Using Chaos to Characterize and Train Neural Networks

Oguz Yetkin[*]

J.C. Sprott[†]

Department of Physics, University of Wisconsin

Madison, WI 53706 USA

Abstract

An artificial neural network can be behave as a dynamical system if the outputs generated by the network are used as inputs for the next time step. In this way, a time series can be obtained from the network. These networks are also capable of chaotic behavior (sensitive dependence on initial conditions). We present methods to characterize the degree of chaos in a given network, and investigate how this information can be used to improve training. Our results indicate that training by simulated annealing can be improved by rejecting highly chaotic solutions encountered by the training algorithm, but is hindered if all chaotic solutions are rejected.

[*] yetkin@cs.wisc.edu

[†] sprott@juno.physics.wisc.edu, to whom all correspondence should be addressed

Note: software used to implement the experiments in this paper can be found at

http://sprott.physics.wisc.edu/chaos/yetkin/

1. **Introduction**

Neural networks can be used to generate time series of arbitrary length by feeding outputs of the network at time t into inputs of the network at time $t+1$. The fact that they are universal approximators (Hornik, et. al 1990), combined with their trainability has rendered artificial neural networks attractive as models of dynamical systems (i.e., systems that change in time, such as the stock market). Neural networks can be trained to replicate time series without prior knowledge of the system that produced the dynamical behavior. For example, Gershenfeld (1993) reports a neural network that can predict the time series representing the fluctuations in the output of a far-infrared laser (described by three coupled nonlinear ordinary differential equations). Neural networks can also reproduce the dynamics of chaotic systems such as the Hénon map (Gençay and Dechert, 1992) trained on a small number of time steps.

Some of the dynamical systems modeled by these networks (as well as the networks themselves) are capable of exhibiting chaotic behavior. (Chaos is usually defined as sensitive dependence on initial conditions. See Banks (1992) and Devaney (1989) for a more complete definition.) This has raised the hope of simulating the dynamics of systems in nature that are chaotic without much knowledge about the system. The fact that a network can exhibit chaos also raises concerns about the reliability of long term predictions from such networks. We were interested in quantifying the chaos in a given network, as well as determining the percentage of networks (with randomly chosen connection strengths) that are expected to exhibit chaotic behavior as a function of their size (and certain other parameters). The sensitivity of the network to initial conditions was quantified by numerically calculating the largest Lyapunov exponent (LE) (Frederickson, et. al., 1983).

After estimating the fraction of randomly connected networks (in a certain region of parameter space) that can be expected to exhibit chaos, we have investigated the relationship of the Lyapunov exponent of a network to how well it can be trained to replicate a time series. We have also investigated how the LE of a network affects its performance for prediction of data beyond the training set.

We conclude that long-term prediction of data suffers from certain theoretical as well as practical problems. We have also discovered that certain constraints placed on the Lyapunov exponent of the network during training can improve the performance of the training algorithm. Specifically, simulated annealing (Press et al., 1993) tends to converge to the desired solution faster if it is allowed to explore regions in weight space that yield networks with slightly positive Lyapunov exponents. Suppressing chaos altogether severely hinders performance even when training on periodic data—a small amount of chaos appears to be necessary for the algorithm to perform well.

2. Network Architecture

The artificial neural network we used for this experiment consists of N neurons that use the hyperbolic tangent as the squashing function. This network is fully connected, which means that each neuron is connected to every other neuron in the network, including itself. The connection weights of this network are represented by a weight matrix $\mathbf{W}$. A global scaling parameter, s, is also multiplied by the weights. The matrix $\mathbf{W}$, the scaling parameter s, and the network size N constitute our parameter space. This is a very general architecture of which feedforward networks (which are generally used in

engineering applications) are a subset. The "outputs" of each neuron are represented by a vector $\mathbf{x} \in \mathbf{R}^N$.

More formally, the recurrent neural network is defined as follows:

Let $\mathbf{W}$ be an NxM matrix of connection weights between neurons (initially assigned Gaussian random numbers with zero mean and unit variance) where N is the number of neurons and M is the number of inputs to each neuron. For a fully connected network (and for all the networks in this paper), M equals N. We can make this simplification because an N x N square matrix can be used to represent any artificial neural network of this type containing N neurons. Let s be a real, positive scaling parameter (usually set to 1) that acts as the standard deviation of the elements of $\mathbf{W}$.

