AN ‘INTROSPECTIVE’ NETWORK THAT CAN LEARN TO RUN ITS OWN WEIGHT CHANGE ALGORITHM

In Proc. of the Intl. Conf. on Artificial Neural Networks, Brighton, pages 191-195. IEE, 1993.

J. Schmidhuber

Technische Universität München
Germany

Abstract. *Usually weight changes in neural networks are exclusively caused by some hard-wired learning algorithm with many specific limitations. I show that it is in principle possible to let the network run and improve its own weight change algorithm (without significant theoretical limits). I derive an* initial *gradient-based supervised sequence learning algorithm for an* ‘introspective’ *recurrent network that can ‘speak’ about its own weight matrix in terms of activations. It uses special subsets of its input and output units for observing its own errors and for explicitly analyzing and manipulating all of its own weights, including those weights responsible for analyzing and manipulating weights. The result is the first ‘self-referential’ neural network with explicit potential control over all adaptive parameters governing its behavior.*

INTRODUCTION

Usually weight changes in artificial neural networks are exclusively caused by some *fixed hard-wired* learning algorithm with many specific limitations. In contrast, humans can reflect about their own learning behavior and *modify* it and tailor it to the needs of various types of learning problems.

The thought experiment in this paper is intended to show the theoretical possibility of ‘self-referential’ neural networks that can learn to run and improve their own weight change algorithm. The first step is the design of a general finite-size hard-wired ‘introspective’ architecture with access to performance evaluations and with the potential to analyze and modify its own weight matrix. The second step is the design of an *initial* learning algorithm that finds useful self-manipulating algorithms (weight matrices) for the architecture, where ‘usefulness’ is strictly defined by performance evaluations provided by the environment.

The paper is structured as follows: Section 2 starts with a general finite, ‘self-referential’ architecture involving a sequence-processing recurrent neural-net (see e.g. Robinson and Fallside [2], Williams and Zipser [8], and Schmidhuber [3]) that can potentially implement any computable function that maps input sequences to output sequences — the only limitations being unavoidable time and storage constraints imposed by the architecture’s finiteness. These constraints can be extended by simply adding storage and/or allowing for more processing time. The major novel aspect of the system is its ‘self-referential’ capability. The network is provided with special input units for explicitly observing performance evaluations (external error signals are visible through these special input units). In addition, it is provided with the basic tools for explicitly reading and quickly changing *all* of its own adaptive components (weights). This is achieved by (1) introducing an address for each connection of the network, (2) providing the network with output units for (sequentially) addressing *all* of its own connections (including those connections responsible for addressing connections) by means of time-varying activation patterns, (3) providing special input units whose activations become the weights of connections currently addressed by the network, and (4) providing special output units whose time-varying activations serve to quickly change the weights of connections addressed by the network. It is possible to show that these unconventional features allow the network (in principle) to compute any computable function mapping *algorithm components* (weights) and *performance evaluations* (e.g., error signals) to *algorithm modifications* (weight changes) – the only limitations again being unavoidable time and storage constraints. This implies that algorithms running on that architecture (in principle) *can change not only themselves but also the way they change themselves, and the way they change the way they change themselves, etc.*, essentially without theoretical limits to the sophistication (computational power) of the self-modifying algorithms.

Connections are addressed, analyzed, and manipulated with the help of differentiable functions of activation patterns across special output units. This allows the derivation of an *exact* gradient-based *initial* weight change algorithm for ‘introspective’ supervised sequence learning. The system starts out as *tabula rasa.* The initial weight change procedure serves to find improved weight change procedures – it favors algorithms (weight matrices) that make sensible use of the ‘introspective’ potential of the hard-wired architecture, where ‘usefulness’ is solely defined by conventional performance evaluations (the performance measure we use is the sum of all error signals over all time steps of all training sequences).

A disadvantage of the algorithm is its high computational complexity per time step which is independent of the sequence length and equals $O(n_{conn} log n_{conn})$, where n_{conn} is the number of connections. Another disadvantage is the high number of local minima of the unusually complex error surface. The purpose of this paper, however, is not to come up with the most efficient 'introspective' or 'self-referential' weight change algorithm, but to show that such algorithms are possible at all.

