Learning Long-Term Dependencies with Gradient Descent is Difficult

Yoshua Bengio†, Patrice Simard†, and Paolo Frasconi‡

†AT&T Bell Laboratories

‡Dip. di Sistemi e Informatica, Universitá di Firenze

Abstract

Recurrent neural networks can be used to map input sequences to output sequences, such as for recognition, production or prediction problems. However, practical difficulties have been reported in training recurrent neural networks to perform tasks in which the temporal contingencies present in the input/output sequences span long intervals. We show why gradient based learning algorithms face an increasingly difficult problem as the duration of the dependencies to be captured increases. These results expose a trade-off between efficient learning by gradient descent and latching on information for long periods. Based on an understanding of this problem, alternatives to standard gradient descent are considered.

Paper to appear in the special issue on Recurrent Networks of the IEEE Transactions on Neural Networks

1 Introduction

We are interested in training recurrent neural networks to map input sequences to output sequences, for applications in sequence recognition, production or time-series prediction. All of the above applications require a system that will store and update context information, i.e., information computed from the past inputs and useful to produce desired outputs. Recurrent neural networks are well suited for those tasks because they have an internal state that can represent context information. The *cycles* in the graph of a recurrent network allow it to keep information about past inputs for an amount of time that is not fixed a-priori, but rather depends on its weights and on the input data. In contrast, static networks (i.e., with no recurrent connection), even if they include delays (such as Time Delay Neural Networks [15]), have a finite impulse response and can't store a bit of information for an indefinite time. A recurrent network whose inputs are not fixed but rather constitute an *input sequence* can be used to transform an input sequence into an output sequence while taking into account contextual information in a flexible way. We restrict here our attention to discrete-time systems.

Learning algorithms used for recurrent networks are usually based on computing the gradient of a cost function with respect to the weights of the network [22, 21]. For example, the back-propagation through time algorithm [22] is a generalization of back-propagation for static networks in which one stores the activations of the units while going forward in time. The backward phase is also backward in time and recursively uses these activations to compute the required gradients. Other algorithms, such as the forward propagation algorithms [14, 23], are much more computationally expensive (for a fully connected recurrent network) but are local in time, i.e., they can be applied in an on-line fashion, producing a partial gradient after each time step. Another algorithm was proposed [10, 18] for training constrained recurrent networks in which dynamic neurons – with a single feedback to themselves – have only incoming connections from the input layer. It is local in time like the forward propagation algorithms and it requires computation only proportional to the number of weights, like the back-propagation through time algorithm. Unfortunately, the networks it can deal with have limited storage capabilities for dealing with general sequences [7], thus limiting their representational power.

A task displays long-term dependencies if computation of the desired output at time t depends on input presented at an earlier time $\tau \ll t$. Although recurrent networks can in many instances outperform static networks [4], they appear more difficult to train optimally. Earlier experiments indicated that their parameters settle in sub-optimal solutions which take into account short-term dependencies but not long-term dependencies [5]. Similar results were obtained by Mozer [19]. It was found that back-propagation was not sufficiently powerful to discover contingencies spanning long temporal intervals. In this paper, we present experimental and theoretical results in order to further the understanding of this problem.

For comparison and evaluation purposes, we now list three basic requirements for a parametric dynamical system that can *learn* to *store* relevant state information. We require that

1. the system be able to store information for an arbitrary duration,
2. the system be resistant to noise (i.e., fluctuations of the inputs that are random or irrelevant to predicting a correct output).
3. the system parameters be trainable (in reasonable time).

Throughout the paper, the long-term storage of definite bits of information into the state variables of the dynamic system is referred to as *information latching*. A formalization of this concept, based on hyperbolic attractors, is given in section 4.1.

The paper is divided in five sections. In Section 2 we present a minimal task that can be solved only if the system satisfies the above conditions. We then present a recurrent network candidate solution and negative experimental results indicating that gradient descent is not appropriate even for such a simple problem. The theoretical results of Section 4 show that either such a system is stable and resistant to noise or, alternatively, it is efficiently trainable by gradient descent, but not both. The analysis shows that when trying to satisfy conditions (1) and (2) above, the magnitude of the derivative of the state of a dynamical system at time t with respect to the state at time 0 decreases exponentially as t increases. We show how this makes the back-propagation algorithm (and gradient descent in general) inefficient for learning of long term dependencies in the input/output sequence, hence failing condition (3) for sufficiently long sequences. Finally, in Section 5, based on the analysis of the previous sections, new algorithms and approaches are proposed and compared to variants of back-propagation and simulated annealing. These algorithms are evaluated on simple tasks on which the span of the input/output dependencies can be controlled.

2 Minimal Task Illustrating the Problem

The following minimal task is designed as a test that must necessarily be passed in order to satisfy the three conditions enumerated above. A parametric system is trained to classify two different sets of sequences of length T. For each sequence $u_1, \ldots, u_T$ the class $\mathcal{C}(u_1, \ldots, u_T) \in \{0, 1\}$ depends only on the first L values of the external input:

$$\mathcal{C}(u_1, \ldots, u_T) = \mathcal{C}(u_1, \ldots, u_L).$$

We suppose L fixed and allow sequences of arbitrary length $T \gg L$. The system should provide an answer at the end of each sequence. Thus, the problem can be solved only if the dynamic system is able to store information about the initial input values for an arbitrary duration. This is the simplest form of long-term computation that one may ask a recurrent network to carry out. The values $u_{L+1}, \ldots, u_T$ are irrelevant for determining the class of the sequences. However, they may affect the evolution of the dynamic system and eventually erase the internally stored information about the initial values of the input. Thus the system must latch information robustly, i.e., in such a way that it cannot be easily deleted by events which are unrelated with the classification criterion. We assume here that for each sequence, u_t is zero-mean Gaussian noise for $t > L$.

The third required condition is learnability. There are two different computational aspects involved in this task. First, it is necessary to process $u_1, \cdots, u_L$ in order to extract some information about the class, i.e., perform *classification*. Second, it is necessary to store such information into a subset of the state variables (let us call them *latching* state variables) of the dynamic system, for an arbitrary duration. For this task, the computation of the class does not require accessing latching state variables. Hence the latching state variables do not need to affect the evolution of the other state variables. Therefore, a simple solution to this task may use a latching subsystem, fed by a subsystem that computes information about the class.

We are interested in assessing learning capabilities on this latching problem independently on a particular set of training sequences, i.e. in a way that is independent on the specific problem of classifying $u_1, \ldots, u_L$. Therefore we will focus here only on the latching subsystem. In order to train any module feeding the latching subsystem, the learning algorithm should be able to transmit error information (such as gradient) to such a module. An important question is thus whether the learning algorithm can propagate error information to a module that feeds the latching subsystem and detects the events leading to latching.

Hence, instead of feeding a recurrent network with the input sequences defined as above we use only the latching subsystem as a test system and we reformulate our minimal task as follows. The test system has one input h_t and one output x_t (at each discrete time step t). The initial inputs h_t, for $t \leq L$, are values which can be *tuned by the learning algorithm* (e.g., gradient descent) whereas h_t is Gaussian noise for $L < t \leq T$. The connection weights of the test system are also trainable parameters. Optimization is based on the cost function

$$C = \frac{1}{2} \sum_p (x_T^p - d^p)^2$$

where p is an index over the training sequences and d^p is a target of $+0.8$ for sequence of class 1 and -0.8 for sequences of class 0.

In this formulation, h_t $(t = 1, \ldots, L)$ represent the result of the computation that extracts the class information. Learning h_t directly is an easier task than computing it as a parametric function $h_t(u_t, \theta)$ of the original input sequence and learning the parameters θ. In fact, the error derivatives $\frac{\partial C}{\partial h_t}$ (as used by backpropagation through time) are the same as if h_t were obtained as a parametric function of u_t. Thus, if h_t cannot be directly trained as parameters in the test system (because of vanishing gradient), they clearly cannot be trained as a parametric function of the input sequence in a system that uses a trainable module to feed a latching subsystem. The ability of learning the free input values $h_1, \ldots, h_L$ is a measure of the effectiveness of the gradient of error information which would be propagated further back if the test system were connected to the output of another module.