Let $\mathbf{x}$ be an NxN matrix representing N outputs over N time steps such that $\mathbf{x}_{i,t}$ is the output value of the ith neuron at time step t. (While this matrix could contain any number of time steps, we have chosen N steps in order to arrive at a unique solution.)

The output of a neuron i at time t+1 (i.e., $\mathbf{x}_{i,t+1}$) is dependent on the values of all neurons at time t and the weight matrix $\mathbf{W}$ in the following manner:

$$\mathbf{x}_{i,t+1}=\tanh\left(\sum_{j=1}^{M} s\mathbf{W}_{i,j}\mathbf{x}_{j,t} / \sqrt{M}\right) \qquad (1)$$

The weights are normalized by the square root of the number of connections (M) so that the mean square input to the neurons is roughly independent of M.

Using the outputs of the network as its inputs for the next time step makes the system dynamical. A dynamical system is one defined in the form $X_{t+1}=F(X_t)$ where X ? $\mathbf{R}^N$ and F is differentiable at least once. Each iteration of equation (1) constitutes a time step. This network can now be trained to generate a desired time series. The system is also capable of exhibiting chaos (due, in part, to the nonlinearity of the hyperbolic tangent function).

3.1 Quantification of Chaos

It is possible to determine whether the aforementioned system is chaotic by calculating its largest Lyapunov exponent (LE) (Frederickson, et. al., 1983) for a specific set of parameter values. The LE is a measure of sensitivity to initial conditions; a positive LE implies chaotic behavior.

The LE for a network with weight matrix **W** was calculated numerically as follows (described in Dechert, et. al., 1999):

An initial point (y_0,…, y_{d-1}) (representing the outputs of the network in a given time step) and a nearby point (v_0, …,v_{d-1}) were chosen randomly.

Define

$\Delta y_t = (y_t - v_t, \ldots, y_{t+d-1} - v_{t+d-1})$

and let the initial separation of the two points $|\Delta y_0| = \varepsilon << 1$. Both points were advanced by one time step and the scaling ratio of $|\Delta y_1|/|\Delta y_0|$ was recorded. The vector Δy_1 is then rescaled to length ε, and the new neighbor, $(v_1,\ldots,v_d) = (y_1,\ldots,y_d) + \Delta y_1$, was advanced one time period along with $(y_1,\ldots,y_d)$. This process is then repeated, and the largest Lyapunov exponent was estimated by the average of the logs of the scalings:

$$l = t^{-1} \sum_{s=0}^{t-1} \ln\left(\frac{|\Delta y_{s+1}|}{|\Delta y_s|} \right)$$

where t is the number of iterations of the map. We used a t of approximately 200 and an ε of 10^{-8} to obtain 2-digit accuracy in l.

3.2 Probability of Chaos

Albers, et. al. (1996) report that untrained, feedforward networks with randomly chosen weights have a probability of chaos that approaches unity as the number of neurons and connections are increased. We have extended this work by testing the probability of chaos in fully connected recurrent networks.

The network size (*N*) and the scaling parameter (*s*) affect the frequency of chaos and the average Lyapunov exponent. The effect of these parameters on the average LE (and hence the percentage of chaos) was determined by using 500 different networks with randomly chosen weights. Figure 1 shows the effects of the scaling parameter on two different sized networks.

