1989.

[PFTV88] W. H. Press, B. H. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes in C*, chapter 10.6 Conjugate Gradient Methods in Multidimensions, pages 317–323. Cambridge University Press, first edition, 1988.

[Pin87] F. J. Pineda. Generalization of backpropagation to recurrent neural networks. *Physical Review Letters*, 59(19):2229–2232, November 1987.

[Pre97] L. Prechelt. Investigation of the cascor family of learning algorithms. *Neural Networks*, 10(5):885–896, 1997.

[RB92] M. Riedmiller and H. Braun. Rprop – a fast adapting learning algorithm. Technical report, Universität Karlsruhe, 1992.

[RHW86] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. In D. E. Rumelhart and J. L. McClelland, editors, *Foundations*, volume 1 of *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, pages 318–362. MIT Press, Cambridge, MA, 1986.

[Rös76] O. E. Rössler. An equation for continuous chaos. *Phys. Lett. A*, 57:397–398, 1976.

[Sta03] Burkhard von Stackelberg. *Konstruktionsverfahren vorwärtsgerichteter neuronaler Netze.* PhD thesis, Universität Stuttgart, July 2003.

[Wer74] P. Werbos. *Beyond regression: new tools for prediction and analysis in the behavioral sciences.* PhD thesis, Harvard Univerity, Cambridge, MA, 1974.

[WZ89] R. J. Williams and D. Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1:270–280, 1989.

Appendix

	RSE					
N	10^{-1}	10^{-2}	$3 \cdot 10^{-2}$	10^{-3}	10^{-4}	10^{-5}
1	0	0	0	0	0	0
2	**100**	0	0	0	0	0
3	0	**22**	0	0	0	0
4	0	**78**	0	0	0	0
5	0	0	**39**	0	0	0
6	0	0	**59**	0	0	0
7	0	0	**2**	**2**	0	0
8	0	0	0	**60**	0	0
9	0	0	0	**32**	0	0
10	0	0	0	**6**	0	0
11	0	0	0	0	0	0
12	0	0	0	0	0	0
13	0	0	0	0	0	0
14	0	0	0	0	0	0
15	0	0	0	0	**1**	0
16	0	0	0	0	**2**	0
17	0	0	0	0	**7**	0
18	0	0	0	0	**14**	0
19	0	0	0	0	**13**	0
20	0	0	0	0	**4**	0
21	0	0	0	0	0	0
22	0	0	0	0	**2**	0
23	0	0	0	0	0	0
24	0	0	0	0	0	0
25	0	0	0	0	0	0

	RSE					
N	10^{-1}	10^{-2}	$3 \cdot 10^{-2}$	10^{-3}	10^{-4}	10^{-5}
1	0	0	0	0	0	0
2	0	0	0	0	0	0
3	0	0	0	0	0	0
4	0	0	0	0	0	0
5	0	0	0	0	0	0
6	0	0	0	0	0	0
7	0	0	0	0	0	0
8	0	0	0	0	0	0
9	0	0	0	0	0	**1**
10	0	0	0	0	0	**1**
11	0	0	0	0	**1**	**1**
12	0	0	0	0	**1**	**3**
13	0	0	0	0	**4**	**3**
14	0	0	0	0	**7**	**6**
15	0	0	0	0	**5**	**12**
16	0	0	0	0	**15**	**10**
17	0	0	0	0	**13**	**10**
18	0	0	0	0	**4**	**10**
19	0	0	0	0	**3**	**9**
20	0	0	0	0	**2**	**9**
21	0	0	0	0	**1**	**16**
22	0	0	0	0	0	**4**
23	0	0	0	0	**1**	**1**
24	0	0	0	0	0	**2**
25	0	0	0	0	0	**2**

TABLE I

Lorenz run statistics at several error levels. Run frequency over number of neurons N and residual errors RSE. Left part: RSE level reached. Right part: RSE level not reached, construction abandoned. Description see section IV-B.

