Phase-Space learning for recurrent networks

Fu-Sheng Tsung and Garrison W Cottrell*
Department of Computer Science & Engineering
and
Institute for Neural Computation
University of California, San Diego, USA

February 17, 1993

Abstract

We study the problem of learning nonstatic attractors in recurrent networks. With concepts from dynamical systems theory, we show that this problem can be reduced to three sub-problems, (a) that of embedding the temporal trajectory in phase space, (b) approximating the local vector field, and (c) function approximation using feedforward networks. This general framework overcomes problems with traditional methods by providing more appropriate error gradients and enforcing stability explicitly. We describe an online version of our method we call ARTISTE, that can learn periodic attractors without teach-forcing.

1 Introduction

We study the issue of learning attractors with recurrent networks. In particular, we are interested in learning nonstatic attractors, e.g., periodic and chaotic attractors. In this paper, we view the problem of training recurrent networks from the *phase space* of the trajectory being learned, and show that it can offer valuable insights. A brief background on the phase space terminology is provided in section 2. Then, using this point of view, we discuss three existing training methods in section 3: Real Time Recurrent Learning (RTRL) [1], back propagation through time (BPTT) [2][3] and prediction training [4, 5] of iterated maps. All current methods for learning nonstatic attractors require some form of *teacher-forcing*, where the network's internal state is reset to lie upon the trajectory being learned. This analysis makes clear at least one reason why teacher-forcing is necessary to train recurrent networks, why some of these methods are only approximations to the correct approach, and why the hidden units of the first two approaches are not as well informed as in the third approach. Also, for the task of training attractors, these methods do not explicit insure stability. In

*This work was supported by NIH grant R01 MH46899-01A3. Thanks for comments from Steve Biafore, Kenji Doya, Peter Rowat, Bill Hart, and especially Dave DeMers for his timely assistance with simulations.

section 4, we show how ideas from prediction training and the phase space framework give rise to general learning methods that can amend these problem. These methods use training *within the phase space*, thereby reducing the problem of training a recurrent network to training a feed forward one. In section 5, we describe a specific implementation of phase space learning, where the network is allowed to explore the phase space on its own. Using this algorithm, the network is able to learn periodic attractors without teacher-forcing. We summarize our findings in section 6.

2 Background

Recurrent neural networks are dynamical systems; they evolve in time. Most recurrent networks in use today are discrete dynamical systems, that is, they are *iterated maps* represented by the system of equations:

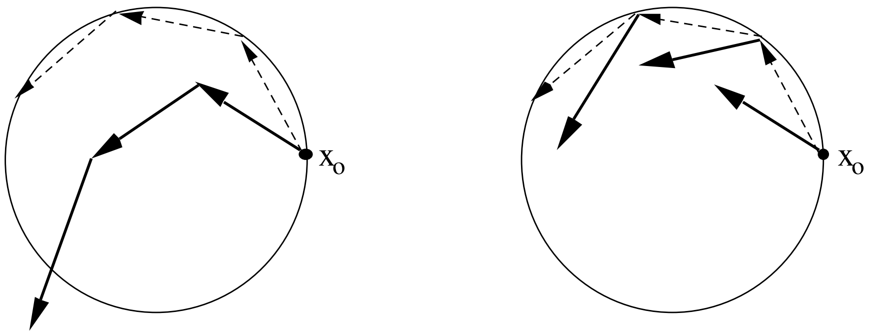
$$y_1(t+1) = f_1(y(t))$$
$$\dots$$
$$y_n(t+1) = f_n(y(t))$$

For most problems where we want the behavior of the network to match a smooth trajectory $x(t)$, we can consider this discrete system to be an approximation to the continuous system:

$$\dot{y}_i = -y_i + f_i(y)$$

The trajectory $x(t)$ is itself generated by a dynamical system $X(t)$ of dimension m. The phase space of $X(t)$ is the m-dimensional space, without the time-dimension. A specific trajectory $x(t)$ of $X(t)$ traces out a curve in the phase space. Thus time is implicit in this representation as a parameterization of the curve. If a dynamical system converges to a fixed point, it traces out a curve that eventually terminates. If a system is periodic, it traces out a closed loop. A periodic attractor in phase space is a *stable limit cycle*, which means nearby curves (other trajectories) converge to the closed curve. The vector field of the dynamical system is the point-wise derivative with respect to time in this space of the system $X(t)$. As such, the vector field specifies how each point in the phase space of the dynamical system evolves.

For the recurrent network (of n units) to learn the trajectory $x(t)$, n must be equal to or greater than m. The problem, then, may be stated as minimizing a cost function between the trajectories $x(t)$ in the phase space of $X(t)$, and the trajectories of the subset of the network of dimension m (which we will term the *visible units*) that are supposed to match the behavior of the target system. That is, we can look at the process of training a recurrent network from the point of view of matching dynamical systems in phase space.



a b

Figure 1: Learning a pair of sine waves with RTRL learning. a) without teacher forcing, the network dynamics take it far from where the teacher might be informative. The gradient information is then unhelpful, or even wrong. b) With teacher forcing, the network's visible units are returned to the trajectory.

3 Analysis of current approaches

There are three current methods for training a recurrent network to fit a dynamical system. With the phase space point of view, and the point of view of unrolling the networks in time, certain problems of these methods become evident. Insights gained from these considerations will lead to our new algorithm in the next section.