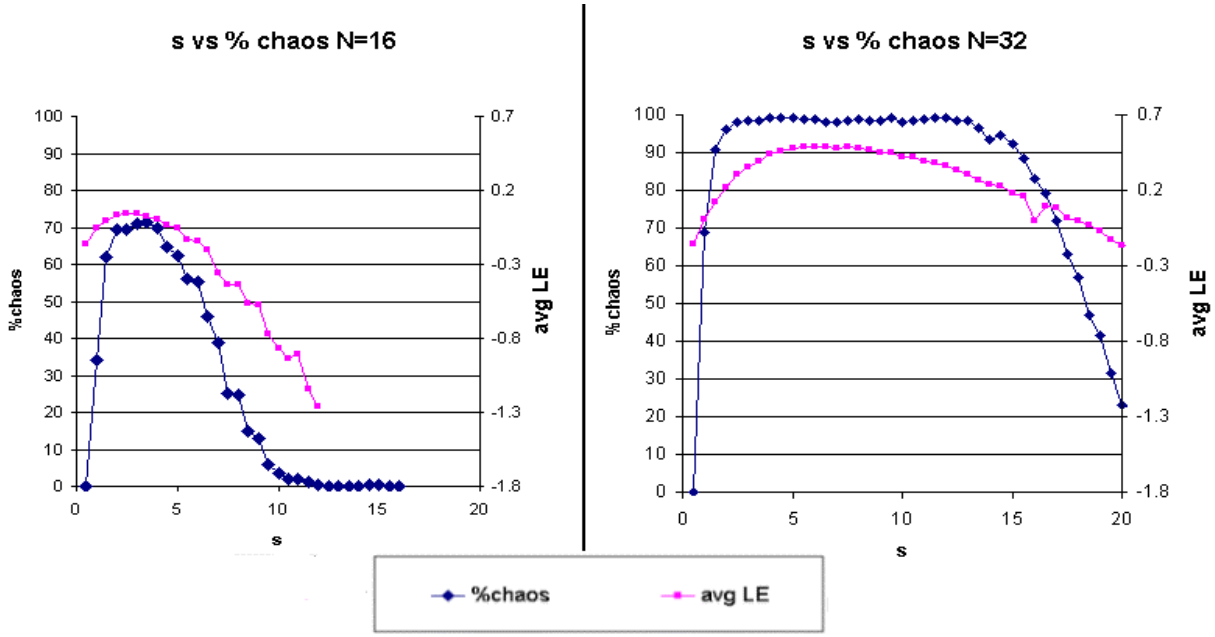

Figure 1: Percent chaos and average LE vs. *s*. 1a) *N*=16. 1b) *N*=32. Note that the larger network is 100% chaotic for a wide range of *s*.

As *N* gets larger, the range of *s* for which the networks are 100% chaotic also increases. This behavior is similar to the one observed by Albers and Sprott (1996) on feedforward networks with only one feedback, except that the transition to 100% chaos with respect to the parameters *s* and *N* is more abrupt for the fully connected recurrent networks studied here. The decrease in percent chaos as the scaling parameter *s* gets larger (observed in

figure 1) is due to the saturation of the network (all outputs are driven to either 1 or –1, which constitute upper and lower bounds for the range of the hyperbolic tangent function). In addition, none of the networks are chaotic when *s* is small enough. This is because non-linearity is a necessary condition for chaos, and the behavior of the hyperbolic tangent function is linear when the input is around zero.

These findings indicate that, for a large recurrent network (with *s* between 1 and 10, as is usually the case), chaos is the rule rather than the exception. This result has important consequences for time series prediction as well as training. While the training algorithm alters the LE of an individual network, the average LE of a large number of networks that are trained agrees with the values given in Figure 1, and is not affected significantly by the nature of the training set. This is important because it means that the parameters *N* and *s*, rather than the nature of the training set, determines the behavior of the network. Networks optimized for short-term prediction rarely replicate the long-term dynamics of the training set.

4. Training

We have used a variant of the simulated annealing algorithm (Press, et. al., 1993) to train our networks. Simulated annealing is a very general training algorithm that minimizes an error function. For any given training set, a function $e(\mathbf{W})$ can be defined which measures the root mean squared error between the training set and the time series generated by the network for a given choice of $\mathbf{W}$. Simulated annealing perturbs the matrix $\mathbf{W}$ with random numbers scaled by a factor called temperature (T). This "temperature" is gradually reduced until a lower error (i.e., a better solution) is found. When a better solution is found, and the temperature is less than one, it is doubled to

explore a larger region of weight space. The process is roughly analogous to slowly cooling down a metal in order to get the molecules to arrange themselves in a low energy configuration, or shaking a jar of sugar to reduce its volume. The error minimum in simulated annealing corresponds to the minimum energy state in the physical process.

Our version of the simulated annealing algorithm was implemented as follows:

All weights of the connections between neurons (in the matrix **W**) were initialized to zero. These weights were perturbed by multiplying a Gaussian random number with zero mean and variance *T* (initially chosen to be 1) divided by the square root of *M*, the number of connections per neuron. An error term *e* was initialized to a very high value.

The training set for periodic time series consisted of *N* identical sine waves (one per neuron) sampled at *N* time points with different initial phases, simulating a traveling wave. The sample chaotic time series we used was generated by *N* logistic maps in a similar fashion. The logistic map is a very simple chaotic system given by $X_{t+1}=4X_t(1-X_t)$ (May, 1976).