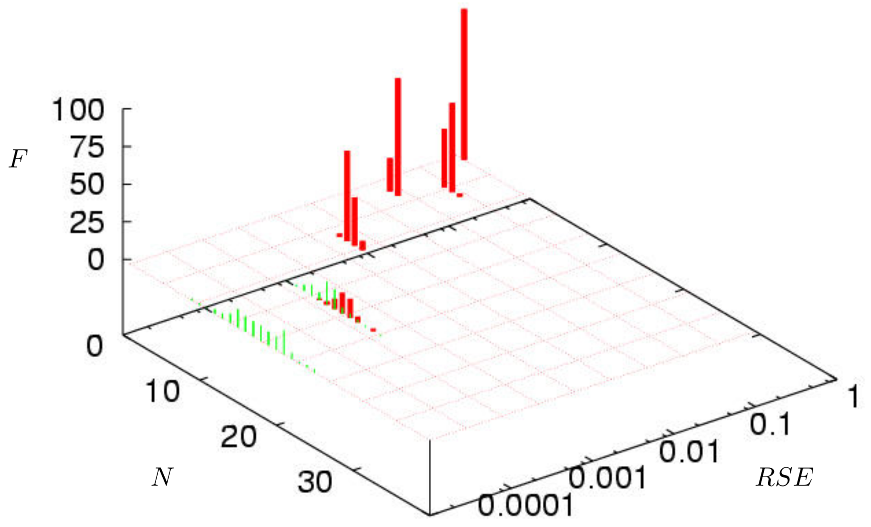


Fig. 6. *Lorenz run statistics at several error levels. Run frequency F over number of neurons N and residual errors RSE. Compare also Tab. I.*

	RSE					
N	10^{-1}	10^{-2}	$3 \cdot 10^{-2}$	10^{-3}	10^{-4}	10^{-5}
1	**1**	0	0	0	0	0
2	**81**	0	0	0	0	0
3	**17**	**1**	0	0	0	0
4	0	**56**	**2**	0	0	0
5	**1**	**29**	**15**	0	0	0
6	0	**11**	**39**	**2**	0	0
7	0	**2**	**26**	**18**	0	0
8	0	**1**	**12**	**30**	0	0
9	0	0	**5**	**20**	0	0
10	0	0	**1**	**22**	0	0
11	0	0	0	**6**	**1**	0
12	0	0	0	**2**	**2**	0
13	0	0	0	0	**2**	0
14	0	0	0	0	**3**	0
15	0	0	0	0	**5**	0
16	0	0	0	0	**8**	0
17	0	0	0	0	**4**	0
18	0	0	0	0	**2**	0
19	0	0	0	0	**4**	0
20	0	0	0	0	**1**	0
21	0	0	0	0	**2**	0
22	0	0	0	0	0	0
23	0	0	0	0	0	0
24	0	0	0	0	0	0
25	0	0	0	0	0	0
26	0	0	0	0	0	0

	RSE					
N	10^{-1}	10^{-2}	$3 \cdot 10^{-2}$	10^{-3}	10^{-4}	10^{-5}
1	0	0	0	0	0	0
2	0	0	0	0	0	0
3	0	0	0	0	0	0
4	0	0	0	0	0	0
5	0	0	0	0	0	0
6	0	0	0	0	**1**	0
7	0	0	0	0	0	0
8	0	0	0	0	0	**2**
9	0	0	0	0	**2**	**1**
10	0	0	0	0	**4**	**1**
11	0	0	0	0	**5**	**2**
12	0	0	0	0	**5**	**6**
13	0	0	0	0	**6**	**8**
14	0	0	0	0	**8**	**10**
15	0	0	0	0	**10**	**13**
16	0	0	0	0	**9**	**15**
17	0	0	0	0	**8**	**9**
18	0	0	0	0	**3**	**10**
19	0	0	0	0	**3**	**8**
20	0	0	0	0	**1**	**4**
21	0	0	0	0	0	**4**
22	0	0	0	0	0	0
23	0	0	0	0	**1**	**3**
24	0	0	0	0	0	**2**
25	0	0	0	0	0	0
26	0	0	0	0	0	**2**

TABLE II

Rössler run statistics at several error levels. Run frequency over number of neurons N and residual errors RSE, Left part: RSE level reached. Right part: RSE level not reached, construction abandoned. Description see section IV-B.