3 Simple Recurrent Network Candidate Solution

We performed experiments on this minimal task with a single recurrent neuron, as shown in Fig. 1a. Two types of trajectories are considered for this test system, for the two classes ($k = 0,\ k = 1$):

$$\begin{aligned} x_t^k &= f(a_t^k) = \tanh(a_t^k) \\ a_t^k &= w\, f(a_{t-1}^k) + h_t^k \qquad t = 1 \ldots T \\ a_0^0 &= a_0^1 = 0 \end{aligned} \tag{1}$$

If $w > \ 1/f'(0)\ = 1$, then the autonomous dynamic of this neuron has two attractors $\overline{x} > 0$ and $-\overline{x}$ that depend on the value of the weight w [7, 8] (they can be easily obtained as non zero intersections of the curve $x = \tanh(a)$ with the line $x = a/w$). Assuming that the initial state at $t = 0$ is $x_0 = -\overline{x}$, it can be shown [8] that there exists a value $h^* > 0$ of the input such that, (1) x_t maintains its sign if $|h_t| < h^* \ \ \forall t$, and, (2) there exists a finite number of steps L_1 such that $x_{L_1} > \overline{x}$ if $h_t > h^* \ \ \forall t \leq L_1$. A symmetric case occurs for $x_0 = -\overline{x}$. h^* increases with w. For fixed w, the transient length L_1 decreases with $|h_t|$. Thus the recurrent neuron of Fig. 1a. can robustly latch one bit of information, represented by the sign of its activation. Storing is accomplished by keeping a large input (i.e., larger than h^* in absolute value) for a long enough time. Small noisy inputs (i.e., smaller than h^* in absolute value) cannot change the sign of the activation of the neuron, even if applied for arbitrary long time. This robustness essentially depends on the nonlinearity.

The recurrent weight w is also trainable. The solution for $T \gg L$ requires $w > 1$ to produce two stable attractors $\overline{x}$ and $-\overline{x}$. Larger w correspond to larger critical value h^* and, consequently, more robustness against noise. The trainable input values must bring the state of the neuron towards $\overline{x}$ or $-\overline{x}$ in order to robustly latch a bit of information against the input noise. For example this can be accomplished by adapting, for $t = 1, \ldots, L$,

$h_t^1 \geq H$ and $h_t^0 \leq -H$, where $H > h^*$ controls the transient duration towards one of the two attractors.

In Fig. 1b we show two sample sequences that feed the recurrent neuron. As stated in section 2, h_t^k are trainable for $t \leq L$ and samples from a Gaussian distribution with mean 0 and variance s for $t > L$. The values of h_t for $t \leq L$ were initialized to small uniform random values before starting training. A set of simulations were carried out to evaluate the effectiveness of back-propagation (through time) on this simple task. In a first experiment we investigated the effect of the noise variance s and of different initial values w_0 for the self loop weight (see also [3].) A density plot of convergence is shown in Fig. 2a, averaged over 18 runs for each of the selected pairs (w_0, s). It can be seen that convergence becomes very unlikely for large noise variance or small initial values of w. $L = 3$ and $T = 20$ were chosen in these experiments.

In Fig. 2b, we show instead the effect of varying T, keeping fixed $s = 0.2$ and $w_0 = 1.25$. In this case the task consists in learning only the input parameters u_t. As explained in section 2, If the learning algorithm is unable to properly tune the inputs u_t, then it will not be able to learn *what* should trigger latching in a more complicated situation. Solving this task is a minimum requirement for being able to transmit error information backwards, towards modules feeding the latch unit.

When T becomes large it is extremely difficult to attain convergence. These experimental results show that even in the very simple situation where we want to robustly latch on 1 bit of information about the input, gradient descent on the output error fails for long-term input/output dependencies, for most initial parameter values.

4 Learning to Latch with Dynamical Systems

In this section, we attempt to understand better why learning even simple long-term dependencies with gradient descent appears to be so difficult. We discuss the general case of a real-time recognizer based on a parametric dynamical system. We find that the conditions under which a recurrent network can robustly store information (in a way defined below, i.e. with hyperbolic attractors) yield a problem of vanishing gradients that can make learning very difficult.

We consider a non-autonomous discrete-time system with additive inputs:

$$a_t = M(a_{t-1}) + u_t \tag{2}$$

and the corresponding autonomous dynamics

$$a_t = M(a_{t-1}) \tag{3}$$

where M is a nonlinear map, and a_t and u_t are n-vectors representing respectively the system state and the external input at time t.

To simplify the analysis presented in this section, we consider only a system with additive inputs. However, a dynamic system with non-additive inputs, e.g., $a_t = N(a_{t-1}, u_{t-1})$, can be transformed into one with additive inputs by introducing additional state variables and corresponding inputs. Suppose $a_t \in \mathbf{R}^n$ and $u_t \in \mathbf{R}^m$. The new system is defined by the additive inputs dynamics $a'_t = N'(a'_{t-1}) + u'_t$ where $a'_t = (a_t, y_t)$ is a n+m-vector state, and the first n elements of $u'_t = (0, u_t) \in \mathbf{R}^{n+m}$ are 0. The new map N' can be defined in terms of the old map N as follows: $N'(a'_{t-1}) = (N(a_{t-1}, y_{t-1}), 0)$, with m zeroes for the last elements of $N'()$. Hence we have $y_t = u_t$. Note that a system with additive inputs with a map of the form of $N'()$ can be transformed back into an equivalent system with non-additive inputs. Hence without loss of generality we can use the model in eq. 2.

In the next subsection, we show that only two conditions can arise when using hyperbolic attractors to latch bits of information. Either the system is very sensitive to noise, or the derivatives of the cost at time t with respect to the system activations a_0 converge exponentially to 0 as t increases.

This situation is the essential reason for the difficulty in using gradient descent to train a dynamical system to capture long-term dependencies in the input/output sequences.

4.1 Analysis

In order to latch a bit of state information one wants to restrict the values of the system activity a_t to a subset S of its domain. In this way, it will be possible to later interpret a_t in at least two ways: inside S and outside S. To make sure that a_t remains in such a region, the system dynamics can be chosen such that this region is the basin of attraction of an attractor (or of an attractor in a sub-manifold or subspace of a_t's domain). To "erase" that bit of information, the inputs may push the system activity a_t out of this basin of attraction and possibly into another one. In this section, we show that if the attractor is hyperbolic (or can be transformed into one, e.g. a stable periodic attractor), then the derivatives $\frac{\partial a_t}{\partial a_0}$ quickly vanish as t increases. Unfortunately, when these gradients vanish, training becomes very difficult because the influence of short-term dependencies dominates in the weights gradient.

Definition 1 *A set of points E is said to be* invariant *under a map M if $E = M(E)$.*

Definition 2 *A* hyperbolic attractor *is a set of points X invariant under the differentiable map M, such that $\forall a \in X$, all eigenvalues of $M'(a)$ are less than 1 in absolute value.*

An attractor X may contain a single point (fixed point attractor), a finite number of points (periodic attractor), or an infinite number of points (chaotic attractor). Note that a stable and attracting fixed point is hyperbolic for the map M, whereas a stable and attracting periodic attractor of period l for the map M is hyperbolic for the map M^l. For a recurrent net, the kind of attractor depends on the weight matrix. In particular, for a network defined by $a_t = W \tanh(a_{t-1}) + u_t$, if W is symmetric and its minimum eigenvalue is greater than -1, then the attractors are all fixed points [17]. On the other hand, if $|W| < 1$ or if the system is linear and stable, the system has a single fixed point attractor at the origin.