Using a sine wave in the above fashion (one per neuron) results in a training set that is an *NxN* matrix, **y**, with each column representing a sine wave. To calculate the error in training, the time series generated by the neurons in the system are also placed in an *N*x*N* matrix, **x**. The RMS distance between these matrices yields the error term *e* and can be calculated as:

$$e = \frac{1}{N}[\sum_{i=1}^{N}\sum_{t=1}^{N}(y_{i,t} - x_{i,t})^2]^{1/2} \qquad (3)$$

At each iteration, the weight matrix **W** of the network is perturbed by Gaussian random numbers with variance $T/\sqrt{M}$.

The network is then allowed to generate a time series consisting of *NxN* points (*N* time points per neuron), and the error is compared to the best error known so far. If the new error is less, it is assigned to *e* and the new weights are kept, and *T* is doubled (if less than 1). If not, *T* is decreased by a cooldown factor *c* and the procedure is repeated (we used c=0.995). The cooldown factor, *c*, determines how fast the algorithm "shrinks" the region of space it is exploring. Assigning *c* a value that is close to one will cause the algorithm to explore the space more thoroughly at the expense of computation time. As *c* gets smaller, the algorithm becomes faster but more likely to become "stuck" in a local minimum.

5. Relation of Chaos to Training

To determine the relationship between the chaos that the network is capable of exhibiting and the training algorithm, we calculated the LE of the network at each “improvement”. This allowed the characterization of the path taken through LE space by various instances of the training algorithm. Figure 2 shows three typical paths. The algorithm performed acceptably only when the LE of the network remained near zero (path B in figure 2).

