The "Moving Targets" Training Algorithm

Richard Rohwer
Centre for Speech Technology Research
University of Edinburgh
80 South Bridge
Edinburgh EH1 1HN
rr@uk.ac.ed.eusip

Abstract

A simple method for training the dynamical behavior of a neural network is derived. It is applicable to any training problem in discrete-time networks with arbitrary feedback. The method resembles back-propagation in that it is a least-squares, gradient-based optimization method, but the optimization is carried out in the hidden part of state space instead of weight space. A straightforward adaptation of this method to feedforward networks offers an alternative to training by conventional back-propagation. Computational results are presented for simple dynamical training problems, with varied success. The failures appear to arise when the method converges to a chaotic attractor. A patch-up for this problem is proposed. The patch-up involves a technique for implementing inequality constraints which may be of interest in its own right.

1 Introduction

This paper presents an algorithm for training the dynamical behavior of a discrete-time neural network model. In some ways this method is similar to back-propagation. It is based on the calculation of the gradient of an error measure, the error measure is a sum of squares, and the algorithm is intended for pattern classification. But this algorithm is very different in other ways. It applies to totally connected networks with full feedback, not just feedforward networks. It is intended for pattern completion as well as pattern classification, and makes no formal distinction between the two. It is applicable to dynamical patterns in fully recurrent networks and presents a radically altered framework for treating static patterns in feedforward networks. The central idea is to avoid setting the minimization problem in weight space by transforming the problem so that the minimization takes place in state-trajectory space. Werbos [21] used the term "moving targets" to describe the qualitative idea that a network should set itself intermediate objectives, and vary these objectives as they are found to be more or less helpful for achieving overall objectives, or more or less achievable themselves. This algorithm of (by accidental coincidence) the same name can be regarded as a quantitative realization of this qualitative idea.

A minimization approach to dynamical problems is provided by a simple extension to the back-propagation method, back-propagation through time [17]. This method is reasonably successful as long as it does not have to react to distant temporal correlations. The method of Rohwer and Forrest [15] for training state transitions is also vulnerable to this criticism. The forward propagation of derivatives [22, 7] may overcome this problem. Its main defect is a large memory requirement, cubic in the number of nodes, for recording the derivative of each node value with respect to each weight. Perlmutter [8] has presented methods for training the

dynamics of continuous-time networks. These networks are not readily trainable by the moving targets method.

It is known that a feedforward network can implement a wide variety of nonlinear filters, but little is known about whether a similarly rich set of dynamical behaviors can be achieved. If this were not the case, then dynamical training algorithms would be shackled by the limitations of the underlying model. Numerical studies of the attractors produced with random networks such as provide encouragement (but not proof) that any such limitations are not severe [11].

The problem is formalized in section 2, and the general scope of the formalism is noted. The "moving targets" training method is defined in section 4, concluding a discussion of similar methods in section 3. The main formulas required are derived and discussed in sections 5 and 6. Estimates for the computational requirements of the method are given in 7, and compared to the computational requirements for back-propagation. The analogous method for feedforward networks is outlined in section 8. Section 9 takes up two problems with the method and discusses solutions. The proposed solutions involve a treatment of inequality constraints which may be of interest for other purposes. Computational results are given in 10.

2 Notation and statement of the training problem

Consider a neural network model with feedback as a dynamical system in which the dynamical variables x_{it} change with time according to a dynamical law given by the mapping

$$\left.\begin{array}{rcll} x_{it} & = & \sum_j w_{ij} f(x_{j,t-1}) & i > 0 \\ x_{0t} & = & \text{bias constant} & \end{array}\right\} \tag{1}$$

unless specified otherwise. The *weights* w_{ij} are arbitrary parameters representing the connection strength from node j to node i. f is an arbitrary differentiable function, usually taken to be the logistic:

$$f(x) = 1/(1 + e^{-x}). \tag{2}$$

Let us call any given variable x_{it} the *"activation"* on node i at time t. It represents the total input into node i at time t.

We have made a small departure from the usual notation in which the dynamical variables are the node *"outputs"* $y_{it} = f(x_{it})$. In the usual formulation, a *"bias node"* $y_{0t} = 1$ is used to give the weights w_{i0} the significance of activation thresholds. For complete equivalence we must take the bias constant to be $f^{-1}(1)$, which is infinite for the logistic. However the same dynamics results from the choice $y_{0t} = 1 - \epsilon$, for any small positive ϵ, with a corresponding rescaling of the threshold weights.

