REDUCING THE RATIO BETWEEN LEARNING COMPLEXITY AND NUMBER OF TIME VARYING VARIABLES IN FULLY RECURRENT NETS (Proc. ICANN'93, p. 460-463. Springer, 1993)

J. Schmidhuber
Institut für Informatik
Technische Universität München
Arcisstr. 21, 8000 München 40, Germany

ABSTRACT. *Let m be the number of time-varying variables for storing temporal events in a fully recurrent sequence processing network. Let R_{time} be the ratio between the number of operations per time step (for an* exact *gradient based supervised sequence learning algorithm), and m. Let R_{space} be the ratio between the maximum number of storage cells necessary for learning arbitrary sequences, and m. With conventional recurrent nets, m equals the number of units. With the popular 'real time recurrent learning algorithm' (RTRL), $R_{time} = O(m^3)$ and $R_{space} = O(m^2)$. With 'back-propagation through time' (BPTT), $R_{time} = O(m)$ (much better than with RTRL) and R_{space} is infinite (much worse than with RTRL). The contribution of this paper is a novel fully recurrent network and a corresponding* exact *gradient based learning algorithm with $R_{time} = O(m)$ (as good as with BPTT) and $R_{space} = O(m^2)$ (as good as with RTRL).*

1 INTRODUCTION

Architecture. The basic architecture considered in this paper is the one of a traditional fully recurrent sequence processing network. The network has n non-input units and $n_x = O(n)$ input units. Each input unit has a directed connection to each non-input unit. Each non-input unit has a directed connection to each non-input unit. Obviously there are $(n_x + n)n = n_{conn} = O(n^2)$ connections in the network. The k-th input unit is denoted by x_k. The k-th non-input unit is denoted by y_k. The k-th output unit is denoted by o_k (the n_o output units are a subset of the non-input units). The connection from unit j to unit i is denoted by w_{ij}. For instance, one of the names of the the connection from the j-th input unit to the the k-th output unit is $w_{o_k x_j}$.

Dynamics of conventional recurrent nets. To save indices, I consider a *single* discrete sequence of real-valued input vectors $x(t)$, $t = 1, \ldots, n_{time}$, each with dimension n_x. In what follows, if $v(t)$ denotes a vector then $v_k(t)$ denotes the k-th component of $v(t)$. If u denotes a unit, then $u(t)$ denotes the activation of the unit at time t. w_{ij}'s real-valued weight at time t is denoted by $w_{ij}(t)$. *Throughout the remainder of this paper, unquantized variables are assumed to take on their maximal range.* The dynamic evolution of a traditional discrete time recurrent net (e.g. [2], [7]) is given by

$$x_k(t) \leftarrow environment, \quad net_{y_k}(1) = 0, \quad y_k(t) = f_{y_k}(net_{y_k}(t)), \quad net_{y_k}(t+1) = \sum_{units\ l} w_{y_k l}(t) l(t), \quad (1)$$

where f_i denotes the semi-linear activation function of unit i, and where $w_{y_k l}(t)$ is *constant* for all t.

Typical objective function. There may exist specified target values $d_k(t)$ for certain outputs $o_k(t)$. We define

$$e_k(t) = d_k(t) - o_k(t) \quad if\ d_k(t)\ exists,\ and\ 0\ otherwise.$$

The supervised sequence learning task is to minimize (via gradient descent)

$$E^{total}(n_{time}), \quad where\ E^{total}(t) = \sum_{\tau=1}^{t} E(\tau), \quad where\ E(t) = \frac{1}{2} \sum_k (e_k(t))^2. \quad (2)$$

Complexity of traditional recurrent net algorithms. Let m be the number of time-varying variables for storing temporal events. Let R_{time} be the ratio between the number of operations per time step (for an *exact* gradient based supervised sequence learning algorithm), and m. Let R_{space} be the ratio between the maximum number of storage cells necessary for learning arbitrary sequences, and m.

The fastest known exact gradient based learning algorithm for minimizing (2) with fully recurrent networks is 'back-propagation through time' (BPTT, e.g. [3]). With BPTT, $m = n$, and $R_{time} =$

$O(m)$. BPTT's disadvantage is that it requires $O(n_{time})$ storage – which means that R_{space} is infinite: BPTT is not a *fixed-size storage* algorithm. The most well-known *fixed-size storage* learning algorithm for minimizing $E^{total}(n_{time})$ with fully recurrent nets is RTRL [2][8]. With RTRL, $m = n$, $R_{time} = O(m^3)$ (much worse than with BPTT) and $R_{space} = O(m^2)$ (much better than with BPTT). [1]

Contribution of this paper. The contribution of this paper is an extension of the conventional dynamics (as in equation (1)) plus a corresponding exact gradient based learning algorithm with $R_{time} = O(m)$ (as good as with BPTT) and $R_{space} = O(m^2)$ (as good as with RTRL). The basic idea is: The $O(n^2)$ weights themselves are included in the set of time-varying variables that can store temporal information ($m = n+O(n^2) = O(n^2)$). Section 2 describes the novel network dynamics that allow the network to actively and quickly manipulate its own weights (via so-called *intra-sequence* weight changes) by creating certain appropriate internal activation patterns [2] during observation of an input sequence[3]. Section 3 derives an *exact* supervised sequence learning algorithm (for creating *inter-sequence* weight changes which affect only the *initial* weights at the beginning of each training sequence) forcing the network to use its association building capabilities to minimize $E^{total}(n_{time})$. The gradient-based learning algorithm for inter-sequence weight changes takes into account the fact that intra-sequence weight changes at some point in time may influence both activations and intra-sequence weight changes at later points in time.