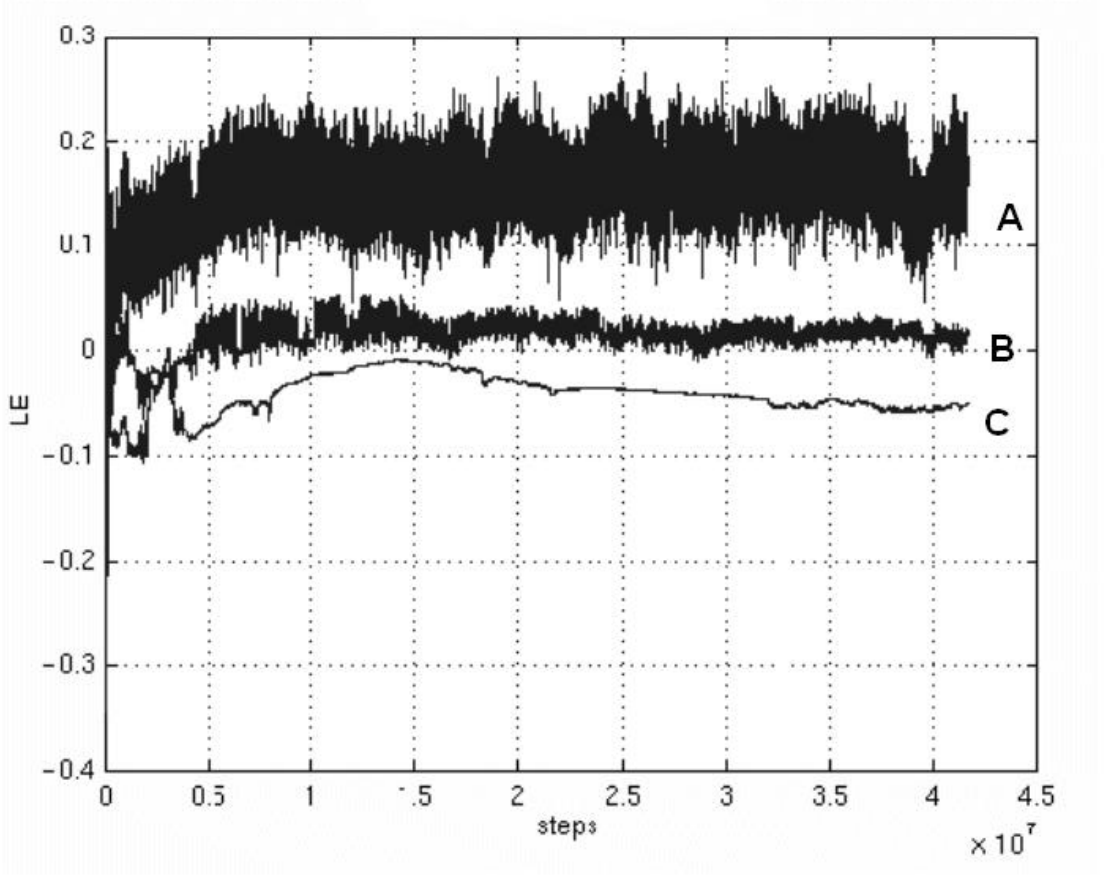

Figure 2: LE vs. steps during training on 16 points of 16 sine waves for three typical instances of the algorithm. Instance B, whose LE is slightly positive on the average, performed well. The others (A and C) converged to unacceptable errors. The variance of the LEs is higher in the positive region—this is because perturbing a network with a highly positive LE has a greater effect (on the LE of the new network obtained) than perturbing a network with a lower LE by the same amount.

There are certain networks for which the error converges much faster (using the fewest number of CPU cycles) than average. Examination of LE vs. time graphs of several networks (such as those in figure 2) revealed that the networks that converge the slowest also go through regions with highly positive LEs implying, as one would expect, that too much chaos hinders training. However, constraining the LE to be negative (i.e., rejecting all solutions with positive LEs during training) does not yield better performance. In fact, discarding all chaotic networks in such a manner actually decreases the average performance. After some experimentation, we found that discarding highly chaotic

networks encountered by the algorithm while still allowing the LEs to be slightly positive causes the error to converge faster and increases training performance (see figure 3).

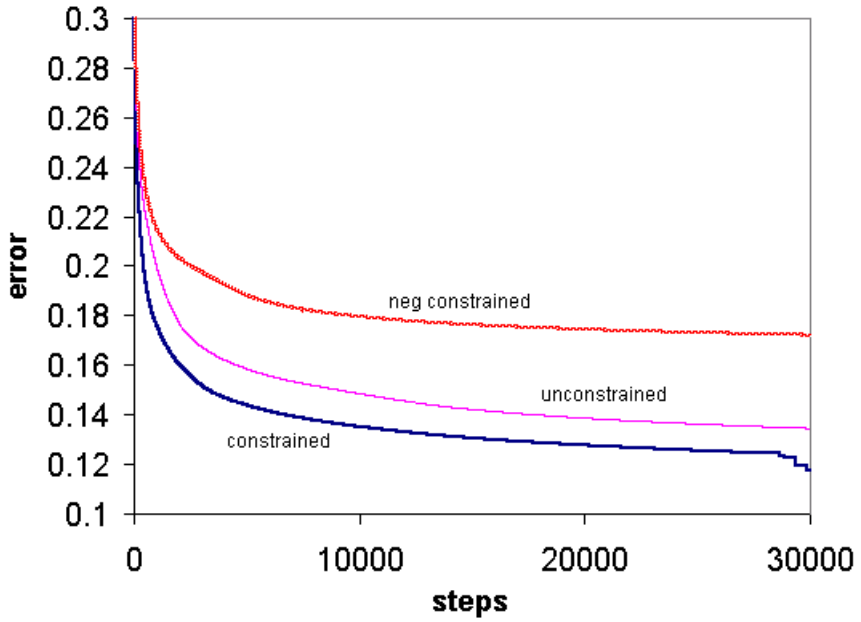

Figure 3: Networks constrained to have an LE between [-1, 0.005] perform better than unconstrained networks or ones constrained to have only negative LEs (30 cases, averaged).

Figure 3 illustrates the result of the improvement we propose on simulated annealing. The constraint works because we know that networks with highly positive LEs will not give the desired dynamics. It is interesting, however, that a small amount of chaos appears necessary for the algorithm to perform acceptably—constraining the LE to be at or below zero yielded significantly decreased performance compared to even the unconstrained training runs.

The sine wave experiment was repeated by replacing the 16 out-of-phase sine wave values in the training set with values from the logistic map ($X_{t+1}=4X_t(1-X_t)$) and the Hénon map. The error does not converge as fast, and the long-term dynamics (or the LE) are not reproduced in these cases. Even if the training set is well matched, the long-term behavior of the network is better indicated by the calculated LE of the network.

6. Dynamics of Trained Networks

While networks produced by the training algorithm that have positive LEs may perform well for the duration of the training set, they invariably do poorly on any prediction beyond it, as illustrated by Figure 4.