3.1 Real Time Recurrent Learning

The first training method we consider is the RTRL algorithm of [1]. This is a forward-gradient algorithm that keeps a matrix of partial derivatives of the network values with respect to every weight. To train a periodic trajectory, it is necessary to *teacher-force* the visible units [1, 6]. In its basic form, teacher-forcing means that on every iteration of the network, after the gradient has been calculated from the error on the visible units, the current values of the visible units are replaced by the correct values as provided by the teacher. One reason why teacher-forcing is necessary for learning periodic signals is illustrated in Figure 1. We consider learning a pair of sine waves offset by 90°. In phase space, this becomes a circle (Figure 1) (a). Initially the network (the thick arrows) is at position x_0 and has arbitrary dynamics. After a few iterations, it wanders far away from where the teacher (the dashed arrows) assumes it to be. The teacher then fails to become informative, because it is only telling the network to be at position x after a certain number of iterations, starting at x_0. The teacher completely disregards the network's current position. Teacher-forcing, illustrated in Figure 1 (b) resets the network back on the circle, where the teacher again provides useful information. The same reasoning applies to training chaotic attractors.

However, if the network has hidden units, then the phase space of the visible units is just a projection of the actual phase space of the network, and the teaching signal gives no information as to where the hidden units should be in this higher-dimensional phase space.

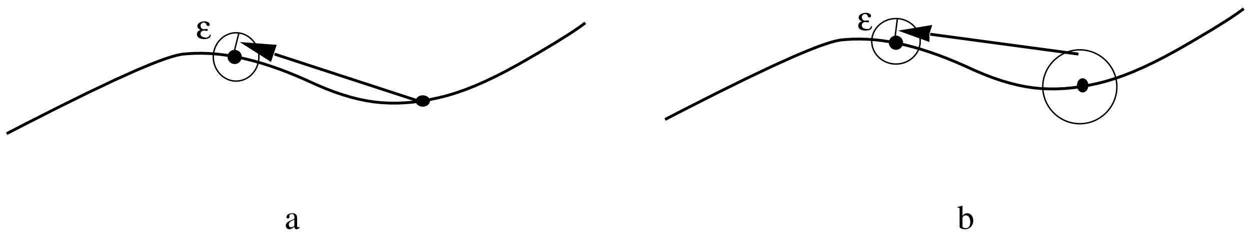


Figure 2: The paradox of attractor learning with teacher forcing. a) During learning, the network learns to move from the trajectory to a point near the trajectory. b) After learning, the network moves from nearby points towards the trajectory.

Hence the hidden units in standard teacher forcing retain their current values, which may be unrelated to where they should be. One may teacher-force the network for a while before learning resumes, to entrain the hidden units so that they are near where they should be. But this hopeful thinking at best, and there is no theoretical justification. Neither can this problem be overcome by using a smaller time step to maintain the hidden units in "nearby" regions, as the gradients then becomes too small for the system to learn [6]. (Discussions of the necessity of teacher-forcing, from other points of view, can be found in [1][7][8].)

An interesting issue concerning networks trained by RTRL and teacher-forcing is: Why are the learned trajectories stable? Experience suggests that the trajectories so learned are indeed strong limit cycle attractors [9]. But when one considers how the networks are trained, a paradox emerges. At each step, the networks are trained to go from a point on the trajectory, to a point within the ball defined by the error criterion ϵ (see Figure 2 (a)). However, after learning, the networks behave such that from a place near the trajectory, they head for the trajectory (Figure 2 (b)). Hence the paradox. It is unclear whether this is a general effect, or simply a manifestation of the fact that using digital computers, we would never consider a network that treats the trajectory as an unstable attractor to have learned. In either case, this is an unresolved question. A point to be taken from this, however, is that teacher forcing based on the trajectory does not directly inform the network that the trajectory *should be* an attractor.

3.2 Back propagation through time

A second standard method for training attractors is called back propagation through time (BPTT) [2] [3]. In this method, the network is duplicated for some number of iterations of the teaching signal, and gradients are computed "backwards through time". Hence the problem of training a recurrent network is reduced to the problem of training a feed-forward network, although constraints must be imposed to ensure that identified weights at different time steps remain equal. In Figure 3, we repeat the previous exercise of sine wave learning for BPTT. Here, we see that a weaker form of teacher forcing is used, where the network is only required to be in synch with the signal at the initial time step. Of course, teacher forcing may be used at more time steps with this method as well, and often is required to learn difficult trajectories.

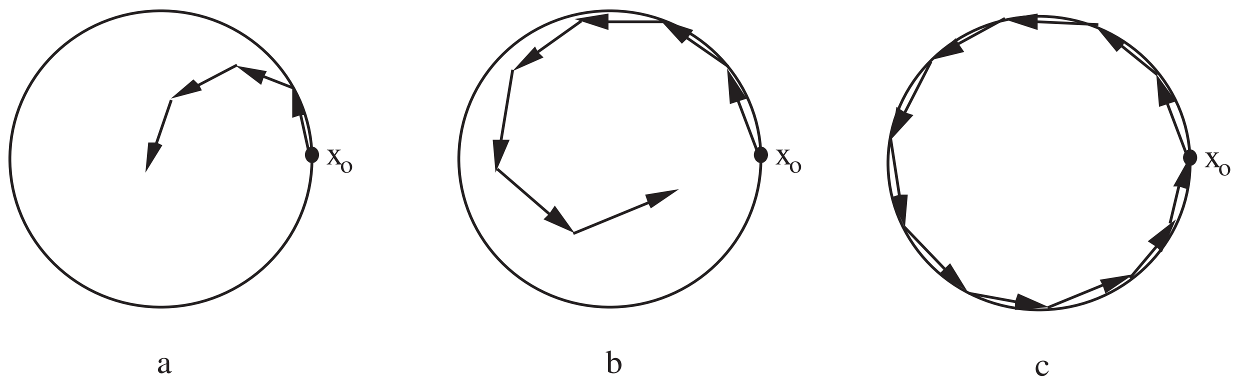


Figure 3: Learning a trajectory with BPTT. The network is essentially restarted on the trajectory many times. a) Early in learning, the behavior looks like RTRL without teacher forcing. b) midway through learning, the network is good early on, but loses its way later. c) At the end of learning, the network is able to follow the whole trajectory.