Definition 3 *The* basin of attraction *of an attractor X is the set $\beta(X)$ of points a* converging *to X under the map M, i.e.,* $\beta(X) = \{a : \forall \epsilon, \exists l, \exists x \in X \text{ s.t. } \|M^l(a) - x\| < \epsilon\}$.

Definition 4 *We call $\Gamma(X)$, the* reduced attracting set *of a hyperbolic attractor X, the set of points y in the basin of attraction of X, such that $\forall l \geq 1$, all the eigenvalues of $(M^l)'(y)$ are less than 1.*

Note that by definition, for a hyperbolic attractor X, $X \subset \Gamma(X) \subset \beta(X)$.

Definition 5 *A system is* robustly latched *at time t_0 to X, one of several hyperbolic attractors, if a_{t_0} is in the reduced attracting set of X under a map M defining the autonomous system dynamics.*

For the case of non-autonomous dynamics, it remains robustly latched to X as long as the inputs u_t are such that $a_t \in \Gamma(X)$ for $t > t_0$. Let us now see why it is more robust to store a bit of information by keeping a_t in $\Gamma(X)$, the reduced attracting set of X.

Theorem 1 *Assume x is a point of $\mathbf{R}^n$ such that there exist an open sphere $U(x)$ centered on x for which $|M'(z)| > 1$ for all $z \in U(x)$. Then there exist $y \in U(x)$ such that $\|M(x) - M(y)\| > \|x - y\|$.*

Proof: see the Appendix.

This theorem implies that for a hyperbolic attractor X, if a_0 is in $\beta(X)$ but not in $\Gamma(X)$, then the size of a ball of uncertainty around a_0 will grow exponentially as t increases, as illustrated in figure 3(a). Therefore, small perturbations in the input could push the trajectory towards another (possibly wrong) basin of attraction. This means that *the system will not be resistant to input noise.* What we call input noise here may be simply components of the inputs that are not relevant to predict the correct future outputs. In contrast, the following results show that if a_0 is in $\Gamma(X)$, a_t is guaranteed to remain within a certain distance of X when the input noise is bounded.

Definition 6 *A map M is* contracting *on a set D if $\exists \alpha \in [0,1)$ such that $\|M(x) - M(y)\| \leq \alpha \|x - y\| \;\; \forall x, y \in D$.*

Theorem 2 *Let M be a differentiable mapping on a convex set D. If $\forall x \in D$, $|M'(x)| < 1$, then M is contracting on D.*

Proof: See [20].

A crucial element in this analysis, is to identify the conditions in which one can robustly latch information with an attractor:

Theorem 3 *Suppose the system is robustly latched to X, starting in state a_0, and the inputs u_t are such that for all $t > 0$, $\|u_t\| < b_t$, where $b_t = (1 - \lambda_t)d$. Let $\tilde{a}_t$ be the autonomous trajectory obtained by starting at a_0 and no input u. Also suppose $\forall y \in D_t$, $|M'(y)| < \lambda_t < 1$, where D_t is a ball of radius d around a_t. Then a_t remains inside a ball of radius d around $\tilde{a}_t$, and this ball intersects X when $t \to \infty$.*

Proof: see the Appendix.

The above results justifies the term "robust" in our definition of robustly latched system: as long as a_t remains in the reduced attracting set $\Gamma(X)$ of a hyperbolic attractor X, a bound on the inputs can be found that guarantees a_t to remain within a certain distance of some point in X, as illustrated in figure 3(b). The smaller is $|M'(y)|$ in the region around a_t, the looser is the bound b_t on the inputs, meaning that the system is more robust to input noise. On the other hand, outside $\Gamma(X)$ but in $\beta(X)$, M is not contracting, it is expanding, i.e., the size of a ball of uncertainty grows exponentially with time.

We now show the consequences of robust latching, i.e., vanishing gradient:

Theorem 4 *If the input u_t is such that a system remains robustly latched on attractor X after time 0, then $\frac{\partial a_t}{\partial a_0} \rightarrow 0$ as $t \rightarrow \infty$.*

Proof: see the Appendix.

The results in this section thus show that when storing one or more bit of information in a way that is resistant to noise, the gradient with respect to past events rapidly becomes very small in comparison to the gradient with respect to recent events. In the next section we discuss how that makes gradient descent on parameter space (e.g., the weights of a network) inefficient.

4.2 Effect on the Weight Gradient

Let us consider the effect of vanishing gradients on the derivatives of a cost C_t at time t with respect to parameters of a dynamical system, say a recurrent neural network with weights W:

$$\frac{\partial C_t}{\partial W} = \sum_{\tau \leq t} \frac{\partial C_t}{\partial a_\tau} \frac{\partial a_\tau}{\partial W} = \sum_{\tau \leq t} \frac{\partial C_t}{\partial a_t} \frac{\partial a_t}{\partial a_\tau} \frac{\partial a_\tau}{\partial W} \tag{4}$$

Suppose we are in the condition in which the network has robustly latched. Hence for a term with $\tau \ll t$, $\left|\frac{\partial C_t}{\partial a_\tau}\frac{\partial a_\tau}{\partial W}\right| \rightarrow 0$. This term tends to become very small in comparison to terms for which τ is close to t. This means that even though there might exist a change in W that would allow a_τ to jump to another (better) basin of attraction, the gradient of the cost with respect to W does not reflect that possibility. This is because the effect of a **small** change in W would be felt mostly on the near past (τ close to t).

Let us see an example of how this result hampers training a system that requires robust latching of information. Consider for example a system made of two sub-systems A and B with the output of A being fed to the input of B. Suppose that any good solution to the learning problem requires B storing information about events detected by A at time 0, with the output of B at a later distant time T used to compute an error, as in our minimal problem defined in section 2. If B has not been trained enough to be able to store information for a long time, then gradients of the error at T with respect to the output of A at time 0 are very small since B doesn't latch and the outputs of A at time 0 have very little influence on the error at time T. On the other hand, as soon as B is trained enough to reliably store information for a long time, the right gradients can propagate, but because they quickly vanish to very small values, training A is very difficult (depending of the size of T and the amount of noise between 0 and T).

5 Alternative Approaches

The above section helped us understand better why training a recurrent network to learn *long range* input/output dependencies is a hard problem. Gradient-based methods appear inadequate for this kind of problem. We need to consider alternative systems and optimization methods that give acceptable results even when the criterion function is not smooth and has long plateaus. In this section we consider several alternative optimization algorithms for this purpose, and compare them to two variants of back-propagation.

One way to help in the training of recurrent networks is to set their connectivity and initial weights (and even constraints on the weights) using prior knowledge. For example, this is accomplished in [8] and [11] using prior rules and sequentiality constraints. In fact, the results in this paper strongly suggest that when such prior knowledge is given, it should be used, since the learning problem itself is so difficult. However, there are many instances where many long-term input/output dependencies are unknown and have to be learned from examples.

5.1 Simulated Annealing

Global search methods such as simulated annealing can be applied to such problems, but they are generally very slow. We implemented the simulated annealing algorithm presented in [6] for optimizing functions of continuous variables. This is a "batch learning" algorithm (updating parameters after all examples of the training set have been seen). It performs a cycle of random moves, each along one coordinate (parameter) direction. Each point is accepted or rejected according to the Metropolis criterion [13]. New points are selected according to a uniform distribution inside a hyperrectangle around the last point. The dimensions of the hyperrectangle are updated in order to maintain the average percentage of accepted moves at about one-half of the total number of moves. After a certain number of cycles, the temperature is reduced by a constant multiplicative factor (0.85 in the experiments). Training stops when some acceptable value of the cost function is attained, when learning gets "stuck" [1], or if a maximum number of function evaluations is performed. A

[1] when the cost value on the last N_ϵ points does not change by more than ϵ (a small constant) and these values are all within ϵ of the current optimal cost value found by the algorithm. In the experiments, $\epsilon = 0.001$ and $N_\epsilon = 4$.