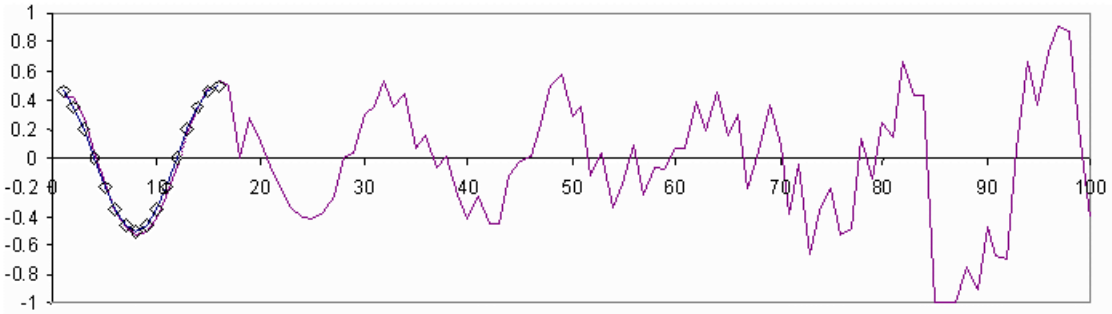

Figure 4: Chaotic net predicting beyond the first16 points of a sine wave in the training set

Networks trained on periodic systems (such as the sine wave) have LEs that approach zero as the error decreases, as one would expect for a slowly varying periodic solution. If a network is trained on time series generated by a chaotic system (such as the system of logistic maps), both the error and the LE converge more slowly. It is possible to find periodic or chaotic solutions that replicate the training set, but the nature of the behavior after that is highly variable. This is a fundamental problem with fitting a potentially chaotic system to replicate time series. Chaotic systems can exhibit a phenomenon called intermittency where the observed behavior is apparently periodic for a long time (compared to the typical time period of the system) and abruptly switches to chaotic behavior (Hilborn, 1994). It is therefore possible for a training algorithm to find such a region of apparent periodicity that matches the training set, even when the dynamics of

the system is actually chaotic. Increasing the number of steps used in the calculation of the LE will help reduce the chances of mistaking a chaotic network for a periodic one.

It must be noted, however, that a negative LE only guarantees a periodic solution, but not necessarily the solution in the training set. Even if the first *N* points generated by the network match the training set perfectly, the long-term behavior may still be different. Figure 5 shows some examples of this. The time course generated by the network may attract to a fixed point (not shown) or cyclic orbits with very different periods.

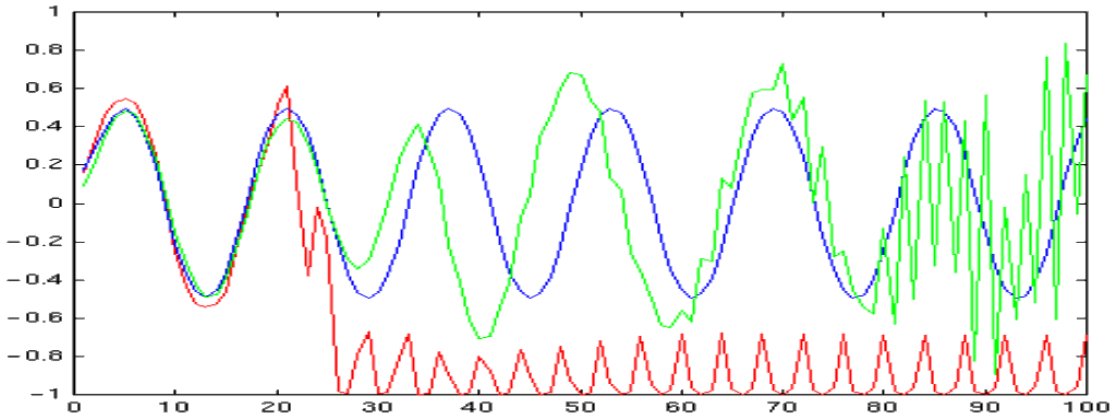

Figure 5: Data generated by three different networks with negative LEs trained on the same periodic waveforms.

Networks trained on chaotic systems (such as the logistic map) generate data that appear chaotic, but do not necessarily resemble the dynamics or the LE of the original system after a long time. The error also converges more slowly.

The percent chaos data for networks with random weights may also be of practical importance in knowing which parameters to choose when a chaotic or non-chaotic network is desired. As mentioned before, networks in which the scaling parameter (*s*) is

between 1 and 10 and the size is larger than 32 are almost always chaotic, regardless of the training set. The networks can also be forced into periodicity by decreasing the *s* parameter. Similarly, periodic networks can often be forced into chaos by increasing the *s* parameter.