Considering the network in terms of the BPTT formalism reveals another "hidden unit abuse" problem. Suppose that the visible units' next state is a non-linearly separable function of their current state. Then the hidden units are necessary to achieve this. However, the hidden units are at the same level in the unrolled network as the visible units. This is illustrated in Figure 4. The hidden units in this scheme must compute the nonlinearly separable function *every other iteration.* This suggests that BPTT schemes will work better if the teaching signal is given every other iteration. Recent experience suggests this may be the case [10].

3.3 Prediction training

The third method we consider is prediction training [4] [5], which may be considered an interesting variant of BPTT. In the standard scheme, a feedforward network is given inputs that are previous points on the trajectory to be learned, delayed in time. The output is the next point on the trajectory. In essence, the inputs are again teacher-forcing the network. Once the network has learned, the network may be considered a recurrent network when used as an *iterated prediction network.* Here, the output is fed back to the input and the inputs are shifted left (see Figure 5. A thorough discussion of prediction network architecture can be found in [11]).

An issue with prediction networks is that the time window (the delayed inputs) parameters must be chosen by the experimenter. With the appearance of recurrent learning algorithms such as RTRL, many hoped that the network would learn to extract the relevant temporal structures from the teacher automatically. This may be true for the simplest trajectories (e.g., a sine wave), but by now it is well acknowledged in the recurrent network community that empirical results have not carried out this promise.

It is interesting now to consider the prediction network as a kind of BPTT network (Figure 5). There are two differences from "standard" BPTT. First, the network architecture

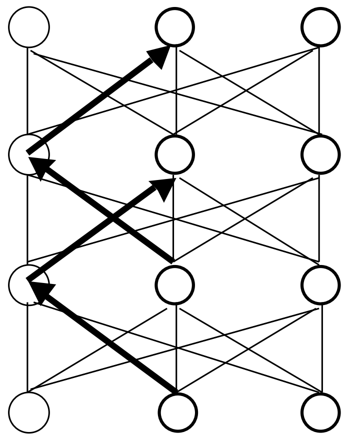


Figure 4: A nonlinearly separable problem as it looks to hidden units trained by BPTT. Here, the network is unrolled in time four steps. The heavy arrows show how the next state must be computed by the hidden units every other time step.

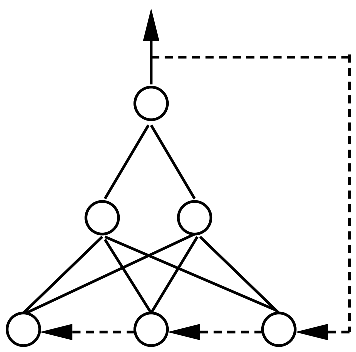


Figure 5: The network used for prediction training. Dashed connections are used after learning.

is extended by the visible units encoding a selected temporal history of the trajectory (via a time-delay window). More importantly, the hidden units are given their "proper" place *between* the visible units, allowing the network to produce nonlinearly separable transformations of the visible units in a single iteration. When this network is treated as a recurrent network, the delays on these connections can be considered to have delay $\frac{1}{2}$. That is, the hidden units of this recurrent network are "fast" units.

One can see that prediction training is generally more powerful than the first two methods because of the way the hidden units are treated, and because it gives the network more information about the trajectory. However, as with the RTRL and BPTT methods, there is the question of stability. Since the network has learned to predict one step, there is no guarantee that the network will remain on the trajectory by iterating its own prediction. This is not surprising, since in phase space, the prediction training method trains the network *on the desired trajectory only.* No information is given about the attractor structure *near* the trajectory.

3.4 Summary

We have pointed out several problems with these traditional training methods:

1. None of them inform the network that the trajectory is an attractor
2. Teacher forcing for RTRL and BPTT with hidden units is at best an approximation
3. Hidden units are not given proper information in RTRL or BPTT

On the other hand, prediction training does seem to make more expedient use of hidden units. In the next section, we use these insights to derive a new algorithm we call Phase Space learning.

4 Phase Space Learning

Our goal here is to develop an algorithm that 1) explicitly learns attractor behavior and 2) makes efficient use of hidden units. The method we describe is applicable to a general temporal trajectory, but for now let us assume it is periodic or chaotic, and that we want to learn it as an attractor.

Based on the phase space diagrams above, we suggest that instead of the temporal teacher used in RTRL and BPTT, a *spatial teacher*, based on the current position of the network in phase space, is a better alternative (Figure 6). Recall from section 2 that we cast the task of recurrent network training as matching two dynamical systems $X(t)$ (the teacher) and $Y(t)$ (the network). In phase space, this means the vector field of Y must learn to match that of the desired dynamical system X. Provided X is bounded, we normalize the phase space of X into a compact space $\mathbb{I}^m$ within the boundaries of the activation function of our units. Here m is the dimension of X. In the ideal case, we know the vector field of X completely; for example, the differential equations defining X is given. Then the vector field of X serves as the spatial teacher, showing where any point should go next. To approximate this with