'function evaluation' corresponds to performing a single pass through the network, for one input sequence.

5.2 Multi-Grid Random Search

This simple algorithm is similar to the simulated annealing algorithm. Like simulated annealing, it tries random points. However, if the main problem with the learning tasks was plateaus (rather than local minima), an algorithm that accepts only points that reduce the error could be more efficient. This algorithm has this property. It performs a (uniform) random search in a hyperrectangle around the current (best) point. When a better point is found, it reduces the size of the hyperrectangle (by a factor of 0.9 in the experiments) and re-centers it around the new point. The stopping criterion is the same as for simulated annealing.

5.3 Time-Weighted Pseudo-Newton Optimization

The pseudo-Newton algorithm [2] for neural networks has the advantage of re-scaling the learning rate of each weight dynamically to match the curvature of the energy function with respect to that weight. This is of interest because adjusting the learning rate could potentially circumvent the problem of vanishing gradient. The pseudo-Newton algorithm computes a diagonal approximation to the Hessian matrix (second derivatives of the cost with respect to the parameters) and updates parameters according to the following on-line rule:

$$\Delta w_i(p) = -\frac{\eta}{\frac{\partial^2 C(p)}{|\partial w_i^2|} + \mu} \times \frac{\partial C(p)}{\partial w_i} \tag{5}$$

where $\Delta w_i(p)$ is the update for weight w_i after pattern p has been presented, $C(p)$ is the cost for pattern p, and μ and η are small positive constants. This amounts to computing a local learning rate for each parameter by using the inverse of the second derivative with respect to each parameter as a normalizing factor. When $\frac{\partial^2 C(p)}{|\partial w_i^2|}$ is small, the curvature is small (around the current value of w) in the direction corresponding to the w_i axis. Hence a larger step can be taken in that direction. This algorithm was tested in the experiments described in section 5.5. It consistently performs better than standard back-propagation but still fails more and more as we increase the span of input/output dependencies.

This algorithm and our theoretical results of section 4 inspired the following *time-weighted pseudo-Newton* algorithm. The basic idea is to consider the unfolding of the recurrent network in time, and each instantiation of a weight (at different times) as a *separate* variable, albeit with the *constraint* that these now separate variables should be equal. To simplify the problem, we consider here a cost $C(p)$ which depends on the output of the network at the final time step of sequence p. Hence the weight update for w_i can be computed as follows:

$$\Delta w_i(p) = -\sum_t \frac{\eta}{\frac{\partial^2 C(p)}{|\partial w_{it}^2|} + \mu} \times \frac{\partial C(p)}{\partial w_{it}} \tag{6}$$

where w_{it} is the instantiation for time t of parameter w_i. In this way, each (temporal) contribution to $\Delta w_i(p)$ is weighted by the inverse curvature with respect to w_{it}, the instantiation of parameter w_i at time t [2]. The reader may compare the above equation with equation 4, where all the temporal contributions are uniformly summed. Consequently, updating w according to equation 6 does not actually follow the gradient (but neither would following equation 5). Instead, several gradient contributions are weighted using second derivatives, in order to make faster moves in the flatter directions. Like for the pseudo-Newton algorithm of [2], we prefer using a diagonal approximation of the Hessian

[2]The idea of using second derivatives in this way was inspired from discussions with L. Bottou.

which is cheap to compute and guaranteed to be positive. η is a global learning rate (0.01 in our experiments). The constant μ is introduced to prevent Δw from becoming very large (when $\frac{\partial^2 C(p)}{|\partial w_{it}^2|}$ is very small). However, we found that much better performance can be attained with the recurrent networks when μ is adapted on-line. This prevents the maximum Δw from being greater than a certain upper bound (0.3 in the experiments) or smaller than a certain lower bound (0.001 in the experiments). The constant μ is updated with a "momentum" term (0.8 in the experiments), in order to prevent it from decreasing too rapidly when the first and second derivatives vary widely from sequence to sequence and have very small magnitude (for example when the norm of the weight matrix $|W|$ is less than 1).

5.4 Discrete Error Propagation

The analysis of section 4 could suggest that the root of the problem lies in the essentially *discrete* nature of the process of storing information for an indefinite amount of time. Indeed, the gradient backpropagated through time vanishes when the system stays in the same stable state for several time steps. Intuitively, we would like to recover some error information at the time when the input made the system reach that stable state. Instead of propagating a gradient through differentiable units, the algorithm presented here was explicitly designed to propagate discrete error information through units that compute a non-differentiable function, such as a hard threshold. In this way we hope to find algorithm that directly address the problem of propagating error backwards in time, even though the process of robustly storing information appears to have a discrete nature.

Other methods have been explored in order to train layered networks of hard threshold units. For example, in [1] it is shown how to train two layered networks using a probabilistic approach. In [9] a method is proposed that iterates two training steps: adjusting the network internal representation (units activations) and training the parameters to produce such representation. This algorithm can be applied to recurrent networks as well. Both methods take advantage of probabilities in order to make differentiable the error function, thus permitting the use of gradient descent. Another approach, proposed in [12], applies to two layer networks. The space of activities of hidden units is searched in a greedy way in order reduce output error. An earlier algorithm also related to the one presented here, but based on the propagation of targets was proposed in [16]. The algorithm introduced here, instead, relies on propagating discrete error information, obtained with a finite difference approach.

A neural network can be represented as a series of local elements with each a forward propagation function and an error propagation function. We will derive these functions for a discrete element and show how they can be used together with standard differentiable elements to minimize a cost function. Our building block for discrete elements is the non-linear threshold function. The forward propagation is given by

$$y_i(x) = \mathrm{sign}(x_i) \tag{7}$$

where $y_i \in \{-1, 1\}$ is the output of unit i and $x_i \in \mathbf{R}$ its input. We are now interested in finding the discrete counterpart of gradient propagation for this unit. To backpropagate an error signal, we should first establish the relation between variations of the output Δy_i and variations of the input Δx_i. This can be done in a systematic way. The variation $\Delta y_i(\Delta x_i)$ can be easily computed from equation 7 by considering under which conditions the output y of a discrete threshold unit will change by 2, -2 or 0:

$$\Delta y_i = \left| \begin{array}{ll} 2 & \text{if } \ x_i < 0 \ \text{ and } \ x_i + \Delta x_i \geq 0 \\ -2 & \text{if } \ x_i \geq 0 \ \text{ and } \ x_i + \Delta x_i < 0 \\ 0 & \text{otherwise} \end{array} \right. \tag{8}$$

and from this equation we can compute the desired variation $\Delta x_i(\Delta y_i)$ of x_i when the desired variation of y_i is Δy_i:

$$\Delta x_i = \left| \begin{array}{ll} \epsilon - x_i & \text{if } \ \Delta y_i = 2 \\ -\epsilon - x_i & \text{if } \ \Delta y_i = -2 \\ 0 & \text{otherwise} \end{array} \right. \tag{9}$$

where ϵ is a positive constant. We now denote by Δy_i and Δx_i the desired changes in y_i and x_i respectively. Let C be a cost function on our system when a certain pattern (sequence) is presented. A "pseudo-gradient" $\frac{\Delta C}{\Delta x_i}$ should reflect the influence of a change of x_i on the cost C. In our experiment we set $\frac{\Delta C}{\Delta x_i}$ to $\frac{1}{\Delta x_i(\Delta y_i)}$ if $\Delta y_i \neq 0$ and 0 otherwise. To use the "pseudo gradient" we must insure that Δy_i is in $\{2, -2, 0\}$ since $\Delta x_i(\Delta y_i)$ is not defined for other values. This is achieved using a stochastic process. Let's assume that there exist two constants MIN and MAX such that the error signal $\frac{\Delta C}{\Delta y_i} = g_i$ to be backpropagated is a real number satisfying $MIN \leq g_i \leq MAX$. We define the stochastic function $\Delta y_i = S(g_i)$ which maps g_i to $\{-2, 2\}$ as follows:

$$\begin{cases} P(S(g_i) = 2) & = \frac{g_i - MIN}{MAX - MIN} \\ P(S(g_i) = -2) & = \frac{MAX - g_i}{MAX - MIN} \end{cases} \tag{10}$$

Provided that $-2 < MIN$ and $MAX < 2$, it is easy to show that the expectation of $S(g_i)$ is exactly g_i, even though $S(g_i)$can only take two values (if $|g_i| > 2$ the resulting expected value will be -2 or +2). Furthermore the sum of this "pseudo gradient" over several patterns quickly converges to the sum of the continuous valued g_i's.