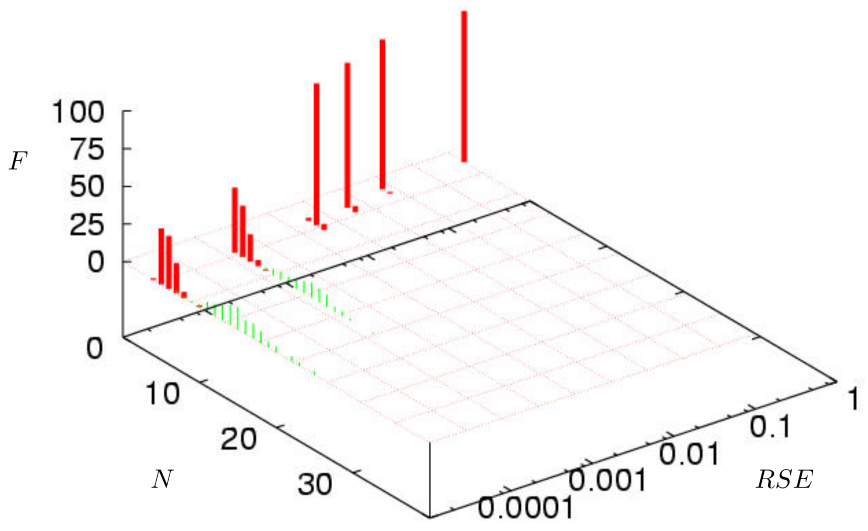


Fig. 7. *Rössler run statistics at several error levels. Run frequency F over number of neurons N and residual errors RSE. Compare also Tab. II.*

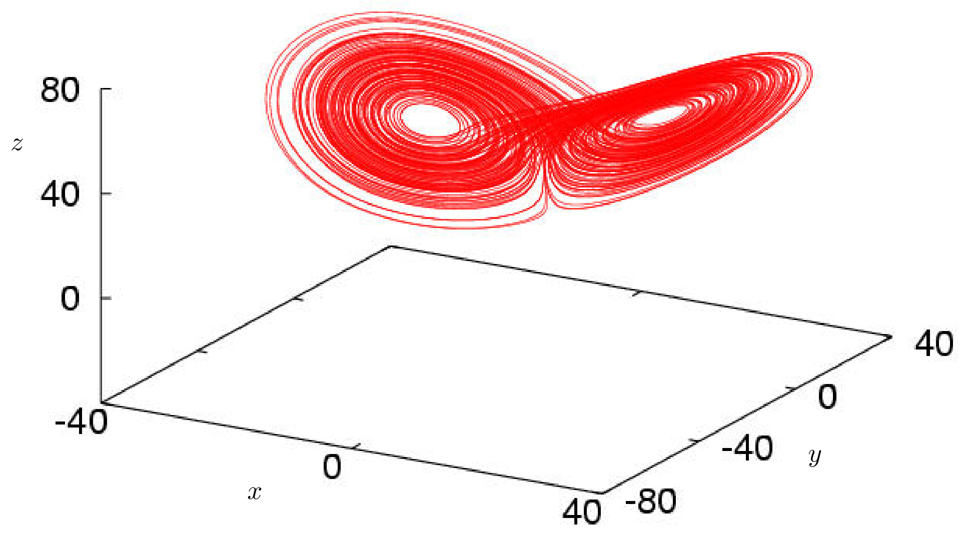


Fig. 8. Approximation to the Lorenz 63 attractor. Run with $RSE = 10^{-4}$. As network initializations vary randomly, each approximation even at the same RSE level has its individual behavior. At this level, all approximative trajectories follow the Lorenzian topology.