THE 'INTROSPECTIVE' NETWORK

Throughout the remainder of this paper, to save indices, I consider a single limited pre-specified time-interval of discrete time-steps during which our network interacts with its environment. An interaction sequence actually may be the concatenation of many 'conventional' training sequences for conventional recurrent networks. This will (in theory) help our 'self-referential' net to find regularities among solutions for *different* tasks.

The network's output vector at time t, $o(t)$, is computed from previous input vectors $x(\tau), \tau < t$, by a discrete time recurrent network with n_I input units and n_y non-input units. A subset of the non-input units, the 'normal' output units, has a cardinality of $n_o < n_y$.

z_k is the k-th unit in the network. y_k is the k-th non-input unit in the network. x_k is the k-th 'normal' input unit in the network. o_k is the k-th 'normal' output unit. If u stands for a unit, then f_u is its differentiable activation function and u's activation at time t is denoted by $u(t)$. If $v(t)$ stands for a vector, then $v_k(t)$ is the k-th component of $v(t)$.

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. There are $(n_I + n_y)n_y = n_{conn}$ connections in the network. The connection from unit j to unit i is denoted by w_{ij}. For instance, one of the names of the connection from the j-th 'normal' input unit to the the k-th 'normal' output unit is $w_{o_k x_j}$. w_{ij}'s real-valued weight at time t is denoted by $w_{ij}(t)$. Before training, all weights $w_{ij}(1)$ are randomly initialized.

The following features are needed to obtain 'self-reference'. *Details of the network dynamics follow in the next section.*

1. The network receives performance information through the *eval units*. The eval units are special input units which are not 'normal' input units. $eval_k$ is the k-th eval unit (of n_{eval} such units) in the network.

2. Each connection of the net gets an address. One way of doing this is to introduce a *binary* address, $adr(w_{ij})$, for each connection w_{ij}. This will help the network to do computations concerning its own *weights* in terms of *activations*, as will be seen later.

3. ana_k is the k-th *analyzing unit* (of $n_{ana} = ceil(log_2 n_{conn})$ such units, where $ceil(x)$ returns the first integer $\geq x$). The analyzing units are special non-input units which are not 'normal' output units. They serve to indicate which connections the current algorithm of the network (defined by the current weight matrix plus the current activations) will access next (see next section). A special input unit for reading current weight values that is used in conjunction with the analyzing units is called *val*.

The network may modify any of its weights. Some non-input units that are not 'normal' output units or analyzing units are called the *modifying units*. mod_k is the k-th modifying unit (of $n_{mod} = ceil(log_2 n_{conn})$ such units). The modifying units serve to address connections to be modified. A special output unit for modifying weights (used in conjunction with the modifying units, see next section) is called $\triangle$. $f_\triangle$ should allow both positive and negative activations of $\triangle(t)$.

'SELF-REFERENTIAL' DYNAMICS AND OBJECTIVE FUNCTION

I assume that the input sequence observed by the network has length $n_{time} = n_s n_r$ (where $n_s, n_r \in \mathbf{N}$) and can be divided into n_s equal-sized blocks of length n_r during which the input pattern $x(t)$ does not change. This does not imply a loss of generality — it just means speeding up the network's hardware such that each input pattern is presented for n_r time-steps before the next pattern can be observed. This gives the architecture n_r time-steps to do some sequential processing (including immediate weight changes) before seeing a new pattern of the input sequence.