In normal back-propagation, a network architecture is defined which divides the network into input, hidden, and target nodes. The moving targets algorithm makes itself applicable to arbitrary training problems by defining analogous concepts in a manner dependent upon the training data, but independent of the network architecture. Let us call a node-time pair an *"event"*. To define a training problem, the set of all events must be divided into three disjoint sets, the *input* events I, *target* events T, and *hidden* events H. For every input event $(it) \in I$, we require *training data* X_{it} with which to overrule the dynamical law (1) using

$$x_{it} = X_{it} \quad (it) \in I. \tag{3}$$

(The bias events $(0t)$ can be regarded as a special case of input events.) For each target event $(it) \in T$, we require training data X_{it} to specify a desired activation value for event $(0t)$. No

notational ambiguity arises from referring to input and target data with the same symbol X because I and T are required to be disjoint sets. The training data says nothing about the hidden events in H.

Considerable flexibility in the definition of the training problem arises from the fact that the time dimension can be used to separate the three classes of events from each other. Suppose, for example, that a sequence of inputs is to be recognized as a particular phoneme by having a particular node "turn on" whenever the sequence goes by. If it were necessary to specify a target value for this node at every time step, then one would have to make some rather arbitrary decisions about what target value to assign when only some of the data needed to recognize the phoneme has been presented. For instance, Watrous [19] uses various functions to interpolate between 0 and 1 during the course of a sequence. The moving targets method allows the node to be classified as hidden during these ambiguous times, and as a target otherwise.

In general, different sequences will result if the dynamical law (1) is applied to different initial conditions x_{i0}. If the network is always initialized to a particular state, then these initial events are classified as inputs. If no such training data exists, they can be classified as hidden events, in which case the training algorithm will generate values for them which should be used when the network is run. If they are given target values, the network will seek an optimal compromise between the given values and others which might be needed to obtain the desired future behavior.

3 Related least-squares training methods

The least-squares training methods, typified by back-propagation, proceed by minimization of an *"error"* function such as the *"output deficit"*

$$E_{\rm od} = \tfrac{1}{2} \sum_{(it)\in T} \{y_{it} - Y_{it}\}^2, \tag{4}$$

where $y_{it} = f(x_{it})$ and $Y_{it} = f(X_{it})$. A minimization method employing the derivatives of this error with respect to the weights is used. The conjugate gradient method was used in the work reported here [10]. The moving targets method requires an *"activation deficit"* error function:

$$E_{\rm ad} = \tfrac{1}{2} \sum_{(it)\in T} \{x_{it} - X_{it}\}^2. \tag{5}$$

If either of these functions is zero, then so is the other. To this extent they are equivalent. However if the minima are nonzero, they are generally different, and represent a different tradeoff of errors between different target events. These differences are discussed quantitatively in [14]. The differences are relatively small when the activations are in the linear region of the logistic ($|X_{it}| \not\gg 1$), but can be seriously different in the saturated regions. Unfortunately it is (4) which represents the more sensible trade-offs in these cases. But since (5) is strongly preferable for a technical reason stated below, this problem is addressed in an indirect and somewhat inelegant manner presented in section 9.

Back-propagation and related methods for networks with feedback [15, 16, 1, 9] are based on a calculation of the derivative of the error (4) with respect to the weights. Let us trace the weight dependence of the error through the three different types of events by expanding the term x_{it} in (5):

$$E = \tfrac{1}{2} \sum_{(it)\in T} \left\{ \sum_{j\in I_{t-1}} w_{ij} f(X_{j,t-1}) + \sum_{j\in H_{t-1}} w_{ij} f(x_{j,t-1}) + \sum_{j\in T_{t-1}} w_{ij} f(x_{j,t-1}) - X_{it} \right\}^2 . \tag{6}$$

The subscripts on I, H, and T, denote the nodes of the input, hidden, or target events respectively at the indicated time step. Commas have been used to separate subscripts where confusion is possible. Besides the explicit w-dependence in each sum, there is an implicit dependence through the factor $x_{j,t-1}$ in the final two sums. The first sum does not raise this complication because it has an X instead of an x. Using the chain rule to follow this dependence from time step to time step back to the initial conditions leads to the *"back-propagation through time"* method [17].

Besides being the most complicated part of the calculation, the recursion of the chain rule through time is difficult to do accurately because with each time step the terms involved are multiplied by the derivative of the logistic (2). This derivative has a maximum of $\frac{1}{4}$. The terms are also multiplied by the weights, but large weight magnitudes produce saturated node values (unless there are delicate cancellations) which in turn produce exponentially small derivatives of the logistic. Therefore the method has difficulties with problems requiring attention to correlations of inputs separated by several time steps.