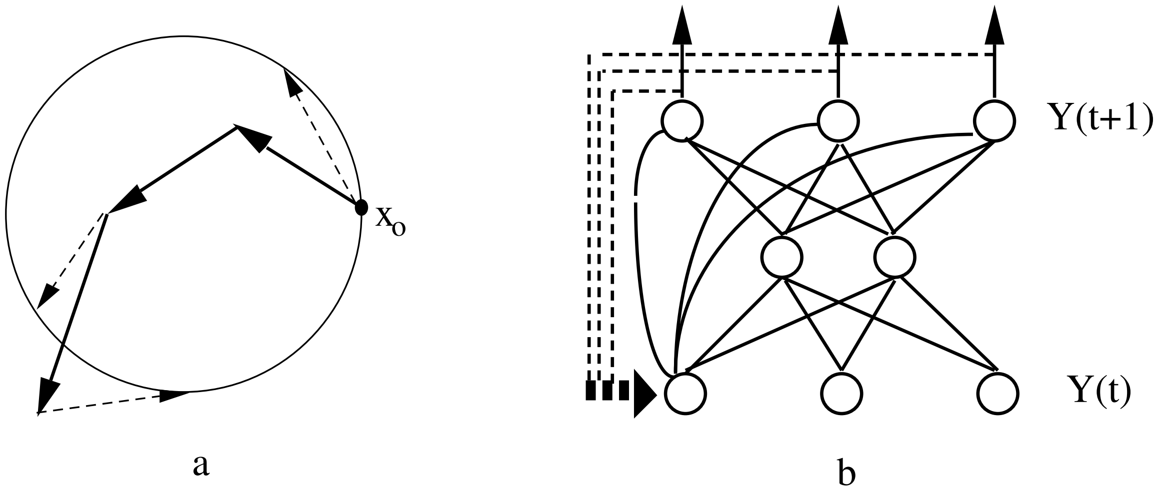


Figure 6: Phase space learning a) In phase space, the network is given a sample of the vector field as a training set. b) The network used for phase space learning. It is identical to the prediction network, except that the dimensionality of the input and output are the same, and the visible units are fully connected. Dashed connections are used after learning.

a discrete network, we choose an arbitrary step for the next iterate of the network. Since trajectories cannot cross themselves in phase space (if they did, the system would not be deterministic), the next iterate of each point is uniquely determined by its current position. Thus we obtain, from the vector field, a set of teacher data$\{(x_t, x_{t+1})\}$ where each (x_t, x_{t+1}) is a vector in $\mathbb{I}^m$, and x_{t+1} is the next iterate of x_t. This is a well-defined mapping

$$x_{t+1} = F(x_t)$$

from $\mathbb{I}^m$ to $\mathbb{I}^m$. F is a discrete approximation of X. Now the task of learning the dynamical system X becomes the task of learning the mapping F. This means the network will have m visible units, such that every point y_t, formed by the visible units of the network, gets mapped to $F(y_t)$. Function approximation is a feed forward problem, so a three layer feed forward network will suffice. Thus, the network Y consists of m inputs, m outputs, and an appropriate number of hidden units. Once the network has learned this mapping, the output values are fed back as input values just as in iterated prediction.

By the universal approximation theorems of feed forward nets (e.g., [12]), this architecture can approximate any mapping to any desired degree of accuracy. Thus, we seem to have arrived at a universal architecture for approximating any behavior describable as a dynamical system. However, as we will see in the simulations (section 5), having approximated the mapping well does not necessarily correspond to having approximated the dynamical system well.

In practical situations, we rarely know X completely. We are often only given a single trajectory. Furthermore, the signal given may be a lower-dimension projection of the m dimensional system. A figure-8 in two dimensions (or any trajectory that crosses itself) is an example. This is also common in time-series applications, where usually only one observable is available. In this case, standard phase-space reconstruction techniques can be used to embed it into the appropriate phase space [13, 14]. We now have teachers (x_t, x_{t+1}) on the single, desired trajectory in phase space, but do not know the entire vector field. That is, there are many possible vector fields (or mappings F) corresponding to the desired trajectory. If we can approximate the local vector field near the desired trajectory, we would obtain more data pairs as teachers to enforce the learning of the desired trajectory. Because of the feed forward property of the phase space, we can simply add this data to the training set.

For periodic attractors, the local phase space around the desired curve contracts towards points on the closed curve. Near most attractors, trajectories converge exponentially, thus we can artificially generate this data based on the given time series and the distance between the network and the attractor. For chaotic attractors, local approximation techniques such as those used in [15] are exactly what we need. This is an efficient algorithm that uses a linear polynomial to predict the next iterate based on nearest neighbors. However, any number of nonlinear optimization techniques may be used instead. When the network has learned the mapping satisfactorily, then it has in fact integrated all of these local approximators into a global model.

In summary, we have transformed the problem of learning attractors (or more generally, learning a temporal trajectory) in recurrent networks into the problem of learning the vector field mapping in phase space. Ideally we would have the complete vector field, which would specify the teacher for the network at every point. In practice, this is rarely so; usually, we only have the vector field on the desired trajectory. However, this is not a problem because the local vector near the desired curve can be approximated. The local vector is what we really care about because it serves to ensure the stability of the attractor explicitly. In fact, preliminary simulations suggest that using only training data near the attractor is preferable to a balanced sampling of the entire vector field, as the teachers far away from the attractor tend to complicate the learning task unnecessarily. The problems associated with hidden units in recurrent nets are countered, partially by increasing the number of visible units (when the given trajectory is a low-dimensional projection), and partially by using the prediction network architecture where the hidden units are given explicit teachers from the visible units.

As to the mechanics of phase space learning, there are several ways to specify the training sequence presented to the network. Since the training is feedforward, one can randomize the training set (consisting of data near the desired trajectory) and train the network in an epoch-wise fashion. This corresponds to, on average, training all local segments of the desired trajectory simultaneously. One may also present the data sequentially along the desired trajectory. This is like training with complete *spatial* teacher-forcing. RTRL and BPTT with teacher-forcing and no hidden units, and sequential prediction training both fall into this category, except the training is on the desired trajectory only.

Another possibility is to allow the network to move freely while its dynamics is modified by online weight changes. Wherever the network is, a teacher showing the local vector field is given as in Figure 6. In this case there is no predetermined training set; the local vector field is approximated on the spot. We call this version of the algorithm ARTISTE, for Autonomous Real TIme Selection of Training Examples. The advantage of this approach is that the training examples are tailored for the current dynamics of the network. ¿From training on the desired trajectory (100% spatial forcing), to autonomous training (0% spatial forcing), there is obviously a continuum of possibilities where the degree of spatial forcing is varied.