7. Conclusion:

As can be seen from our examples, prediction of data past the end of the training set is difficult. There are three reasons for this: First, chaos implies sensitive dependence on initial conditions. If the system we are trying to predict is chaotic, the prediction is guaranteed to fail at some point in the future (unless the initial conditions are matched exactly), even if the dynamics of the system can be matched. Second, the LE of the network to which the training algorithm converges may be spurious. Finally, the training set may not contain enough data to reveal the nature of the underlying system—there may be many networks that accurately replicate the training set, but behave differently after that. Intermittency may also produce a system that has a low error for the given time series.

It is not surprising that eliminating networks with extremely positive LEs during training to periodic data saves time by not exploring solutions that are far from optimal. These highly chaotic nets are unlikely to match the LE of the system even when generating points that are in the training set. What is somewhat surprising, however, is that excluding all networks with positive LEs (even when training on a periodic solution) reduces performance. Going through regions with slightly positive LEs causes the training algorithm to explore different solutions faster.

Biological neural networks might be exploiting this phenomenon as well, since they are usually large enough to be 100% chaotic, but are unlikely to have high LEs, since that would destroy their long-term memory. Further studies could look for chaos in networks that model well-known biological systems, such as the worm C. *Elegans*, whose nervous system architecture has been completely mapped (Achacoso, 1992).

References

Achacoso, Theodore B., Yamamoto, William S. (1992), AY's neuroanatomy of C. elegans for computation. Boca Raton : CRC Press.

Albers, D.J., Sprott, J.C., and Dechert, W.D. (1996), Dynamical behavior of artificial neural networks with random weights. In Intelligent Engineering Systems Through Artificial Neural Networks, Volume 6, ed. C. H. Dagli, M. Akay, C.L.P. Chen, B.R. Fernandez

Banks, J., Brooks, J., Cairns, G., Davis, G., and Stacy, P. (1992), "On Devaney's definition of chaos," Amer. Math. Monthly 99; 332-334

Dechert, W. Davis, Sprott, Julien C., Albers, David J., (1999), "On the Probability of Chaos in Large Dynamical Systems: A Monte Carlo Study," Journal of Economic Dynamics and Control. (in press)

Devaney, R. L. (1989), An Introduction to Chaotic Dynamical Systems, (Addison-Wesley, Redwood City)

Frederickson, P., Kaplan, J.L., Yorke, E.D., Yorke, J.A., (1983), "The Lyapunov Dimension of Strange Attractors," Journal of Differential Equations, Vol. 49, pp. 185-207

Gencay, R. and Dechert W.D. (1992), An Algorithm for the *n* Lyapunov exponents of an *n*-dimensional unknown dynamical system, Physica D 59 142-157.

Gencay, R. and Dechert W.D. (1996), The topological invariance of Lyapunov exponents in embedded dynamics, Physica D 90 40-55.

Gershenfeld, Neil A., Weigend, Andreas S (1993), "The Future of Time Series." In: *Time Series Prediction: Forecasting the Future and Understanding the Past,* A.S. Weigend and N.A. Gershenfeld, eds., 1-70 Addison-Wesley.

Hilborn, Robert C. (1994), Chaos and Nonlinear Dynamics--An Introduction for Scientists and Engineers. Oxford University Presss, New York. pp296

Hornik, K, Stinchcombe, M., White, H (1990), "Universal Approximation of an Unknown Mapping and its Derivatives Using Multilayer Feedforward Networks," Neural Networks, Vol. 3., pp., 551-560

May, R. (1976), "Simple Mathematical Models with very Complicated Dynamics, Nature 261, pp. 45-47.

Seidl, D. and Lorenz, D. (1991), A structure by which a recurrent neural network can approximate a nonlinear dynamic system. In Proceedings of the International Joint Conference on Neural Networks, volume 2, pp. 709-714.

Press, W. H., Teukolsky, S. A., Vetterling W. T., and Flannery, B. P. (1993), Numerical Recipes in C: The Art of Scientific Computing. Cambridge University Press, Cambridge. pp. 436.

Acknowledgements. We thank W. D. Dechert and D.J. Albers for their discussions and help in revising, as well as the University of Wisconsin Condor group for access to their high throughput distributed batch processing system.