The basic strategy in this paper is to stop the chain-rule recursion by turning x's into X's in (6). This is easier to do for the third sum than the second, because training data exists for the target events but not for the hidden events. By simply capitalizing the 'x' in the third term, we arrive at the *teacher forcing* error function [16, 22]. If a solution can be found for which the teacher forcing error vanishes exactly, then the usual error will vanish as well, because a feed-forward pass through the network will produce x's which happen to equal their corresponding X's anyway.

In addition to simplifying the calculation, one can argue that the derivatives computed from the teacher forcing error point more directly at a solution than the usual derivatives do. Suppose that on a particular time step t, a partially trained network is performing poorly on target event (it). In either method, the next adjustment of the weights will take account of error produced at (it), which reflects the inadequacy of the weight matrix for getting the trajectory right before time t. The usual calculation of errors produced after time t assumes that the network will produce the incorrect value x_{it} at time t. Unless the training is doomed to fail anyway, this assumption is false; the trained network will actually produce the correct value X_{it}. Therefore corrections intended to reduce errors produced after time t should be made using the assumption that the trained network will perform correctly at time t, which is precisely what teacher forcing accomplishes.

This argument has a loophole which will become a major issue in section 9. The trouble is that the assumption that the trained network will work correctly at every time step is usually not *exactly* true, even though it provides the only reasonable guess, early in training, for the eventual trajectory of the trained network. The trained network may produce a chaotic attractor, in which case small errors on an early time step will become large errors later. Even if training is perfect, the network cannot be expected to work as intended with inputs slightly different from the training data. There is nothing to prevent the usual method from training to a chaotic attractor as well, but it has the merit of basing future error estimates on the past trajectory that the network actually produces.

4 The "moving targets" method

Now we come to the main idea of this paper. How can we change the 'x' into a 'X' in the second term of (6) without having any training data to specify target values for X_{it} with $(it) \in H$? The trick is to declare that the hidden X's shall be *independent variables* in the minimization problem. Let us pretend for explanatory purposes that there were desired values for events in

H after all, so that the union of H and T would be the set of target events. Then the teacher forcing error function would be:

$$E = \tfrac{1}{2} \sum_{(it) \in T \cup H} \left\{ \sum_j w_{ij} f(X_{j,t-1}) - X_{it} \right\}^2 . \tag{7}$$

This is a function of the weights w_{ij}, and because there are no x's present, the full dependence on w_{ij} is explicitly displayed. We do not actually have desired values for the X_{it} with $(it) \in H$. But any values for which weights can be found which make (7) vanish would be suitable. Therefore let us regard E as a function of both the weights and the "moving targets" $X_{it}, (it) \in H$. This is the main trick. The moving targets are *independent variables*, like the weights. They are not computed from 1. The derivatives with respect to all of the independent variables can be computed and plugged into a standard minimization algorithm. (Note that "moving" of targets in the algorithm's name does not refer to changes during the time in which the network operates, but during the time in which the minimization algorithm operates. The former resembles a spatial coordinate from the viewpoint of the latter.)

The reason for preferring the activation deficit form of the error (5) to the output deficit form (4) is that the activation deficit form makes (7) purely quadratic in the weights. Therefore the equations for the minimum,

$$dE/dw_{ij} = \partial E/\partial w_{ij} = 0, \tag{8}$$

form a linear system, the solution of which provides the optimal weights for any given set of moving targets. Therefore these equations might as well be used to define the weights as functions of the moving targets, thereby making the error (7) a *function of the moving targets alone.*

At this point the hidden activations x and the weights w have undergone a kind of role reversal. In the usual formulation of back propagation, the error depends on the activations and the weights, but the weights are the only independent variables because the activations depend on the weights in a complicated way involving recursion. In the moving targets formulation, the error again depends on activations and weights, but the activations are the only independent variables because the weights depend on the activations in a relatively simple way involving a system of linear equations.

5 The optimal weights

In this and the following section we shall dispense with the chores of computing the formula for the weights in terms of the moving targets, and derivatives of (7) with respect to the moving targets.