5 Simulation results

5.1 ARTISTE applied to learning sine waves

We describe a simple task trained using the ARTISTE algorithm, which allows the network to move freely during learning. This is the spatial, or phase space, analog of online training without teacher forcing. The task was to learn a pair of sine waves centered at (0.5, 0.5), with radius 0.4, as the desired limit cycle attractor. The network is to track one period of the sine wave in 30 iterations. In this case, we assume we have the equations of the sine waves and generate new examples on the fly.

We used a feedforward network as described above without hidden units for this task, as we know that two units can oscillate in this manner. We approximate the local vector field by calculating the current distance r from the current position of the network to the limit cycle (in phase space, of course), and set the next position to be rotated $2\pi/30$ radians and $a * r$ away from the limit cycle (here $a < 1$ is the constant convergence rate, which corresponds to exponential convergence).

The network is initialized with small random weights and started at an arbitrary initial position *not* on the desired trajectory. We used $a = 1/2$, a learning rate of 0.5, and momentum of 0.5. Figure 7 shows the evolution of the network at different stages. Early in learning, the network and the teacher entrain each other into a slow circular trajectory inwards (Figure 7 (a)). Most initial random weights result in fixed point behavior. Thus the network tends to move inwards, despite the teacher pulling it outwards. However, the teacher provides a repelling force at the fixed point. After 2000 iterations (Figure 7 (b)), the network undergoes a Hopf bifurcation and starts to spiral outwards. At first the spiraling is small, corresponding to small period oscillations. The period (and radius) of the oscillation quickly increases (Figure 7 (c)) to match that of the teacher's. The last stage of the learning makes fine adjustments to match the shape and position of the teacher. The final learned limit cycle is shown in Figure 7 (d), and its wave form in time is shown in Figure 7(e). It takes about 4000 iterations to reach this stage. The averaged absolute error per unit for the process is plotted in Figure 7 (f). The peak corresponds to the bifurcation. Repeated simulations always give very similar qualitative and quantitative results.

We have compared this method with direct training, using RTRL learning with teacher forcing. RTRL on this task takes around 1200 iterations while ARTISTE takes 4000 iterations. The RTRL network also undergoes a Hopf bifurcation in order to oscillate (as it must). However, as teacher-forcing moves the network rapidly around the cycle, it de-bifurcates back to being a fixed point and re-bifurcates over and over (Figure 8). Every round of bifurcations produces a structurally more robust limit cycle, until the limit cycle is maintained throughout the learning cycle [16]. This is where we usually stop the training. We call this phenomenon "Hopf hopping", and it has also been observed when hidden units

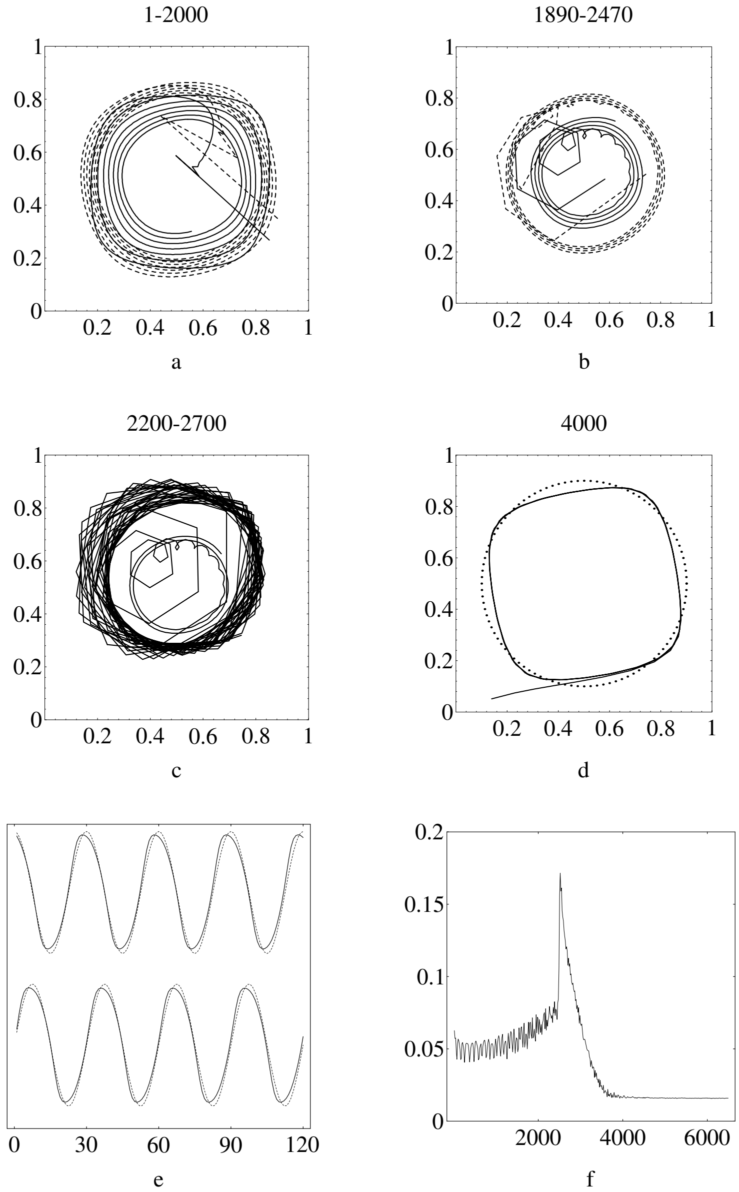


Figure 7: The behavior of a network in phase space while learning a sinewave. a) The network (solid curve) is started a the lower right hand corner. Very quickly it and the teacher (dotted line) entrains each other into a circular trajectory, with the network spiraling inward towards its a stable fixed point, despite the teacher pulling it outward. b) As the network approaches the center (around 2000 iterations), the bigger error forces the network through a rapid sucession of period-increasing bifurcations. c) When the period is approximately correct, the network starts to fine tune its trajectory. d) The final stable limit cycle learned by the network, compared with the teacher. e) The same trajectories in time. f) The maximum absolute error during the learning process. The peak corresponds to the onset of the bifurcations. The error stabilizes at 4000 iterations, and further training produces essentially no differences.

