A 'SELF-REFERENTIAL' WEIGHT MATRIX

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

ABSTRACT. *Weight modifications in traditional neural nets are computed by hard-wired algorithms. Without exception, all previous weight change algorithms have many specific limitations. Is it (in principle) possible to overcome limitations of hard-wired algorithms by allowing neural nets to run and improve their own weight change algorithms? This paper constructively demonstrates that the answer (in principle) is 'yes'. I derive an initial gradient-based sequence learning algorithm for a 'self-referential' recurrent network that can 'speak' about its own weight matrix in terms of activations. It uses some of its input and output units for observing its own errors and for explicitly analyzing and modifying its own weight matrix, including those parts of the weight matrix responsible for analyzing and modifying the weight matrix. The result is the first 'introspective' neural net with explicit potential control over all of its own adaptive parameters. A disadvantage of the algorithm is its high computational complexity per time step which is independent of the sequence length and equals* $O(n_{conn}logn_{conn})$, *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.*

1. INTRODUCTION

In contrast to traditional machine learning systems, humans do not appear to rely on hard-wired learning algorithms only. Instead, they tend to reflect about their own learning behavior and *modify* it and tailor it to the needs of various types of learning problems. To a degree, humans are able to learn how to learn. The thought experiment in this paper is intended to make a step towards 'self-referential' machine learning by showing the theoretical possibility of 'self-referential' neural networks whose weight matrices can learn to implement and improve their own weight change algorithm, without any significant theoretical limits.

Structure of the paper. 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.

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.

2. 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' weight matrix 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*, which are special 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. 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*.

4. 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 $\triangle(t)$.

2.1. '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:

$$net_{y_k}(1) = 0, \quad \forall t \geq 1: \quad x_k(t) \leftarrow environment, \quad y_k(t) = f_{y_k}(net_{y_k}(t)),$$
$$\forall t > 1: \quad net_{y_k}(t) = \sum_l w_{y_k l}(t-1) l(t-1), \tag{1}$$

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

$$val(1) = 0, \quad \forall t \geq 1: \; val(t+1) = \sum_{i,j} g[\|ana(t) - adr(w_{ij})\|^2] w_{ij}(t), \tag{2}$$

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

$$\forall t \geq 1: \quad w_{ij}(t+1) = w_{ij}(t) + \triangle(t)\; g[\; \|adr(w_{ij}) - mod(t)\|^2 \;]. \tag{3}$$

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

$$E^{total}(n_r n_s), \quad where \quad E^{total}(t) = \sum_{\tau=1}^{t} E(\tau), \quad where \quad E(t) = \frac{1}{2} \sum_k (eval_k(t+1))^2,$$

where

$$eval_k(1) = 0, \quad \forall t \geq 1 : eval_k(t+1) = d_k(t) - o_k(t) \quad if\ d_k(t)\ exists,\ and\ 0\ else. \tag{4}$$

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

3. 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

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

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

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

To begin with, note that

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

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 $q^{ij}_{ab}(t)$, as we will see. We have

$$p^{z_k}_{ab}(1) = 0; \quad p^{x_k}_{ab}(t+1) = 0; \quad p^{eval_k}_{ab}(t+1) = -p^{o_k}_{ab}(t), \quad if\ d_k(t)\ exists,\ and\ 0\ otherwise, \tag{8}$$

$$p^{val}_{ab}(t+1) = \sum_{i,j} \{ \ q^{ij}_{ab}(t) g[\|ana(t) - adr(w_{ij})\|^2)] \ + w_{ij}(t) \ [\ g'(\|ana(t) - adr(w_{ij})\|^2) \times$$

$$\times 2 \sum_m (ana_m(t) - adr_m(w_{ij})) p^{ana_m}_{ab}(t) \ ] \ \} \tag{9}$$

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

$$p^{y_k}_{ab}(t+1) = f'_{y_k}(net_{y_k}(t+1)) \sum_l w_{y_k l}(t) p^l_{ab}(t) + l(t) q^{y_k l}_{ab}(t), \tag{10}$$

where

$$q^{ij}_{ab}(1) = 1\ if\ w_{ab} = w_{ij},\ and\ 0\ otherwise, \tag{11}$$

$$\forall t > 1 : \quad q^{ij}_{ab}(t) = q^{ij}_{ab}(t-1) + p^{\triangle}_{ab}(t-1) g(\|mod(t-1) - adr(w_{ij})\|^2) +$$

$$+ 2 \triangle (t-1)\ g'(\|mod(t-1) - adr(w_{ij})\|^2) \times \sum_m [mod_m(t-1) - adr_m(w_{ij})] p^{mod_m}_{ab}(t-1). \tag{12}$$

According to (8)-(12), the $p^j_{ab}(t)$ and $q^{ij}_{ab}(t)$ 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 $O(n^2_{conn})$. The computational complexity per time step (of sequences with arbitrary length) is $O(n^2_{conn} log n_{conn})$.

[1] It should be noted that in quite different contexts, previous papers have shown how one net may learn to perform appropriate lasting weight changes for a second net [4] [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.

4. CONCLUSION

The thought experiment presented in this paper is intended to show the theoretical possibility of certain kinds of 'self-referential' weight matrices. 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 (in principle) 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.

5. ACKNOWLEDGEMENTS

Thanks to Mark Ring, Mike Mozer, Daniel Prelinger, Don Mathis, and Bruce Tesar, for helpful comments. Parts of this paper are based on previous publications [6][5]. 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. 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.

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

[5] J. H. Schmidhuber. An introspective network that can learn to run its own weight change algorithm. In *Proc. of the Third International Conference on Artificial Neural Networks, Brighton*. IEE, 1993. Accepted for publication.

[6] J. H. Schmidhuber. A neural network that embeds its own meta-levels. In *Proc. of the International Conference on Neural Networks '93, San Francisco*. IEEE, 1993. Accepted for publication.

[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.