Let us abbreviate

$$e_{it} = \sum_j w_{ij} f(X_{j,t-1}) - X_{it}, \tag{9}$$

and let

$$\chi_{it} = \begin{cases} 1 & (it) \in T \cup H \\ 0 & (it) \notin T \cup H \end{cases} \tag{10}$$

so that the error (7) can be written as an unrestricted sum:

$$E = \tfrac{1}{2} \sum_{it} \chi_{it} \left\{ \sum_j w_{ij} f(X_{j,t-1}) - X_{it} \right\}^2 . \tag{11}$$

The linear system which defines the weights is

$$0 = dE/dw_{ab} = \sum_{it} \chi_{it} e_{it} \delta_{ia} \sum_j \delta_{jb} f(X_{j,t-1}) \tag{12}$$

$$= \sum_t \chi_{at} e_{at} Y_{b,t-1} \tag{13}$$

$$= \sum_j w_{aj} \sum_t \chi_{at} Y_{j,t-1} Y_{b,t-1} - \sum_t \chi_{at} X_{at} Y_{bt}. \tag{14}$$

The Kronecker δ ($\delta_{ij} = 1$ if $i = j$ and 0 if $i \neq j$) has been used. For each node a, define the matrix M as a correlation matrix of the node outputs:

$$M_{ij}^{(a)} = \sum_t \chi_{at} Y_{i,t-1} Y_{j,t-1} \tag{15}$$

The inverses $M^{(a)-1}$ of $M^{(a)}$, defined by $\sum_k M_{ik}^{(a)} M_{kj}^{(a)-1} = \delta_{ij}$, solve the linear system (14), providing the formula for w:

$$w_{ij} = \sum_k \left(\sum_t \chi_{it} X_{it} Y_{k,t-1} \right) M_{kj}^{(i)-1}. \tag{16}$$

In the event that any of the matrices M are singular, a pseudo-inversion method such as singular value decomposition [10] can be used to define a unique solution among the infinite number available.

From equation (16) it would seem that a different matrix inversion problem must be solved for every node. In general this is true, but in practice the symmetry of typical problems makes only a few inversions necessary. Typically, many nodes switch between being input and non-input nodes at the same times, so χ_{at} in (15) and (16) usually represents the same temporal pattern for several different nodes. For example, it would be normal practice to designate some nodes as inputs for all time, making χ vanish for all time. From the way χ enters (16) one could conclude that connections leading into always-input nodes should be set to 0, which is quite sensible. Note that (15) implies that the corresponding M matrices are all singular, however, so actually this means that weights leading into the always-input nodes are undefined. In a way, this is even more sensible, because the choice of these weight values has no effect on the network. The main reason for changing a node from input to non-input is to initialize the network for learning about a new sequence. But this operation usually involves all the non-input nodes at the same time; every such node is set to some standard value during the same time step. This covers a wide class of problems, including the test problems discussed below, and requires only one matrix inversion.

6 The gradient in moving target space

Next let us compute the derivatives of the error with respect to the moving targets. Let

$$f'_{it} = \left. \frac{df(x)}{dx} \right|_{x=X_{it}}. \tag{17}$$

Then

$$\frac{dE}{dX_{as}} = \sum_{it} \chi_{it} e_{it} \left[\sum_j \frac{dw_{ij}}{dX_{as}} Y_{j,t-1} + \sum_j w_{ij} f'_{j,t-1} \delta_{ja} \delta_{s,t-1} - \delta_{ia} \delta_{st} \right] \tag{18}$$

$$= \sum_{ij} \frac{dw_{ij}}{dX_{as}} \sum_t \chi_{it} e_{it} Y_{j,t-1} + \sum_i \chi_{i,s+1} e_{i,s+1} w_{ia} f'_{as} - \chi_{as} e_{as}. \tag{19}$$

The calculation of the derivative of the weights with respect to the moving targets in the first term of (19) would be a somewhat troublesome task, so it is fortunate that its coefficient $\sum_t \chi_{it} e_{it} Y_{j,t-1}$ is zero by equation (13). This phenomenon, the *"orthogonality of the signal to the error"* is well known in electrical engineering [13]. If the weights are defined by (8), then this will occur for any definition of the error because

$$\frac{dE}{dX_{as}} = \sum_{ij} \frac{\partial E}{\partial w_{ij}} \frac{dw_{ij}}{dX_{as}} + \frac{\partial E}{\partial X_{as}} \tag{20}$$

$$= \frac{\partial E}{\partial X_{as}}. \tag{21}$$

The nonzero terms of the gradient in moving-target space are therefore simply

$$\frac{dE}{dX_{as}} = \sum_i \chi_{i,s+1} e_{i,s+1} w_{ia} f'_{as} - \chi_{as} e_{as}. \tag{22}$$

This is a pleasantly simple formula with clear intuitive significance. The second term says the moving targets at time s should be moved toward their computed values, thus reducing the error on the current time step. The first term says that these same moving targets should be moved so as to reduce the error on the following time step.

7 Computational requirements

In this section we compare the computational requirements of a gradient evaluation in moving target space to a gradient evaluation in weight space using back-propagation through time.