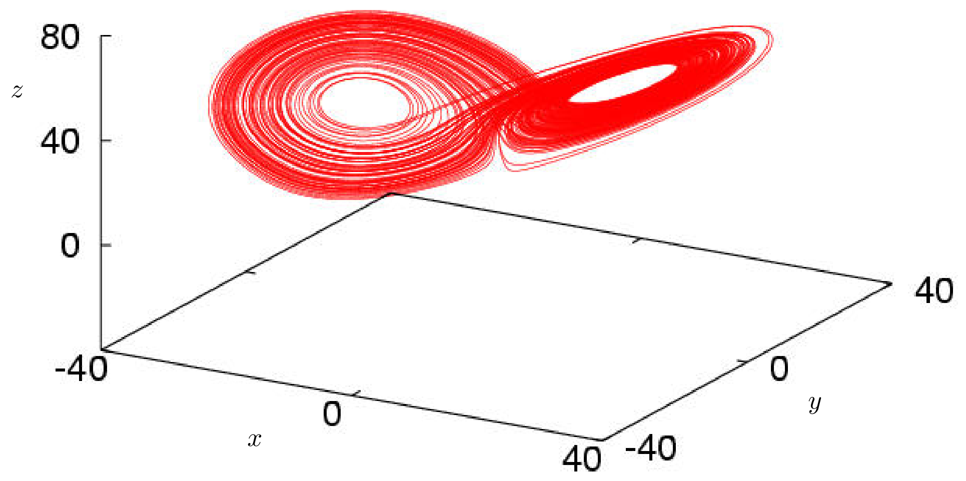


Fig. 9. Approximation to Lorenz 63 attractor. Run with $RSE = 10^{-2}$. As network initializations vary randomly, each approximation even at the same RSE level has its individual behavior. At this level, still some approximative trajectories fit the Lorenzian topology well.

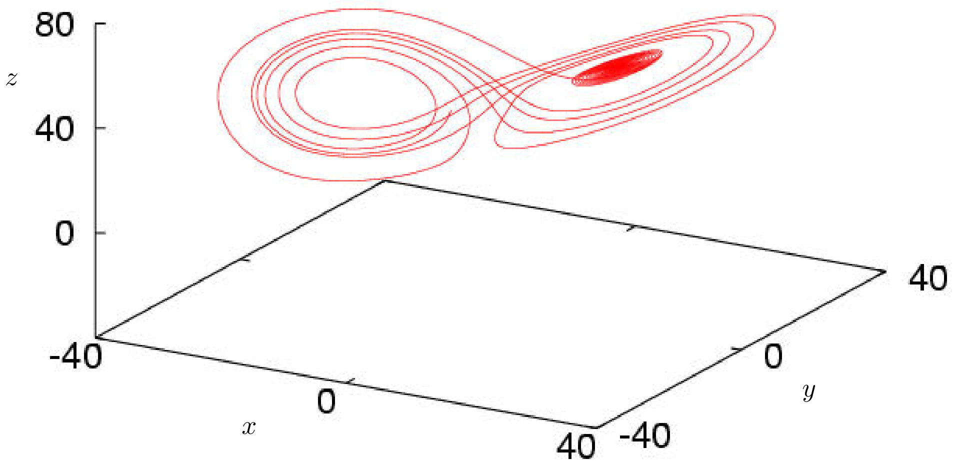


Fig. 10. Approximation to the Lorenz 63 attractor. Run with $RSE = 10^{-2}$. As network initializations vary randomly, each approximation even at the same RSE level has its individual behavior. After some Lorenz cycles, this trajectory collapses to a stable point, which in the original system was unstable.