The non-linear threshold unit can be used in combination with any other differentiable elements which backpropagate the gradient in the usual fashion. The important point is that when a non-linear threshold unit is connected to itself in a loop with a positive gain, two stable fixed points are induced. The "pseudo gradient" along this loop doesn't vanish with time which is the essential reason for using discrete units. This pseudo-gradient doesn't vanish along the loop, as can be observed by repetitively applying equations 9 and 8 and noting that if the pseudo-gradient is large enough in magnitude then it is always propagated.

This approach is in no way optimal and many other discrete error propagation algorithms are possible. Another very promising approach for instance is the trainable discrete flip-flop unit [3] which also preserves error information in time. Our only claim here is that discrete propagation of error offers interesting solutions to the vanishing gradient problem in recurrent network. Our preliminary results on toy problems (see next subsection and [3]) confirm this hypothesis.

5.5 Experimental Results

Experiments were performed to evaluate various alternative optimization approaches on problems on which one can increase the temporal span of input/output dependencies. Of course, when it is possible, first training on shorter sequences helps a lot, but in many problems no such "short-term" version of the problem is available. Hence a goal of these experiments was to measure how these algorithms can perform when it is not possible to train using sequences with equivalent short-term dependencies. Experiments were performed with and without input noise (uniformly distributed in [-0.2,0.2]) and varying the length of the input/output sequences. The criteria by which the performance of these algorithms were measured are (1) the average classification error at the end of training, i.e., after the stopping criterion has been met (when either some allowed number of function evaluations has been performed or the task has been learned), (2) the average number of function evaluations needed to reach the stopping criterion.

Experiments were performed on three problems: the Latch problem, the 2-Sequence problem, and the Parity problem. For each of these problems, a suitable architecture was chosen and all algorithms were used to search in the resulting parameter space (except that the discrete error propagation algorithm used hard threshold neurons instead of symmetric sigmoids). Initial parameters of the networks were randomly generated for each trial (uniformly between -0.5 and 0.5). The choice of inputs and the noise for each training sequence was also randomly generated for each trial. The same initial conditions and training set were used with each of the algorithms (at a given trial). For each trial, a training set was generated with sequences whose length is uniformly distributed between $T/2$ and T. The number T (maximum sequence length) is displayed in Figure 4. The tasks all involved a single input and a single output at each time step.

Latch Problem The Latch problem is the same as described above in Section 3. Here we considered only three adaptive parameters: the self-loop weight w, the initial input value u_1 for "positive" sequences (with positive final target), and the initial input value u_0 for "negative" sequences (with negative final target). The network thus had only one unit.

2-Sequence Problem The 2-Sequence problem is the following: classify an input sequence as one of two sequences when given the first N elements of this sequence. N varies from pattern to pattern and noise may be added to the input. Hence the network can't rely on the *last* particular values it saw. Instead, early on, it must recognize subsequences belonging to one of the two classes and store that information (or update it if conflicting information arrives) until its output is read out (which may be at any time but is done only once per sequence). These initial key subsequences were randomly generated from a uniform distribution in [-1,1]. In all experiments we used a fully connected recurrent network with 5 units and no bias (one of the units received external additive input, i.e., the network has 25 free parameters).

Parity Problem The Parity problem consists in producing the parity of an input sequence of 1's and -1's (i.e., a 1 should be produced in output if and only if the number of 1's in the input is odd). The target is only given at the end of the sequence. The length of the sequence may vary and the input may be noisy. It is a difficult problem that has local minima (like the XOR problem), and that appears more and more difficult for longer sequences. Most local optimization algorithms tend to get stuck in a local minimum for many initial values of the parameters. The minimal size network that we implemented has 7 free parameters and 2 units (2 inputs connected to 1 hidden and 1 output units). Although it requires less parameters than the 2-Sequence problem, it is a more difficult learning problem.

The results displayed in Figure 4 can be summarized as follows:

- Although simulated annealing performed well on all problems, it requires an order of magnitude more training time than all the other algorithms. This is not surprising since it is global search algorithm. The multi-grid algorithm is faster but fails on the Parity problem, probably because of local minima. It is also interesting to note that on the Latch problem with simulated annealing, training time increases with sequence length. Although the best solution is the same for all sequence lengths, the error surface for longer sequences could be more difficult to search, even for simulated annealing.

- The discrete error propagation algorithm performed reasonably well on all the problems and sequence lengths, and was the only one with simulated annealing that could solve the Parity problem. Because it performs an on-line local search it is however much faster than simulated annealing. It seems to be more robust to local minima than the multi-grid random search.

- The pseudo-Newton back-propagation algorithm consistently performs better than the standard back-propagation. However, both see their performance worsen when the temporal span of input/output dependencies increasing.

- The time-weighted pseudo-Newton algorithm appears to perform better than the other two variants of back-propagation but its performance also appears to worsen with increasing sequence length.

6 Conclusion

Recurrent networks are very powerful in their ability to represent context, often outperforming static networks [4]. However, we have presented theoretical and experimental evidence showing that gradient descent of an error criterion may be inadequate to train them for tasks involving long-term dependencies. Assuming hyperbolic attractors are used to store state information, we found that either the system would not be robust to input noise or would not be efficiently trainable by gradient descent when long term context is required. Note that the theoretical results presented in this paper hold for any error criterion and not only for the mean square error criterion. Two simple generalizations are obtained as follows. As mentioned in the analysis section, a periodic attractor can be transformed into a fixed point by subsampling time with the period of the attractor. Hence if corresponding fixed point is stable, it is also hyperbolic and our results hold in that case as well. Another interesting case is the situation in which the system doesn't remain long near an attractor, but rather, jumps rapidly from one stable (hyperbolic) attractor to another. This would arise for example if the continuous dynamics can be made to correspond to the discrete dynamics of a deterministic finite-state automaton. In that case, our results hold as well since the norm of Jacobian of the map derivatives near each of the attractor is less than one ($\frac{\partial a_t}{\partial a_{t-1}} = M'(a_{t-1})$). What remains to be shown is that similar problems occur with chaotic attractors, i.e., that either the gradients vanish or the system is not robust to input noise. It is interesting to note that related problems of vanishing gradient may occur in deep feedforward networks (since a recurrent network unfolded in time is just a very deep feedforward network with shared weights).

The result presented here does *not* mean that it is impossible to train a recurrent network on a particular task. It says that gradient descent becomes increasingly inefficient when the temporal span of the dependencies increases. Furthermore, for a given problem, there are sometimes ways to help the training by setting the network connectivity and initial weights (and even constraints on the weights) using prior knowledge (e.g., [8], [11]). For some tasks, it is also possible to present a variety of examples of the input/output dependencies, including short-term dependencies which are sufficient to infer similar but longer term dependencies. For example, in the Latch problem or the Parity problem, if we start by training with short sequences, the system rapidly settles in the correct region of parameter space.