For simplicity, and to enable a direct comparison with back-propagation through time, let us suppose that each node remains either an input, hidden, or target node for all time. Let there be I input nodes, H hidden nodes, and T target nodes, making a total of $N = I + T + H$ nodes. Let there be P time steps. A forward propagation for one time step in order to update the hidden and target node outputs requires about N(H+T) multiply-and-add operations. Let us neglect the mere (H+T) logistic evaluations. Back-propagation through one time step requires roughly the same number of operations. Therefore the completion of one forward and one backward pass requires roughly

$$T_{\text{bp}} = 2PN(H + T) \tag{23}$$

operations.

The moving targets method also requires P forward and backward passes at a cost of $2PN(H + T)$ (although unlike back-propagation, activations are reset to their (data-specified or moving) target values at each time step). At first glance it may seem that the formation of the matrix M in (15) requires N^2P operations, but it is possible to do better than this. Only those blocks of M which involve the hidden nodes need to be updated; the others can be initialized and left alone. Furthermore M is symmetric, so only half the terms need to be computed. An inspection of (15) written as a product of matrices in block form shows that only about $PH(N - H/2)$ operations are needed. By similar reasoning, about $PH(N + H)$ operations are needed to do the first matrix product in (16). The matrix inversion requires about $\frac{4}{3}N^3$ if done by LU decomposition, more by a small factor if done by singular value decomposition [10]. The final matrix product in (16) requires about $N^2(T + H)$ operations. The total for a computation of w is therefore

$$T_{\text{mt}} = 4PN(H + T) + \tfrac{1}{2}PH^2 + (H + T)N^2 + \tfrac{4}{3}N^3. \tag{24}$$

After a little algebra, we conclude that the time required for a gradient evaluation by the moving targets method is a factor of

$$\frac{T_{\mathrm{mt}}}{T_{\mathrm{bp}}} = 2\frac{H}{H+T}\left(1+\frac{1}{16}\frac{H}{N}\right)+\frac{4}{3}\frac{N}{P}\left(\frac{7}{4}+\frac{I}{T+H}\right) \tag{25}$$

times the time required by backpropagation through time. The first term is strictly less than $2\frac{1}{8}$, and is smallest when there are many more target nodes than hidden nodes. The second term is the most important for evaluating the method. It is small if there are many more time steps than nodes, and large in the opposite situation. It can also be large if most of the nodes are inputs.

The moving targets calculation is therefore not significantly more difficult than a comparable back-propagation calculation, if there are many fewer nodes than time steps, and at least a modest number of hidden and target nodes compared to the number of inputs.

This calculation does not address the question of how many gradient evaluations will be needed to find a minimum. This is difficult to estimate, because it concerns the nature of the error surfaces in weight space and moving-target space. We scarcely know how to frame incisive questions on this topic, and have even less idea of how to answer them. But one thing we can expect is that the difficulty of the problem will increase as the number of variables in the minimization problem increases. There are $N(H+T)$ variables in the backpropagation problem, and PH in the moving-targets problem. Therefore there may be difficulties with using lots of training data. However, if many subsequences of the training data are similar to many others, then the minimization problem may not become proportionately more difficult as the amount of training data increases. Alternatively, training might be done on a small part of the training data initially, and resumed with more data for fine-tuning.

8 Feedforward networks

The basic ideas used in the moving targets algorithm can be applied to feedforward networks to provide an alternative method to back-propagation. The hidden node activations for each training example become the moving target variables. The calculation involves a different matrix inversion problem for each layer of weights. This method is analogous to one derived by Grossman, Meir, and Domany [4] for networks with discrete node values. Birmiwal, Sarwal, and Sinha [2] have developed an algorithm for feedforward networks which incorporates the use of hidden node values as fundamental variables and a linear system of equations for obtaining the weight matrix. Their algorithm differs from the feedforward version of moving targets mainly in the (innessential) use of a specific minimization algorithm which discards most of the gradient information except for the signs of the various derivatives. Heileman, Georgiopoulos, and Brown [5] also have an algorithm which bears some resemblance to the feedforward version of moving targets.

Computational trails are planned for the feedforward version but results are not yet available. Unfettered by sober fact, let us note a philosophical reason for expecting moving targets to outperform backpropagation in feedforward nets! The moving targets method directly varies the internal representations present on the hidden nodes; the very heart of the problem according to current dogma [17]. The weight matrix variations are largely subservient to the moving target variations, and are accomplished by a direct calculation using well-studied methods for solving linear systems; methods which contribute to the impressive speed of radial basis function methods [12, 3]. Additionally, it is possible to use prior knowledge about a problem (for example, a principle components analysis of the training data [20, 18]) to initialize the internal

representations. Finally, this method may be more successful for "deep" feed-forward networks (networks with many layers) than back-propagation because errors are never back-propagated more than one layer, and therefore do not become diminished by repeated multiplications by derivatives of the logistic. Incompatibilities between distant layers are expressed by terms in the error function on layers between them.