2 NOVEL DYNAMICS

I will keep the architecture and the objective function from section 1 but I will modify the system dynamics. Recall that unquantized variables are assumed to take on their maximal range. For our single training sequence with n_{time} discrete time steps, the system dynamics (explanation follows below) are defined by

$$x_k(t) \leftarrow environment, \quad net_{y_k}(1) = 0, \quad y_k(t) = f_{y_k}(net_{y_k}(t)), \quad net_{y_k}(t+1) = \sum_l w_{y_k l}(t) l(t), \tag{3}$$

$$w_{ij}(1) \leftarrow initialization, \quad w_{ij}(t+1) = \sigma_{ij}\left[w_{ij}(t) + g(j(t))h(i(t+1))\right], \tag{4}$$

where σ_{ij} is a differentiable function (e.g. for limiting the weight on w_{ij} to a given interval), and g and h are differentiable monotonic functions (the 'threshold approximators', to be explained below).

Equation (3) is just the conventional recurrent net update rule (1). Unlike with conventional recurrent nets, however, the weights do *not* remain constant during sequence processing : Equation (4) says that connections between units active at successive time steps are immediately strengthened or weakened essentially in proportion to pre-synaptic and post-synaptic activity. These *intra-sequence* weight changes are modulated by the non-linear functions g and h and may be negative (anti-Hebb-like) or zero as well as positive. Let us assume that all input vectors and all f_i are such that all units can take on only activations between 0 and 1. g and h are meant to specify the upper and lower thresholds that determine how strongly units have to be excited or inhibited to contribute to intra-sequence weight changes. A reasonable choice for g and h is one where g and h are strongly negative only if their argument is close to 0 and are strongly positive only if their argument is close to 1. Both g and h should return values close to 0 for arguments from the largest part of the interval between 0 and 1. This implies hardly any intra-sequence weight changes for connections between units that have non-extreme activations during successive time steps.

[1] It should be noted that there is a *fixed-size storage* algorithm (a hybrid between BPTT and RTRL) with $R_{time} = O(m^2)$ (better than with RTRL, worse than with BPTT) and $R_{space} = O(m^2)$ (much better than with BPTT, same as with RTRL) ([7][4]).

[2] Certain biological evidence is consistent with the idea of fast weight changes ('dynamic links', see [6]). In [1], for instance, it is shown that the effective connectivity between certain neurons may change drastically within a few 10 msec.

[3] The active weight changing capabilities represent a similarity to the system described in [5], which is based on two separate modules – one for learning to control fast weight changes of the other one. Unlike with this previous approach, however, the system described herein does not require two separate modules. It can learn to manipulate *its own* weights.

The overall effect is that only connections between units that are exceptionally active or exceptionally inactive during successive time steps can be significantly modified. *Intra-sequence* weight changes essentially occur only if the network 'pays a lot of attention' to certain units by strongly exciting them or strongly inhibiting them. Weights to units that are not 'illuminated by adaptive internal spotlights of attention' essentially remain invariant and participate only in 'automatic processing' as opposed to 'active intra-sequence learning'. The remainder of this paper derives an exact gradient-based algorithm designed to adjust the system (via *inter-sequence* weight changes) such that it creates appropriate *intra-sequence* weight changes at appropriate time steps.

3 SUPERVISED LEARNING ALGORITHM

The following algorithm for minimizing E^{total} is partly inspired by conventional recurrent network algorithms (e.g. [2], [7]). The notation is partly inspired by [8].

Derivation. Before training, all *initial* weights $w_{ab}(1)$ are randomly initialized. The chain rule serves to compute weight increments (to be performed *after* each training sequence) for all initial weights according to

$$w_{ab}(1) \leftarrow w_{ab}(1) - \eta \frac{\partial E^{total}(n_{time})}{\partial w_{ab}(1)}, \tag{5}$$

where η is a constant positive 'learning rate'. Thus we obtain an *exact* gradient-based algorithm for minimizing E^{total} under the dynamics given by (3) and (4).

We write

$$q^{ij}_{ab}(t) = \frac{\partial w_{ij}(t)}{\partial w_{ab}(1)}, \quad \forall\ units\ u : p^{u}_{ab}(t) = \frac{\partial u(t)}{\partial w_{ab}(1)}. \tag{6}$$

(Recall that unquantized variables are assumed to take on their maximal range.)