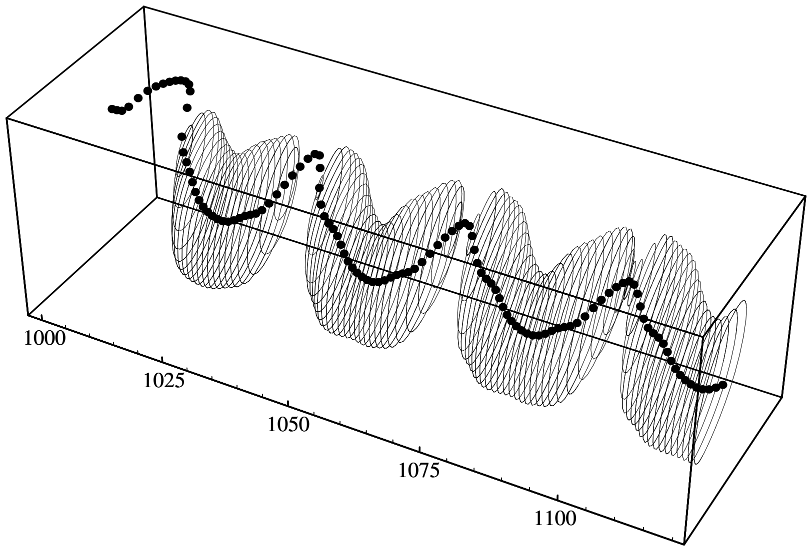


Figure 8: The phase portrait of a network using RTRL learning on the sine wave task. The appearance of the circles signals the Hopf bifurcation, with the fixed point becoming unstable. Then the limit cycle collapses back to a stable fixed point. This process repeats several times before the limit cycle is stable across the whole cycle of the teaching signal.

are used. One pitfall with Hopf hopping is that it is difficult to detect when the network has learned the desired oscillation. One may stop the training at a point between oscillation phases, and be misled into thinking that much more training is required. In fact, the network may be just a few iterations away from robust oscillations. We know of no suitable criteria to detect when the right time to stop is. In contrast, in phase space learning the network moves in small steps around the cycle according to its internal dynamics, and bifurcates only once. Once it has bifurcated there are no conflicting teachers to move it back to the fixed point stage. The error then decreases monotonically and quickly flattens out.

Another problem with traditional methods is overtraining. We have found that continued training (by RTRL) will cause the wave form to deform to the point where it no longer resembles a sine wave. The network is overfitting the 30 points used for training. In contrast, because we generate training points on the fly with ARTISTE, the training set is essentially infinite. We have found that continued training for several thousand iterations does not change the solution.

5.2 ARTISTE applied to the van der Pol Oscillator

We have also applied phase space learning on a more complicated example, the van Der Pol oscillator. The van Der Pol equation is defined by:

$$\begin{aligned} \dot{x} &= y \\ \dot{y} &= -\alpha(x^2 - b)y - \omega^2 x \end{aligned}$$

We used the values 0.7, 1, 1 for the parameters α, b, and ω, for which there is a global periodic attractor (Figure 10). We used a step size of 0.1, which discretizes the trajectory into 70 points. That is, the next iterate is generated by:

$$x_{t+1} = x_t + 0.1\dot{x}$$
$$y_{t+1} = y_t + 0.1\dot{y}$$

The network therefore has two visible units. We used two hidden layers with 4 units each, so that the unrolled, feedforward network has a 2-4-4-2 architecture. We set the learning rate and momentum to 0.5, and again started the network at a random point.

The ARTISTE algorithm applied to this task was quite interesting. At first, the network always converged quickly to the unstable fixed point inside the periodic attractor (there is always an unstable fixed point inside a stable limit cycle, unless there is another limit cycle inside). This is because the usual MSE error is equivalent to measuring the distance between where the network should be (X_{t+1}) and where it ends up (Y_{t+1}). However, the vector field of any dynamical system vanishes as one approaches a fixed point. Therefore, by moving the network's stable fixed point towards the vector field's unstable fixed point, the network actually achieves global minimum in the error!

Clearly, MSE is not a sufficient error function when we are using the true vector field as the teacher. Note that this problem did not exist in the sine wave example, because there we created an artificial vector field such that the unstable fixed point in the center of the circle has a large error value. A somewhat more elegant solution is to notice that MSE does not make use of the *angle* θ between the teacher vector ($dX = (Y_t, X_{t+1})$) and where the network actually goes ($dY = (Y_t, Y_{t+1})$). An error measure based on the angle is independent of the the magnitude of the vector field. In particular, as long as the network moves towards the unstable fixed point of the vector field, θ will be large, and will push the network out of this region. Using the dot product formula for the angle,

$$\theta = \cos(u)$$
$$u = \frac{\langle dX, dY \rangle}{|dX||dY|}$$
$$E_\theta = \frac{1}{2}\theta^2$$

we can derive the back-propagation rule for the angle error. However, the derivative for angle can be unbounded, so we approximate E_θ by $-2u - 2$. The error gradients for the visible units y_i are:

$$-\frac{\partial E_\theta}{\partial w_{ij}} = 2\frac{\partial u}{\partial y_i}\dot{y}_i y_j$$
$$-\frac{\partial E_\theta}{\partial w_{ij}} = 2(\frac{dT_i}{|dT||dY|} - u\frac{dY_i}{|dY|^2})\dot{y}_i y_j$$

The gradients for the hidden units are entirely analogous to regular back-propagation. The final error function we used were a combination of the summed square error and the angle error:

$$E = E_{SSE} + \beta E_\theta$$

where β is on the order of 0.001. We also switched to (-1,1) units. The network was then able to successfully learn to oscillate. The training time was typically from 40000 to 100000 iterations. The error profile was similar to learning sine waves; the network spiraled inwards, went through a series of bifurcations to become periodic, and then stabilizes. However, when run as an iterated prediction network after learning, the attractor learned by the network was typically tilted compared to the target trajectory (see Figure 9 (a)). Occasionally it learns a better fit (Figure 9 (b,c)).