9 Problems and Patches

Two major problems were discovered during the course of the simulations described below. Both were mentioned in section 3. One pertains to the discrepancy between the error measures (5) and (4), and the other is convergence to chaotic attractors.

The first problem is that the activation deficit error measure punishes activations which have the correct sign but greater magnitude than some arbitrary cutoff for the inverse logistic. Let us call this the "overshooting" problem. This problem does not occur on hidden events, because the moving targets are free to become as large as necessary to accommodate a computed activation. But the target events require constant target activations.

The constant targets need to be replaced by inequality constraints. One way of introducing inequality constraints, used by Jordan [6], is to turn the constant targets into variables while introducing a term in the error function which punishes them for violating the constraint. This type of meddling with the error function inevitably introduces tunable parameters which reflect the relative importance of constraint violation to the other sources of error. Instead, the constant targets can be replaced by a function of another variable, choosing a function with a range that satisfies the inequality constraint. Specifically, let the desired, somewhat arbitrary constant target values, formerly called X, now be called D, and for target events let X in all above formulas be redefined as a function g of D and a new minimization variable ξ:

$$X_{it} = g(D_{it}, \xi_{it}) = D_{it} + \frac{D_{it}}{|D_{it}|}\xi_{it}^2 \qquad (it) \in T. \tag{26}$$

This makes the X values free to wander anywhere they like, consistent with the constraint, without affecting the error at all. And the error function retains its all-important quadratic dependence on the weights.

The inelegant point is that more minimization variables have been introduced, one for every target event.

This technique was used successfully in the simulations. Without it, it was possible start with a hand-made solution to a dynamical training problem, initialize the moving targets to the activations produced on the hidden nodes by running the solution network, and nevertheless stray far away from the solution during training.

Next let us turn to the chaos problem. The moving targets error (7) is based on the differences between the computed activations and the moving targets at each time step. The moving targets can be varied within a small neighborhood of the computed activations without incurring a large error penalty. If the weight matrix is such that these small variations on one time step can produce large variations on the next step, then any of a large set of possible moving target values on the next time step can be accommodated cheaply. Consequently the algorithm has an incentive to find weight matrices which make the trajectories in activation-space highly sensitive to initial conditions; ie, chaotic. Such matrices are of no use when the network is asked to perform without the aid of training data with which to correct itself on every time step.

Weight matrices with large weight values give high sensitivity to initial conditions, so one way to combat the chaos problem is to insert a weight decay term

$$\tfrac{1}{2}\alpha \sum_{ij} w_{ij}^2 \tag{27}$$

into the error. This does not spoil the quadratic dependence on the weights, but it does introduce the tunable parameter α and addresses only one manner in which chaotic behavior can arise. But it is easy to do, and was used in the simulations as an interim measure.

If noise were added to the training data, then the network would have to be able to produce transitions similar to those in the original training data, starting from slightly perturbed states at each time step. This addresses the chaos problem, but also raises issues concerning how much noise to use, and introduces extra minimization variables.

I suggest (but have not tested) a method which is meant to concentrate on the fundamental motive for training with noise. There are two versions, one which requires no extra variables and one which requires as many as there are hidden events. Both methods involve tunable parameters, but their settings do not reflect difficult decisions about how much of one type of error is equivalent to how much of another. They are parameters analogous to the amplitude of the noise one might have trained with, and reflect easier decisions about the extent to which the final state of a transition should be immune to variations in the initial state.

Both methods use functions $\delta^{\pm}(Y)$ which produce small perturbations toward and away from $\frac{1}{2}$ with magnitude controlled by a small tunable parameter δ:

$$\delta^{\pm}(Y) = (1 \pm 2\delta)Y \mp \delta. \tag{28}$$

The first method is to redefine the error as:

$$E = \tfrac{1}{2}\sum_{it} \chi_{it} \left\{ \left[\sum_j w_{ij}\delta^{+}(f(X_{j,t-1})) - X_{it} \right]^2 + \left[\sum_j w_{ij}\delta^{-}(f(X_{j,t-1})) - X_{it} \right]^2 \right\}. \tag{29}$$

Another possible variation is to choose fixed random combinations of δ^+ and δ^- in each inner sum. This error function produces penalities if variations of the node output values on one time step do not get "back on track" for the next time step.