First note that

$$\frac{\partial E^{total}(1)}{\partial w_{ab}(1)} = 0, \quad \forall t > 1 : \ \frac{\partial E^{total}(t)}{\partial w_{ab}(1)} = \frac{\partial E^{total}(t-1)}{\partial w_{ab}(1)} - \sum_k e_k(t) p^{o_k}_{ab}(t). \tag{7}$$

Therefore, the remaining problem is to compute the $p^{o_k}_{ab}(t)$, which can be done by incrementally computing all $p^{z_k}_{ab}(t)$ and $q^{ij}_{ab}(t)$:

$$p^{z_k}_{ab}(1) = 0, \quad p^{x_k}_{ab}(t+1) = 0, \tag{8}$$

$$p^{y_k}_{ab}(t+1) = f'_{y_k}(net_{y_k}(t+1)) \sum_l \frac{\partial}{\partial w_{ab}(1)} [l(t) w_{y_k l}(t)] =$$

$$f'_{y_k}(net_{y_k}(t+1)) \sum_l \left[w_{y_k l}(t) p^{l}_{ab}(t) + l(t) q^{y_k l}_{ab}(t) \right], \tag{9}$$

where

$$q^{ij}_{ab}(1) = 1\ if\ w_{ab} = w_{ij},\ and\ 0\ otherwise, \tag{10}$$

$$q^{ij}_{ab}(t+1) = \sigma'_{ij}(w_{ij}(t+1)) \left[q^{ij}_{ab}(t) + h'(i(t+1)) p^{i}_{ab}(t+1) g(j(t)) + h(i(t+1)) p^{j}_{ab}(t) g'(j(t)) \right]. \tag{11}$$

According to equations (8)-(11), variables holding the $p^{j}_{ab}(t)$ and $q^{ij}_{ab}(t)$ values can be updated incrementally at each time step. This implies that (5) can be updated incrementally, too. With non-degenerate networks, the algorithm's storage complexity is dominated by the number of variables for storing the $q^{ij}_{ab}(t)$ values. This number is independent of the sequence length and equals $O(n^2_{conn})$. Since $m = O(n_{conn})$, $R_{space} = O(m^2)$ (like with RTRL). The computational complexity per time step also is $O(n^2_{conn})$ – essentially the same as the one of RTRL. Since $m = O(n_{conn})$, however, $R_{time} = O(m)$ (like with time-efficient BPTT and unlike with RTRL's much worse $R_{time} = O(m^3)$).

4 CONCLUDING REMARKS

I have described a novel fully recurrent network that may choose to behave like a conventional fully recurrent net. In addition, however, the novel net may choose to use its own weights for storing temporal events. The network can do so by creating and directing 'internal spotlights of attention' to cause intra-sequence weight changes that may help the system to achieve its goal (defined by a conventional objective function for supervised sequence learning). The corresponding exact gradient based learning algorithm turns out to have the same ratio between number of learning operations per time step and number of time-varying variables as the time-efficient BPTT algorithm. In addition, it turns out to have the same ratio between maximum storage and number of time-varying variables as the space-efficient RTRL algorithm.

Of course, since fast weights and unit activations are different kinds of variables, I am subsuming different things under the expression 'time-varying variable'. I expect that some problems may be more naturally solved using information processing based on time-varying unit activations, other problems may be more naturally solved using information processing based on fast weights (e.g. certain kinds of temporal variable binding problems, see [5]). Careful experimental investigations of the mutual advantages and disadvantages of both kinds of time-varying variables are needed (experiments are also needed for analyzing different reasonable choices for functions like g, h, σ) but are beyond the scope of this short paper and will be left for the future.

5 ACKNOWLEDGEMENTS

This work was supported in part by a DFG fellowship to the author, as well as by NSF award IRI–9058450, grant 90–21 from the James S. McDonnell Foundation, and DEC external research grant 1250.

References

[1] A.M.H.J. Aertsen, G.L. Gerstein, M.K. Habib, and G. Palm. Dynamics of neuronal firing correlation: Modulation of "effective connectivity". *Journal of Neurophysiology*, 61:900–917, 1989.

[2] A. J. Robinson and F. Fallside. The utility driven dynamic error propagation network. Technical Report CUED/F-INFENG/TR.1, Cambridge University Engineering Department, 1987.

[3] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In *Parallel Distributed Processing*, volume 1, pages 318–362. MIT Press, 1986.

[4] J. H. Schmidhuber. A fixed size storage $O(n^3)$ time complexity learning algorithm for fully recurrent continually running networks. *Neural Computation*, 4(2):243–248, 1992.

[5] J. H. Schmidhuber. Learning to control fast-weight memories: An alternative to recurrent nets. *Neural Computation*, 4(1):131–139, 1992.

[6] C. v.d. Malsburg. Technical Report 81-2, Abteilung für Neurobiologie, Max-Planck Institut für Biophysik und Chemie, Göttingen, 1981.

[7] R. J. Williams. Complexity of exact gradient computation algorithms for recurrent neural networks. Technical Report Technical Report NU-CCS-89-27, Boston: Northeastern University, College of Computer Science, 1989.

[8] R. J. Williams and D. Zipser. A learning algorithm for continually running fully recurrent networks. *Neural Computation*, 1(2):270–280, 1989.