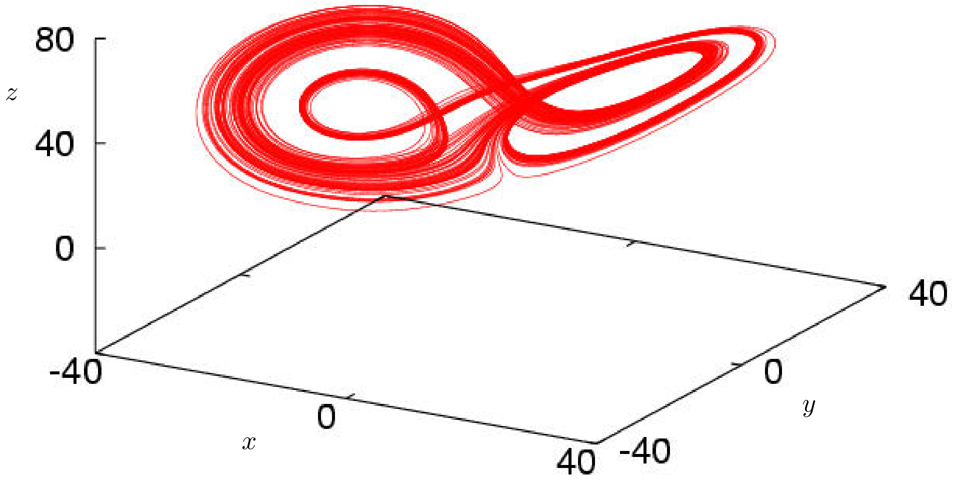


Fig. 11. Approximation to the Lorenz 63 attractor. Run with $RSE = 10^{-2}$. As network initializations vary randomly, each approximation even at the same RSE level has its individual behavior. This trajectory follows a two-banded attractor.

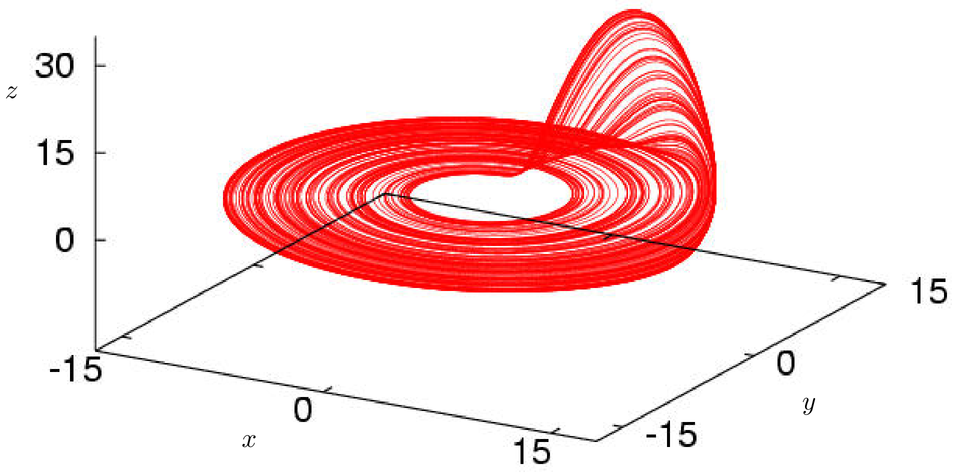


Fig. 12. Approximation to the Rössler attractor. Run with $RSE = 10^{-5}$. As network initializations vary randomly, each approximation even at the same RSE level has its individual behavior. Most trajectories at this RSE level follow the attractor topology just well, though the nominal level is not reached because of abandoning the construction earlier!