It is unreasonable to expect both sets of variations represented in (29) to produce identical activations, so it is manifestly impossible for the error to become 0. Therefore there will always be some uncertainty about whether the optimal solution will settle on some senseless error tradeoff. To fix this, inequality constraints can be placed on the moving target activations in (29). This would introduce two new variable parameters for each moving target. The δ^- variation is arguably more important than the δ^+ variation because the former variations are toward the linear region of the logistic, where they have a greater effect. Therefore let us cut the number of extra variables in half by considering only the δ^- variations. The extra parameters might as well be called ξ, just as they were for the target constraint parameters. Let

$$\Delta(\xi) = (1 - \Delta + 2\Delta f(\xi)), \tag{30}$$

where f is the logistic (2) and Δ (without an argument) is a small tunable parameter. Then the error function

$$E = \tfrac{1}{2}\sum_{it} \chi_{it} \left\{ \left[\sum_j w_{ij}(f(X_{j,t-1}) - X_{it} \right]^2 + \left[\sum_j w_{ij}\delta^{-}(f(X_{j,t-1})) - \Delta(\xi_{it})X_{it} \right]^2 \right\}. \tag{31}$$

can be minimized to 0 if small perturbations in the node outputs produce small enough variations in the activations on the following time steps.

10 Computational Results

The algorithm was tested on a few simple problems. Some succeeded and some failed. The failures appear to be due to the chaos problem.

Figures 1.1–1.8 concern a network trained to switch between two limit cycles under input control. Time proceeds from left to right. There is 1 input node, 1 target node and 1 hidden node as shown. The input node is an input node for all 8 time steps. The target and hidden nodes become input nodes where the vertical lines are drawn; the vertical lines separate different examples of sequences. The values shown at vertical lines also serve as target values for the proceeding transition. The input values, and fixed and moving target values are shown as heavy lines, while computed values are shown as light lines. Node output values between 0 and 1 are shown in all cases, rather than the activation values.

The heavy lines in the I and T rows of figure 1.1 specify the training problem. They can be seen with less clutter in figure 1.5. The first 4 time steps tell the target node to exhibit the sequence 010, when the input node is 0, and to return to the initial state so that the sequence should repeat itself in the freely running network. The last 5 time steps (step 4 does double duty.) tell the target node to repeat the sequence 0110 when the input node is 1.

The hidden node's moving target values are randomly initialized to the solid black line in row H of figure 1.1. The corresponding weight matrix produces the trajectory shown by the light lines in rows H and T. These light lines show the values computed during one time step, starting from the *correct* values (the solid black lines) on the previous time step.

Figures 1.1–1.5 show the same diagrams at various stages of training. The error has decreased by at least a factor of 5 between each figure. Note that the computed values gradually approach the moving target values. In row T the light lines approach the fixed solid lines, while in row H both the solid and light lines are allowed to vary.

Figure 1.5 shows the situation when training stopped. Figure 1.6 shows the results of running the network on the training data. It works. Figure 1.7 shows the results from running the network long enough to allow the cycles to repeat a few times. The vertical lines signify that the network is reset whenever the input is changed. That works too. Finally figure 1.8 shows the network running without reseting it when the input is changed. That works. This is a success story.

Figures 2.1–2.8 show a partial success story, or partial failure, depending on your attitude. Here the problem is to switch between 4 limit cycles under the control of 2 input nodes. The meaning of figures 2.1–2.8 is identical to that of the correspondingly numbered figures 1.1–1.8. Note that little or no perceptible error is evident in figure 2.5, at the end of training. (In fact, the error was small but significantly nonzero.) Running on the training data produced small but noticeable errors in figure 2.6. But when those errors are allowed to accumulate in figures 2.7 and 2.8, some of the limit cycles are severely degraded.

Figures 3.1–3.5 show a dramatic failure. In figure 3.1 the problem is solved correctly by a simple handmade network. The problem is to notice when the second of two impulses occur on the second input node. The first input node is intended as a reset signal and is superfluous to the problem when the network is reset anyway as the vertical lines signify. The target node turns on immediately after the second of two pulses occur on the second input node, regardless of the temporal separation of the pulses. If only one pulse occurs, the target remains off.

A single node with a suitable positive feedback feedback and negative bias can act as a flip-flop. When an input large enough to outweigh the bias comes along, the node turns on and remains on due to its positive feedback. The handmade network works by using both the

hidden and target nodes as flip-flops. The hidden node is biased to turn on when any input comes by. The target takes positive input from the hidden node, and is biased to turn on when both the hidden node and input node are on.

The training algorithm, alas, does not find this simple solution. At the end of training it performs as shown in figure 3.4. There are only small errors, but they occur at the most important time steps. Figure, 3.5 shows the network running on the training data, and failing to make any sense. The small displacements between the computed and moving target values shown in figure 3.4 are large enough to completely change the future behavior. Hopefully this problem will be solved by one of the methods designed to prevent chaotic solutions from forming.