In what follows, *unquantized variables are assumed to take on their maximal range.* The network dynamics are specified as follows:

$$net_{y_k}(1) = 0,$$

$$\forall t \geq 1: \quad x_k(t) \leftarrow environment,$$

$$y_k(t) = f_{y_k}(net_{y_k}(t)),$$

$$\forall t > 1: \quad net_{y_k}(t) = \sum_l w_{y_k l}(t-1) l(t-1), \qquad (1)$$

The network can quickly read information about its current weights into the special *val* input unit according to

$$val(1) = 0, \quad \forall t \geq 1:$$

$$val(t+1) = \sum_{i,j} g[\|ana(t) - adr(w_{ij})\|^2] w_{ij}(t), \quad (2)$$

where $\|\ldots\|$ denotes Euclidean length, and g is a differentiable function emitting values between 0 and 1 that determines how close a connection address has to be to the activations of the analyzing units in order for its weight to contribute to *val* at that time. Such a function g might have a narrow peak at 1 around the origin and be zero (or nearly zero) everywhere else. This essentially allows the network to pick out a single connection at a time and obtain its current weight value without receiving 'cross-talk' from other weights.

The network can quickly modify its current weights using $mod(t)$ and $\triangle(t)$ according to

$$\forall t \geq 1: \quad w_{ij}(t+1) =$$

$$= w_{ij}(t) + \triangle(t)\ g[\ \|adr(w_{ij}) - mod(t)\|^2\ ]. \qquad (3)$$

Again, if g has a narrow peak at 1 around the origin and is zero (or nearly zero) everywhere else, the network will be able to pick out a single connection at a time and change its weight without affecting other weights.

Objective function and dynamics of the eval *units.* As with typical supervised sequence-learning tasks, we want to minimize

$$E^{total}(n_r n_s),$$

where

$$E^{total}(t) = \sum_{\tau=1}^{t} E(\tau),$$

where

$$E(t) = \frac{1}{2}\sum_k (eval_k(t+1))^2,$$

where

$$eval_k(1) = 0, \quad \forall t \geq 1: eval_k(t+1) =$$

$$= d_k(t) - o_k(t)\ \ if\ d_k(t)\ exists,\ and\ 0\ else. \qquad (4)$$

Here $d_k(t)$ may be a desired target value for the k-th output unit at time step t.

INITIAL LEARNING ALGORITHM

The following algorithm[1] for minimizing E^{total} is partly inspired by (but more complex than) conventional recurrent network algorithms (e.g. Robinson and Fallside [2]).

Derivation of the algorithm. We use the chain rule to compute weight increments (to be performed *after* each training sequence) for all *initial* weights $w_{ab}(1)$ according to

$$w_{ab}(1) \leftarrow w_{ab}(1) - \eta \frac{\partial E^{total}(n_r n_s)}{\partial w_{ab}(1)}, \qquad (5)$$

where η is a constant positive 'learning rate'. Thus we obtain an *exact* gradient-based algorithm for minimizing E^{total} under the 'self-referential' dynamics given by (1)-(4). To reduce writing effort, I introduce some short-hand notation partly inspired by Williams [7]. For all units u and all weights w_{ab}, w_{ij} we write

$$p_{ab}^{u}(t) = \frac{\partial u(t)}{\partial w_{ab}(1)}, q_{ab}^{ij}(t) = \frac{\partial w_{ij}(t)}{\partial w_{ab}(1)}. \qquad (6)$$

[1]It should be noted that in quite different contexts, previous papers have shown how 'controller nets' may learn to perform appropriate lasting weight changes for a second net (see Schmidhuber [4] and Möller and Thrun [1]). However, these previous approaches could not be called *'self-referential'* — they all involve at least some weights that can *not* be manipulated other than by conventional gradient descent.

To begin with, note that

$$\frac{\partial E^{total}(1)}{\partial w_{ab}(1)} = 0,$$

$$\forall t > 1: \ \frac{\partial E^{total}(t)}{\partial w_{ab}(1)} =$$

$$= \frac{\partial E^{total}(t-1)}{\partial w_{ab}(1)} - \sum_k eval_k(t+1) p_{ab}^{o_k}(t). \qquad (7)$$