A better understanding of this problem has driven us to design alternative algorithms, such as the time-weighted pseudo-Newton and the discrete error propagation algorithms. In the first case, we consider the instantiation of the weights at different times as different variables and consider the curvature of the cost function for these variables. This information is used to weight the gradient contributions for the different times in such a way as to make larger steps in directions where the cost function is flatter. The discrete error propagation algorithm propagates error information through a mixture of discrete and continuous elements. The gradient is locally quantized with a stochastic decision rule that appears to help the algorithm in locally searching for solutions and getting out of local minima. We have compared these algorithms with standard optimization algorithms on toy tasks on which the temporal span of the input/output dependencies could be controlled. The very preliminary results we obtained are encouraging and suggest that there may be ways to reconcile *learning* with *storing*. Good solutions to the challenge presented here to learning long-term dependencies with dynamical systems such as recurrent networks may have implications for many types of applications for learning systems, e.g., in language related problems, for which long-term dependencies are essential in order to make correct decisions.

Appendix

Proof of Theorem 1:

By hypothesis and definition of norm, $\exists\ u$ s.t. $\|u\| = 1$ and $\|M'(x)u\| > 1$. The Taylor expansion of M at x for small value of λ is:

$$M(x + \lambda u) = M(x) + M'(x)\lambda u + O(\|\lambda u\|^2) \tag{11}$$

Since U(x) is an open set, $\exists\ \lambda$ s.t. $\|O(\|\lambda u\|^2)\| < \lambda(\|M'(x)u\| - 1)$ and $x + \lambda u \in U(x)$. Letting $y = x + \lambda u$ we can write $\|M(y) - M(x) - M'(x)\lambda u\| = \|O(\|\lambda u\|^2)\| < \lambda\|M'(x)u\| - \lambda$ or $-\|M(y) - M(x) - M'(x)\lambda u\| + \|M'(x)\lambda u\| > \lambda$. This implies using the triangle inequality $\|M(y) - M(x)\| > \lambda = \|x - y\|$.$\square$

Proof of Theorem 3:

Let us denote by ρ_t the radius of the "uncertainty" ball $\Phi_t = \{a : \|\tilde{a}_t - a\| < \rho_t\}$ in which we are sure to find a_t, where $\tilde{a}_t$ gives the trajectory of the autonomous system. Let us suppose that at time t, $\rho_t < d$ (this is certainly true at time 0, when $\tilde{a}_0 = a_0$). By Lagrange's mean value theorem and convexity of D_t, $\exists z \in D_t$ s.t. $\|M(x) - M(y)\| \leq |M'(z)|\|x - y\|$, but $|M'(z)| < \lambda_t$ by hypothesis. Then by the contraction theorem [20] we have $\rho_{t+1} \leq \lambda_t d + b_t$. Now by hypothesis we have $b_t = (1 - \lambda_t)d$, so $\rho_{t+1} < d$. The conclusion of the theorem is then obtained since $\tilde{a}_t \in D_t$ by our construction above and $\tilde{a}_t$ converges to X for $t \to \infty$. $\Box$

Proof of Theorem 4:

By hypothesis and definitions 4 and 2, $\left|\frac{\partial a_\tau}{\partial a_{\tau-1}}\right| = |M'(a_{\tau-1})| < 1$ for $\tau > 0$, hence $\frac{\partial a_t}{\partial a_0} \to 0$ as $t \to \infty$ $\Box$.

One could however ask what happens when a_t remains near the boundary between two basins:

Lemma 1 *Suppose that for $t > 0$, a_0 and u_t are such that a_t remains on the boundary between two basins of attraction for attractors X_1 and X_2, and there exists an infinitesimal change in a_0 yielding the state into either X_1 or X_2 and remaining there. Then* $\lim_{t\to\infty} \left|\frac{\partial a_t}{\partial a_0}\right| = \infty$.

It appears that the hypotheses of this lemma will rarely be satisfied, for two reasons. Firstly, the system evolves in discrete time, making it improbable to obtain a_t precisely on the boundary surface. Second, in order to stay on that surface, say $S(a_t) = 0$, u_t must satisfy the equation $S(M(a_{t-1}) + u_t) = 0$. Hence the submanifold of values of u_t in $\mathbf{R}^m$ which satisfy this equation has dimension $m - 1$, thus having null measure.

Generalization to a Projection of the State

The results obtained so far can be generalized to the case when a projection Pa_t of the state a_t converges to an attractor under a map M. This would be the case for example when a subset of the hidden units in a recurrent network participate directly in the dynamics of a stable attractor. Let P and R be orthogonal projection matrices such that

$$
\begin{aligned}
& a_t = P^+ z_t \ + \ R^+ y_t \\
& z_t = Pa_t; \quad y_t = Ra_t \\
& PR^+ = 0; \quad RP^+ = 0
\end{aligned}
\tag{12}
$$

where A^+ denotes the right pseudo-inverse of A, i.e. $AA^+ = I$. Suppose M is such that P can be chosen so that z_t converges to an attractor Z with the dynamics $z_t = M_P(z_{t-1}) = P_t M(P^+ z_t + R^+ y_t)$ for any y_t. Then we can specialize all the previous definitions, lemmas and theorems to the subspace z_t. When we conclude with these results that $\frac{\partial z_t}{\partial z_0} \rightarrow 0$, we can infer that $\frac{\partial a_t}{\partial a_0} \rightarrow R^+ \frac{\partial y_t}{\partial y_0} R$, i.e., that the derivatives of a_t with respect to a_0 depend only on the projection of a on the subspace Ra. Hence the influence of changes in the projection of a on the subspace Pa is ignored in the computation of the gradient with respect to W, even though non-infinitesimal changes in Pa could yield very different results (i.e., jumping into a different basin of attraction). Although training can now proceed in some directions, the effect of parameters that influence detecting and storing events for the long-term or switching between stable states is still not taken very much into account.

References

[1] P.L. Bartlett, and T. Downs, "Using Random Weights to train Multilayer Networks of Hard-Limiting Units," *IEEE Transactions on Neural Networks*, vol. 3, no. 2, 1992, pp. 202–210.

[2] S. Becker and Y. Le Cun, "Improving the convergence of back-propagation learning with second order methods", *Proceedings of the 1988 Connectionist Models Summer School*, (eds. Touretzky, Hinton and Sejnowski), Morgan Kaufmann, pp. 29–37.

[3] Y. Bengio, P. Frasconi, P. Simard, "The problem of learning long-term dependencies in recurrent networks", invited paper at the *IEEE International Conference on Neural Networks 1993*, San Francisco, IEEE Press.

[4] Y. Bengio, "Artificial Neural Networks and their Application to Sequence Recognition," Ph.D. Thesis, McGill University, (Computer Science), 1991, Montreal, Qc., Canada.

[5] Y. Bengio, R. De Mori, G. Flammia, and R. Kompe, "Global Optimization of a Neural Network - Hidden Markov Model Hybrid," *IEEE Transactions on Neural Networks*, vol. 3, no. 2, 1992, pp. 252–259.

[6] A. Corana, M. Marchesi, C. Martini, and S. Ridella, "Minimizing Multimodal Functions of Continuous Variables with the Simulated Annealing Algorithm", *ACM Transactions on Mathematical Software*, vol. 13, no. 13, Sept. 1987, pp. 262–280.

[7] P. Frasconi, M. Gori, and G. Soda, "Local Feedback Multilayered Networks", *Neural Computation* **3**, 1992, pp. 120–130.

[8] P. Frasconi, M. Gori, M. Maggini, and G. Soda, "Unified Integration of Explicit Rules and Learning by Example in Recurrent Networks," *IEEE Trans. on Knowledge and Data Engineering*, in press.

[9] R.J. Gaynier, and T. Downs, "A Method of Training Multi-layer Networks with Heaviside Characteristics Using Internal Representations," *IEEE International Conference on Neural Networks 1993*, San Francisco, pp. 1812–1817.