This points out another subtle issue in training periodic attractors. Even though locally contracting vector field can ensure the stability of the periodic trajectory, the small amount of error remaining can dramatically alter the shape at difficult regions of the attractor when iterated. This is illustrated in (Figure 9 (d)). Instead of following a difficult turn, the network can miss it altogether, resulting in a slant in the overall phase structure. Perhaps further adjustments to the objective function, or learning parameters, would eventually lead the network to learn this task better. In the next section, we used batched phase space learning (non-autonomous) to achieve better results on this problem.

5.3 Batched phase space learning of the van Der Pol oscillator

In this section, we used the same parameters for the van der Pol oscillator. However, we generated a training set in advance using the vector field near the attractor, again using a step size of 0.1. for the next iterate. There are 1500 data pairs in the training set (Figure 10 (a)). The order of the training pairs were randomized.

The attractor learned by network with two layers of 20 hidden units each, after 15000 epochs of training, matched the teacher much better (Figure 10 (b)). The trajectory of the network is drawn as a continuous curve, and the teachers as dots. Comparison of the temporal trajectories are shown in Figure 10 (c), where one can see that there is a slight frequency difference, but the waveform is very close. The average mean square error is 0.000136. Note that the training time was much longer than ARTISTE; 15000 epochs with 1500 data pairs correspond to 22.5 million iterations. However, we did not try very hard to optimize the network. Results from a network with two layers of 5 hidden units each with only 500 data pairs as training set were similar (with MSE=0.00034).

6 Summary and discussion

We have presented a phase space view of the learning process in recurrent nets. This perspective has helped us to understand and overcome some of the problems of using traditional recurrent methods for learning periodic and chaotic attractors. As a consequence of learning the vector field, the phase space methods include explicit stability constraints on the network. The phase space method essentially breaks the difficult recurrent training problem into three sub-problems: (1) That of embedding a temporal signal to recover its phase space structure, (2) generating local approximations of the vector field near the desired trajectory, and (3) functional approximation in feedforward networks. Existing methods developed for these three problems can be directly and independently applied here to help solve the recurrent network training problem. This also means a reduction in the complexity of a single learning trial from the $O(n^4)$ of RTRL and $O(kn^2)$ (where k is the number of steps in time used) of

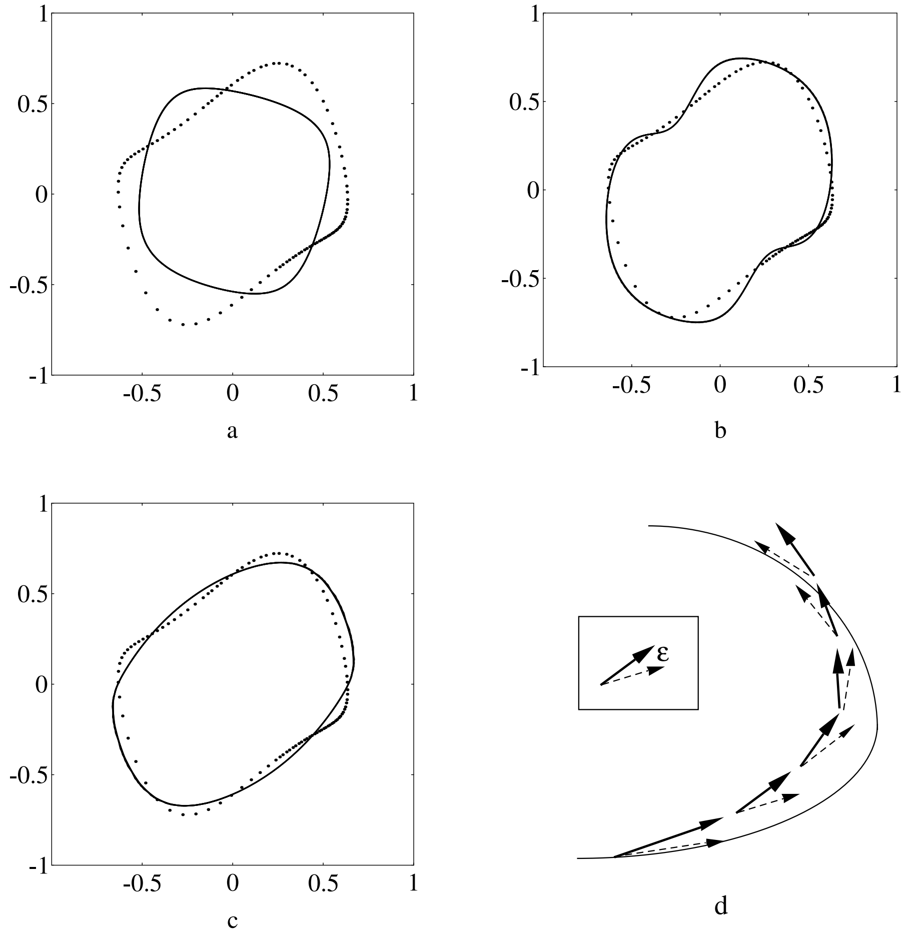


Figure 9: ARTISTE applied to learning the van Der Pol oscillator. a) The attractor typically learned by ARTISTE. The network has 2-4-4 units that range from -1 to 1, learning rate is 0.5, scaled by the number of fan-in. Momentum = 0.5. Trained for 60000 (60K) iterations. b) In about 3 out of 20 trials, the network learned this attractor in 60K iterations. The learning rate was annealed to 0.01 after 40K. c) In one trial out of 20, the network learned this attractor. 100K iterations, learning rate annealed to 0.05 after 80K. d) Why the attractor is slanted. Even though each step is only ϵ distance off, the overall iterated curve has a different global quality.