Therefore, the remaining problem is to compute the $p_{ab}^{o_k}(t)$, which can be done by incrementally computing all $p_{ab}^{z_k}(t)$ and $q_{ab}^{ij}(t)$, as we will see. At time step 1 we have

$$p_{ab}^{z_k}(1) = 0. \qquad (8)$$

For $t \geq 1$ we obtain the recursion

$$p_{ab}^{x_k}(t+1) = 0,$$

$$p_{ab}^{eval_k}(t+1) =$$

$$= -p_{ab}^{o_k}(t),\ \ if\ d_k(t)\ exists,\ and\ 0\ otherwise, \qquad (9)$$

$$p_{ab}^{val}(t+1) =$$

$$\sum_{i,j}\{\ \ q_{ab}^{ij}(t) g[\|ana(t) - adr(w_{ij})\|^2)] +$$

$$+ w_{ij}(t)\ [\ \ g'(\|ana(t) - adr(w_{ij})\|^2) \times$$

$$\times 2 \sum_m (ana_m(t) - adr_m(w_{ij})) p_{ab}^{ana_m}(t)\ \ ]\ \ \ \} \qquad (10)$$

(where $adr_m(w_{ij})$ is the m-th bit of w_{ij}'s address),

$$p_{ab}^{y_k}(t+1) =$$

$$f'_{y_k}(net_{y_k}(t+1)) \sum_l w_{y_k l}(t) p_{ab}^{l}(t) + l(t) q_{ab}^{y_k l}(t), \qquad (11)$$

where

$$q_{ab}^{ij}(1) = 1\ if\ w_{ab} = w_{ij},\ and\ 0\ otherwise, \qquad (12)$$

$$\forall t > 1: \quad q_{ab}^{ij}(t) = q_{ab}^{ij}(t-1) +$$

$$+ p_{ab}^{\triangle}(t-1) g(\|mod(t-1) - adr(w_{ij})\|^2) +$$

$$+ 2 \triangle (t-1)\ g'(\|mod(t-1) - adr(w_{ij})\|^2) \times$$

$$\times \sum_m [mod_m(t-1) - adr_m(w_{ij})] p_{ab}^{mod_m}(t-1). \qquad (13)$$

According to equations (8)-(13), the $p_{ab}^{j}(t)$ and $q_{ab}^{ij}(t)$ can be updated incrementally at each time step. This implies that (5) can be updated incrementally at each time step, too. The storage complexity is independent of the sequence length and equals $O(n_{conn}^2)$. The computational complexity per time step (of sequences with arbitrary length) is $O(n_{conn}^2 log n_{conn})$.

CONCLUDING REMARKS

The network I have described can, besides learning to solve problems posed by the environment, also use its own weights as input data and can learn new algorithms for modifying its weights in response to the environmental input and evaluations. This effectively embeds a chain of *'meta-networks'* and *'meta-meta-...-networks'* into the network itself.

Due to the complexity of the activation dynamics of the 'self-referential' network, one would expect the above error function to have many local minima. Schmidhuber [6] describes a variant of the basic idea (involving a biologically more plausible weight manipulating strategy) which is less plagued by the problem of local minima (and whose initial learning algorithm has lower computational complexity than the one above). [5] describes a more general but less informed and less complex reinforcement learning algorithm.

This paper does not focus on experimental evaluations; the thought experiment presented in this paper is intended only to show the theoretical possibility of certain kinds of 'self-referential' weight change algorithms. Experimental evaluations of alternative 'self-referential' architectures will be left for the future.

ACKNOWLEDGEMENTS

Thanks to Mark Ring, Mike Mozer, Daniel Prelinger, Don Mathis, and Bruce Tesar, for helpful comments. This research 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] K. Möller and S. Thrun. Task modularization by network modulation. In J. Rault, editor, *Proceedings of Neuro-Nimes '90*, pages 419–432, November 1990.

[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] J. 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.

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

[5] J. Schmidhuber. Steps towards "self-referential" learning. Technical Report CU-CS-627-92, Dept. of Comp. Sci., University of Colorado at Boulder, November 1992.

[6] J. Schmidhuber. On decreasing the ratio between learning complexity and number of time varying variables in fully recurrent nets. Technical report, Institut für Informatik, Technische Universität München, 1993. In preparation.

[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.