[10] M. Gori, Y. Bengio and R. De Mori, "BPS: a learning algorithm for capturing the dynamic nature of speech," *Proc. IEEE Int. Joint Conf. on Neural Networks*, Washington DC, 1989, pp. II.417-II.424.

[11] C.L. Giles and C.W. Omlin, "Inserting Rules into Recurrent Neural Networks", *Neural Networks for Signal Processing II, Proceedings of the 1992 IEEE workshop*, (eds. Kung, Fallside, Sorenson and Kamm), IEEE Press, pp. 13-22.

[12] T. Grossman, R. Meir and E. Domany, "Learning by choice of internal representation", Neural Information Processing Systems 1, (ed. D.S. Touretzky), pp. 73–80.

[13] S. Kirkpatrick, C.D. Gelatt, M.P. Vecchi, "Optimization by simulated annealing", *Science 220*, 4598 (May 1983), pp.671–680.

[14] Kuhn G., "A first look at phonetic discrimination using connectionist models with recurrent links." CCRP – IDA SCIMP working paper No.4/87, Institute for Defense Analysis, Princeton, NJ, 1987.

[15] K.J. Lang and G.E. Hinton, "The development of the Time-Delay Neural Network architecture for speech recognition", Technical Report CMU-CS-88-152, Carnegie-Mellon University, 1988.

[16] Y. Le Cun, "Learning Processes in an Asymmetric Threshold Network", in *Disordered systems and biological organization*, (eds. Bienenstock, E. and Fogelman-Soulié, F. and Weisbuch, G.), Springer-Verlag, Les Houches, France, 1986, pp. 233-240.

[17] C.M. Marcus, F.R. Waugh, and R.M. Westervelt, "Nonlinear Dynamics and Stability of Analog Neural Networks", *Physica D* **51** (special issue), 1991, pp. 234-247.

[18] Mozer M.C. "A focused back-propagation algorithm for temporal pattern recognition", *Complex Systems, 3*, 1989, pp. 349-391.

[19] Mozer M.C., "Induction of multiscale temporal structure", Advances in Neural Information Processing Systems 4, (eds. Moody, Hanson, Lippman), Morgan Kaufmann, 1992, pp. 275-282.

[20] Ortega J.M. and Rheinboldt W.C. *Iterative Solution of Non-linear Equations in Several Variables and Systems of Equations*, Academic Press, New York, 1960.

[21] Rohwer R. "The 'Moving Targets' Training Algorithm", Advances in Neural Information Processing Systems 2, (ed. Touretzky), Morgan Kaufmann, 1990, pp. 558-565.

[22] D.E. Rumelhart, G.E. Hinton and R.J. Williams, "Learning internal representation by error propagation," *Parallel Distributed Processing* volume 1. Rumelhart D.E. and McClelland J.L. (eds.), Bradford Books, MIT Press, 1986, pp. 318–362.

[23] Williams R.J. and Zipser D. "A learning algorithm for continuously running fully recurrent neural networks", *Neural Computation, 1*, 1989, pp. 270-280.

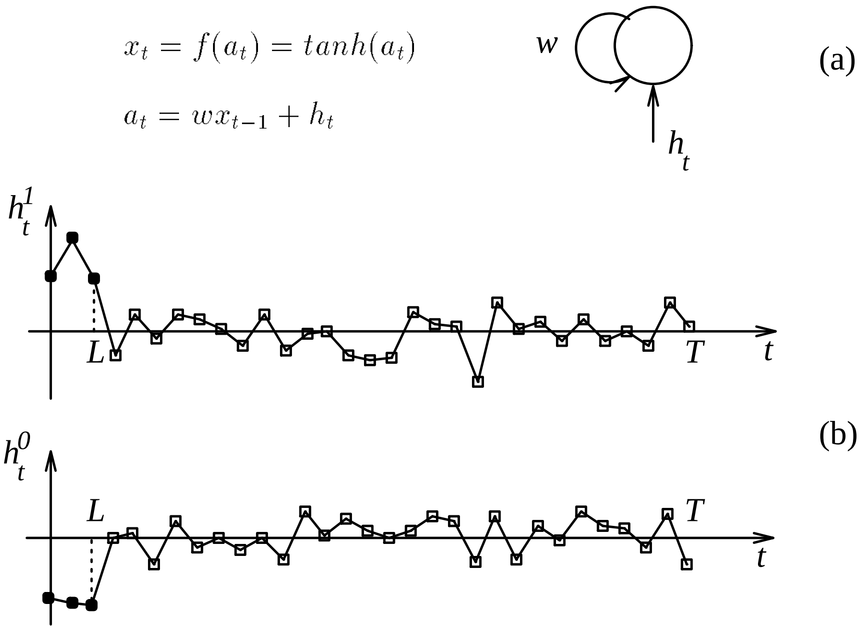

Figure 1: a) Latching recurrent neuron. b) Sample input to the recurrent neuron. The trainable values $h_1, \ldots, h_L$ (marked in bold) have been tuned by one of the successfully learning simulations.

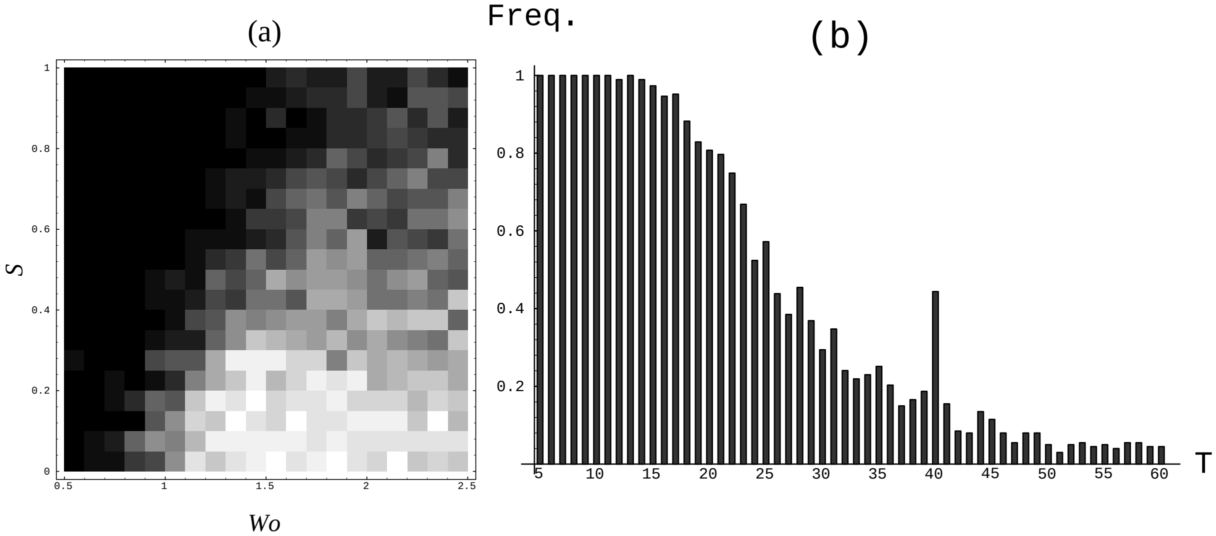

Figure 2: Experimental results for the minimal problem. a) Density of training convergence with respect to the initial weight w_0 and the noise variance s (white $\Rightarrow$ high density), with $L = 3$ and $T = 20$. b) Frequency of training convergence with respect to the sequence length T, (with noise variance $s = 0.2$, and initial weight $w_0 = 1.25$).