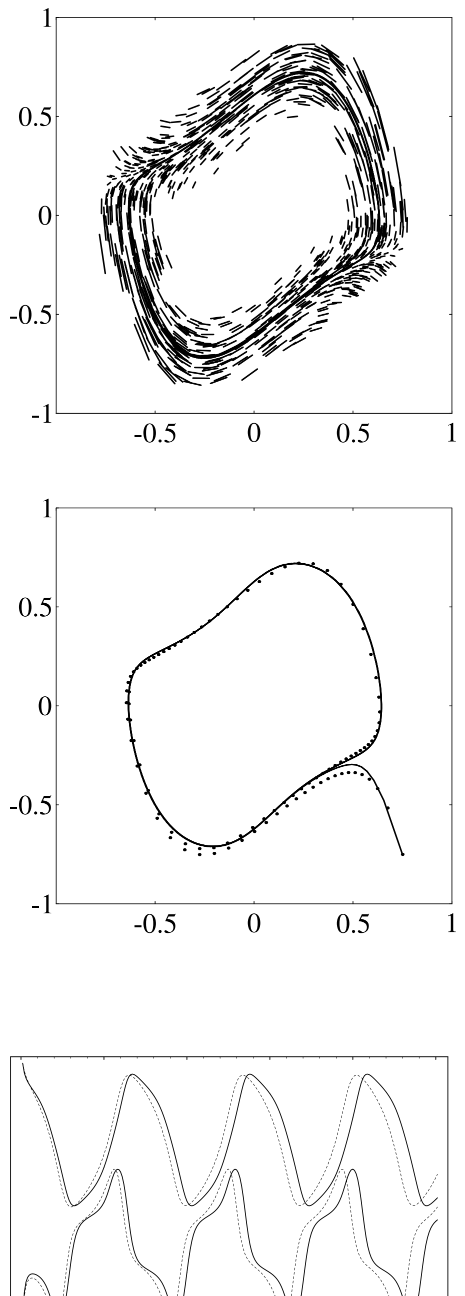


Figure 10: Phase space learning of the van Der Pol oscillator. a) The vector field near the periodic attractor, which is used as the training set. b) The network is 2-20-20, -1 to 1 units, with no connections between the visible units. Learning rate 0.01, scaled by the fan-in. Momentum = 0.75. 15000 epochs. Note that in the lower right hand corner, the network (solid curve) converges to the attractor faster than the teacher (dots). c) Temporal trajectory of the network compared with the teacher. The frequency is slightly off.

BPTT to the $O(n^2)$ of standard back propagation, plus the cost of embedding, and the cost of local approximations. This approximation cost can be constant for simple trajectories, but is often dependent on the number of training data. Even though in one simple example, training time was longer, it is expected to scale better as the dimension and complexity of the dynamical system increases. At the least, phase space learning provides a workable way to approach learning complicated temporal trajectories.

At the time of this writing we have only limited, though encouraging, simulation results. Periodic attractors can be learned without teacher forcing, by allowing the network to explore the phase space according to its own dynamics and supplying the appropriate teacher on the spot. This is a natural active-selection mechanism as defined by [5]: The selection of training examples depends explicitly on the network state. As far as we know, this is the only system to learn oscillations without teacher forcing and using active selection. Using predetermined local vector field training set and epoch-wise phase space training, the network can learn a very close approximation of the van Der Pol oscillator.

References

[1] Williams, R., Zipser, D., 1989, *Neural Computation*, 1, 270-280.

[2] Rumelhart, D.E., Hinton, G.E. & Williams, R.W.

[3] Pearlmutter, B., 1989, *Neural Computation*, 1, 263-269.

[4] Lapedes, A., and Farber, R., 1986, *Physica D*, 22, 247-259.

[5] Plutowski, M., Cottrell, G., White, H., 1993, "Learning Mackey-Glass from 25 examples, plus or minus 2," In *Advances in neural information processing systems 5*, San Mateo: Morgan Kaufman.

[6] Tsung, F-S., 1991, "Learning in recurrent finite difference networks". In D.S. Touretzky, J.L. Elman, T.J. Sejnowski & G.E. Hinton (Eds.), *Connectionist Models: Proceedings of the 1990 Summer School*, 124-130, San Mateo: Morgan Kauffman.

[7] Williams, R., Zipser, D., 1990, *College of Computer Science Technical Report*, NU-CCS-90-9, Northeastern University.

[8] Pineda, F. J., 1988, *Journal of Commplexity*, 4, 216-245.

[9] Tsung, F-S., Cottrell, G.W. & Selverston, A.I. 1990, "Some experiments on learning stable network oscillations". *Proceedings of the international Joint Conference on Neural Networks*, San Diego, June 1990.

[10] Zipser, David. Personal communication.

[11] Mozer, M. C., 1993, Neural net architectures for temporal sequence processing, To appear in: A. S. Weigend and N. A. Gershenfeld (Eds.), *Predicting the future and understanding the past: a comparison of approaches*, Redwood City: Addison-Wesley.

[12] Hornik, K., Stinchcombe, M., White, H., 1989, *Neural Networks*, 2, 359-366.

[13] Packard, N. H., Crutchfield, J. P., Farmer, J. D., Shaw, R. S., 1980, *Phys. Rev. Lett.*, 45, 712-716.

[14] Takens, F., 1981, "Detecting strange attractors in turbulence". In D. A. Rand and L-S. Young (Eds.), *Lecture Notes in Mathematics 898*. Berlin: Springer-Verlag.

[15] Farmer, J. D., and Sidorowich, J. J., 1987, *Phys. Rev. Lett.*, 59, 845-848.

[16] Tsung, F-S. & Cottrell, G. W., 1993, "Hopf bifurcation and Hopf hopping in recurrent nets". To appear in: *IEEE International Conference on Neural Networks*, San Francisco, USA.