11 Acknowledgements

This work was supported by the UK Information Engineering Directorate/ Science and Engineering Research Council as part of the IED/SERC Large Scale Integrated Speech Technology Demonstrator Project (SERC grants D/29604, D/29611, D/29628, F/10309, F/10316), in which Marconi Speech and Information Systems are the industrial partner.

References

[1] L. Almeida, "Backpropagation in Non-Feedforward Networks", in *Neural Computing Architectures*, I. Aleksander, ed., North Oxford Academic, (1989).

[2] K. Birmiwal, P. Sarwal, and S. Sinha, "A new Gradient-Free Learning Algorithm", Tech. report, Dept. of EE, Southern Illinois U., Carbondale, (1989).

[3] D. S. Broomhead and D. Lowe, "Radial Basis Functions, Multi-Variable Functional Interpolation and Adaptive Networks", Memorandum 4148, Royal Signals and Radar Establishment, St. Andrews Rd., Great Malvern, WORCS. UK. (1988)

[4] T. Grossman, R. Meir, and E. Domany, "Learning by Choice of Internal Representations", Dept. of Electronics, Weizmann Institute of Science, Rehovot, Israel, (1988).

[5] G. L. Heileman, M. Georgiopoulos, and A. K. Brown, "The Minimal Disturbance Back Propagation Algorithm", Tech. report, Dept. of EE, U. of Central Florida, Orlando, (1989).

[6] M. Jordan, "Supervised learning and systems with excess degrees of freedom", Computer and Info. Sci. Tech. Report 88-27, U. Mass. at Amherst, (1988).

[7] G. Kuhn, "Connected Recognition with a Recurrent Network", to appear in proc. NEUROSPEECH, 18 May 1989, as special issue of Speech Communication, v. 9, no. 2, (1990).

[8] Perlmutter, B., "Learning State Space Trajectories in Recurrent Neural Networks", Proc. IEEE IJCNN 89, Washington D. C., Publ. IEEE TAB Neural Network Committee., p. II-365, (1989).

[9] F. Pineda, "Dynamics and Architecture for Neural Computation", J. Complexity 4, p. 216, (1988).

[10] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipes, The Art of Scientific Computing*, Cambridge, (1986).

[11] S. Renals and R. Rohwer, "A Study of Network Dynamics", Edinburgh University Centre for Speech Technology Research preprint, submitted to J. Stat. Phys., (1989).

[12] S. Renals and R. Rohwer, "Phoneme Classification using Radial Basis Funcrtions", Proc. IEEE IJCNN 89, Washington, D. C., Publ. IEEE TAB Neural Network Committee., p. I-461, (1989).

[13] R. A. Roberts and C. T. Mullis, *Digital Signal Processing*, Addison Wesley, p.262 (1987).

[14] R. Rohwer, "Instant Solutions for Perceptron-Like Nets", Edinburgh University Centre for Speech Technology Research preprint, (1988).

[15] R. Rohwer and B. Forrest, "Training Time Dependence in Neural Networks" Proc. IEEE ICNN, San Diego, p. II-701, (1987).

[16] R. Rohwer and S. Renals, "Training Recurrent Networks", in *Neural Networks from Models to Applications*, L. Personnaz and G. Dreyfus, eds., I.D.S.E.T., Paris, p. 207, (1989).

[17] Rumelhart, D., Hinton, G. and Williams, R., "Learning Internal Representations by Error Propagation" in *Parallel Distributed Processing*, v. 1, MIT, (1986).

[18] Sanger, T., "Optimal Unsupervised Learning in a Single-Layer Linear Feedforward Neural Network", Tech. Report NE43-743, MIT AI Lab, (1988).

[19] R. L. Watrous, "Phoneme Discrimination using Connectionist Networks", Tech. Report, Dept. of Computer Science, U. Penn., submitted to J. Acoustical Soc. of America, (1989).

[20] Webb, A. and Lowe, D., "A Theorem Connecting Adaptive Feed-forward Layered Networks and Nonlinear Discriminant Analysis" Memorandum 4209, Royal Signals and Radar Establishment, St. Andrews Rd., Great Malvern, WORCS. UK. (1988)

[21] P. Werbos, *Energy Models and Studies*, B. Lev, Ed., North Holland, (1983).

[22] R. Williams and D. Zipser, "A Learning Algorithm for Continually Running Fully Recurrent Neural Networks", ICS Report 8805, UC San Diego, (1988).