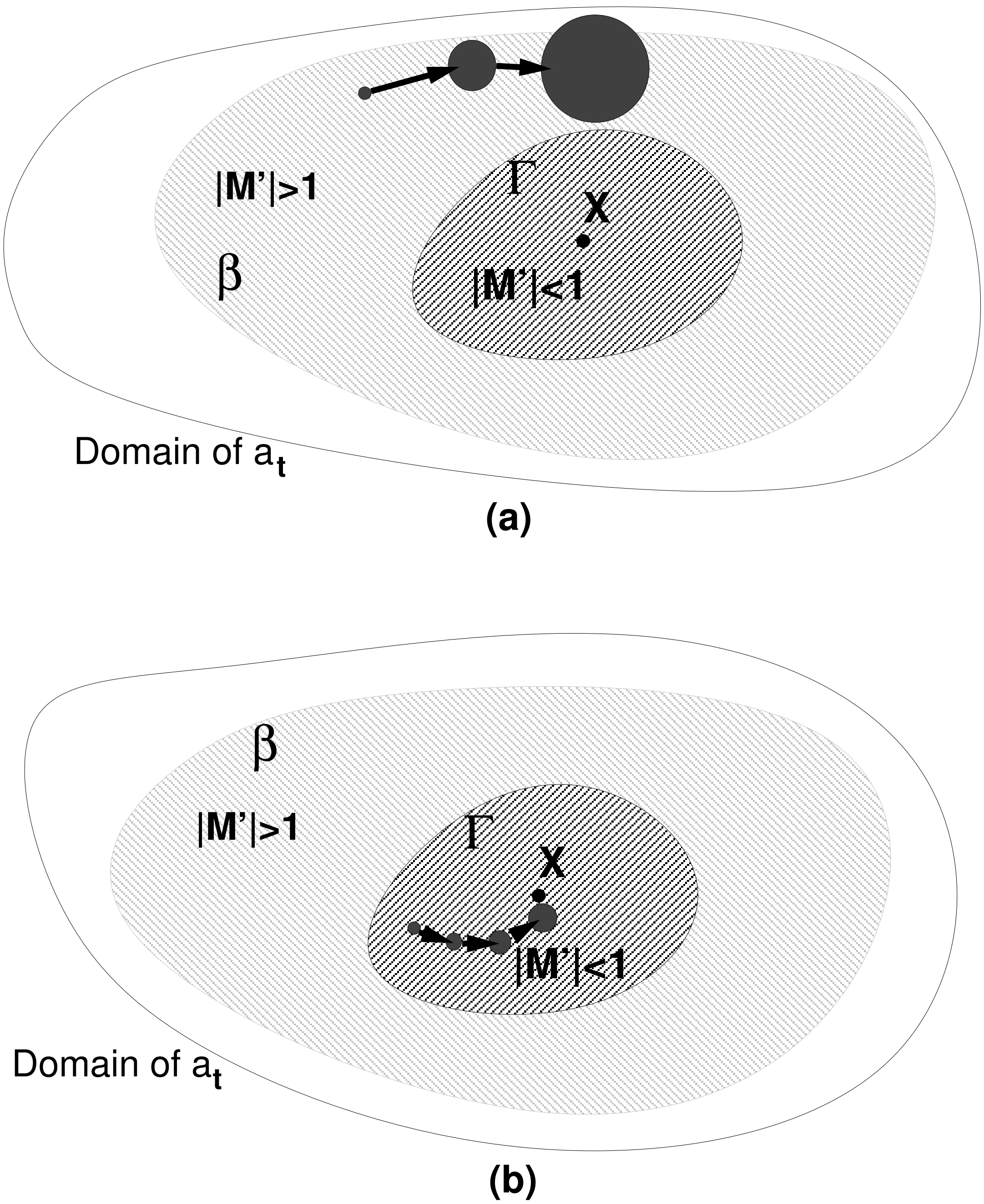

Figure 3: Basin of attraction (β), reduced attracting set (Γ) of an attractor X. Ball of uncertainty grows exponentially (a) outside Γ, but is bounded (b) inside Γ.

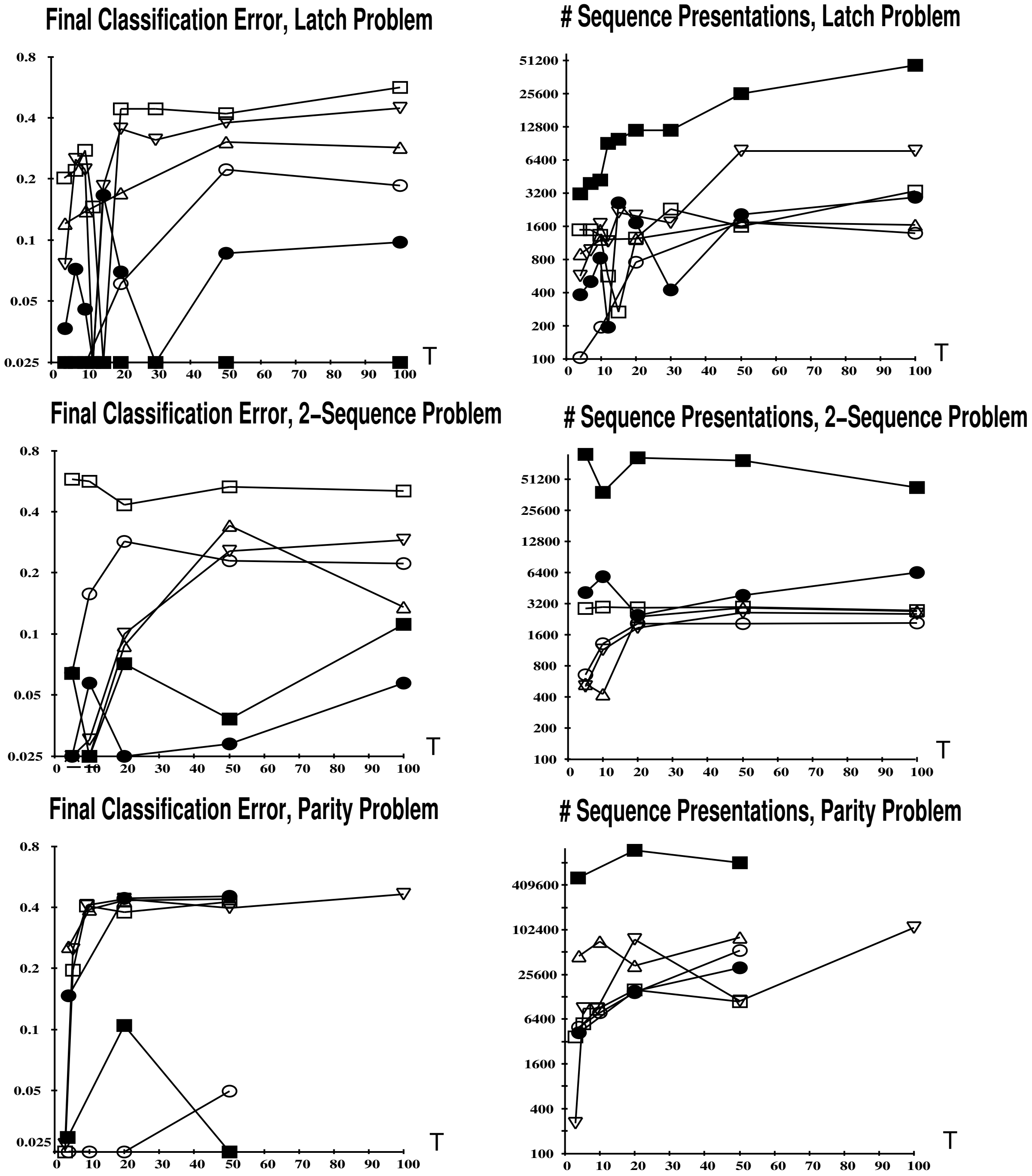

Figure 4: Comparative simulation results for: □ standard back-propagation, ▽ pseudo-Newton, △ time-weighted pseudo-Newton, ◯ discrete error propagation, • multi-grid random search, ▮ simulated annealing. The horizontal axis (T) represents maximum sequence length. On the left, the vertical axis represents classification error after training; on the right, the number of sequence presentations to reach a stopping criterion.