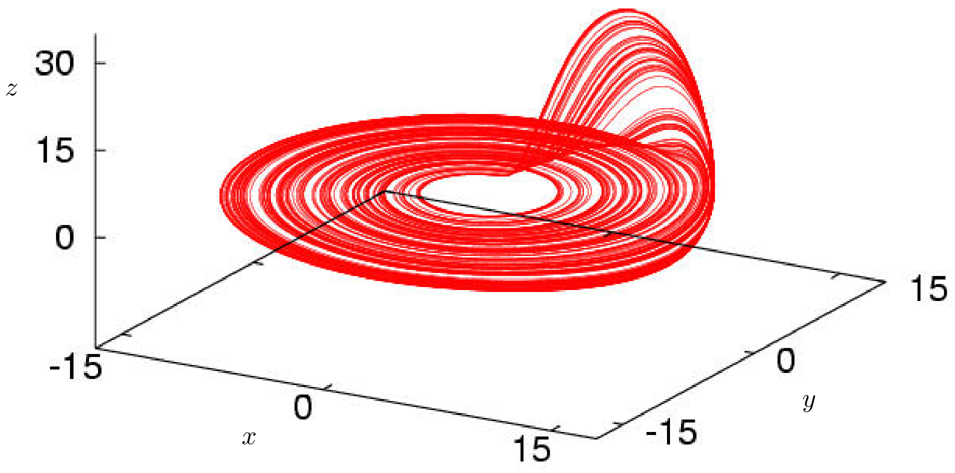


Fig. 13. Approximation to the Rössler attractor. Run with $RSE = 3 \cdot 10^{-3}$. As network initializations vary randomly, each approximation even at the same RSE level has its individual behavior. At this level, still some trajectories follow the attractor topology fairly well.

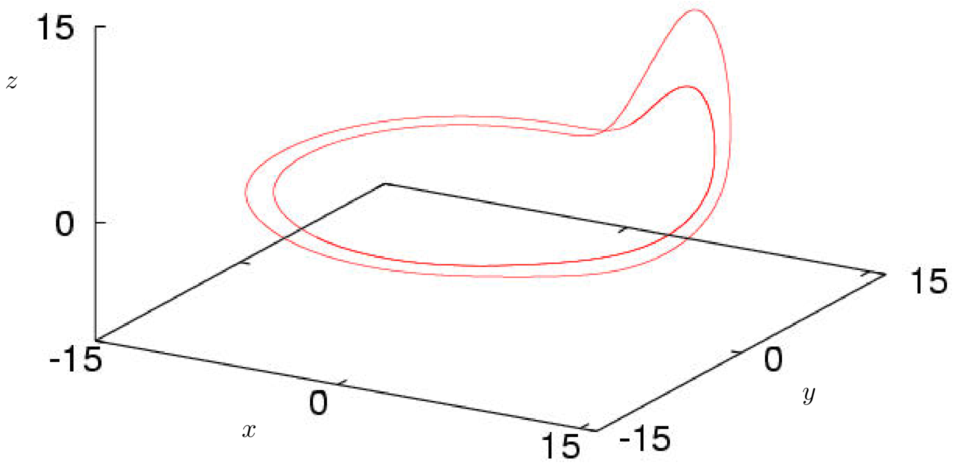


Fig. 14. Approximation to the Rössler attractor. Run with $RSE = 3 \cdot 10^{-3}$. As network initializations vary randomly, each approximation even at the same RSE level has its individual behavior. This trajectory follows a period-two attractor.

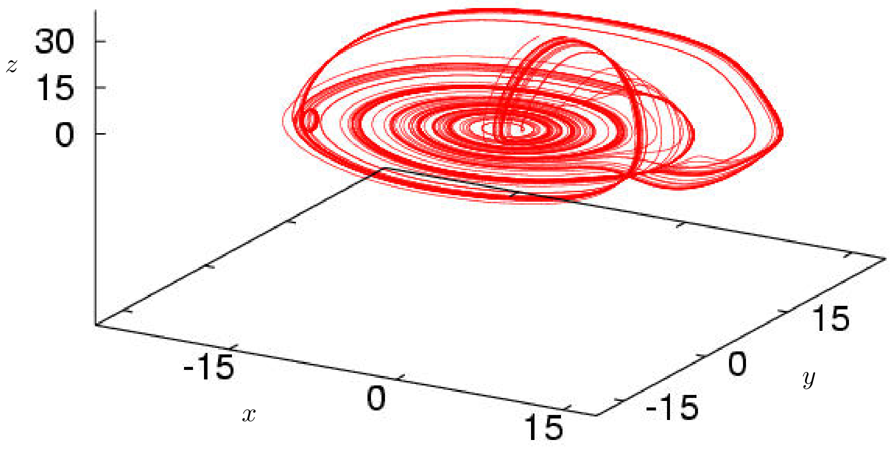


Fig. 15. Approximation to the Rössler attractor. Run with $RSE = 10^{-2}$. As network initializations vary randomly, each approximation even at the same RSE level has its individual behavior. This trajectory follows a fancy attractor. For Rössler, even more fancy approximations of varying "styles" may be found.