A NEURAL NETWORK THAT EMBEDS ITS OWN META-LEVELS

In Proc. of the International Conference on Neural Networks '93, San Francisco. IEEE, 1993.

Jürgen Schmidhuber
Department of Computer Science
University of Colorado
Campus Box 430, Boulder, CO 80309, USA

Abstract

Traditional artificial neural networks cannot reflect about their own weight modification algorithm. Once the weight modification algorithm is programmed, it remains 'hard-wired' and cannot adaptively tailor itself to the needs of specific types of learning problems in a given environment. I present a novel recurrent neural network which (in principle) can, besides learning to solve problems posed by the environment, also use its own weights as input data and learn new (arbitrarily complex) algorithms for modifying its own weights in response to the environmental input and evaluations. The network uses 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. This effectively embeds a chain of 'meta-networks' and 'meta-meta-...-networks' into the network itself.

1 INTRODUCTION

Conventional artificial neural networks cannot reflect about their own weight change algorithm. Once the weight change algorithm is programmed, it cannot adapt itself to the needs of specific types of learning problems in specific environments.

This paper is intended to show the theoretical possibility of a certain kind of 'meta-learning' neural networks that can learn to run their own weight change algorithm and to improve it in an adaptive manner.

The paper is structured as follows: Section 2 introduces the basic finite architecture. This architecture involves a conventional sequence-processing recurrent neural-net (see e.g. [4], [12], [6]) 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 architecture is its 'self-referential' capability. Some of the network's input units serve to explicitly observe performance evaluations (e.g., external error signals are visible through these special input units). In addition, the net 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) using some of the network's output units for sequentially addressing *all* of its own connections (including those connections responsible for addressing connections), (3) using some of the input units for 'reading weights' – their activations become the weights of connections currently addressed by the network, and (4) interpreting the activations of some of the output units as immediate (possibly dramatic) changes of the weights of connections addressed by the network. 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.

In section 3, an *exact* gradient-based *initial* weight change algorithm for 'self-referential' supervised sequence learning is derived. 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 is the sum of all error signals over all time steps of all training sequences).

A disadvantage of the algorithm is its computational complexity per time step which is independent of the sequence length and equals $O(n_{conn}^2 log n_{conn})$, 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 or most practical 'introspective' or 'self-referential' weight change algorithm, but to show that such algorithms are possible at all.

2 THE 'SELF-REFERENTIAL' 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.

Conventional aspects of the net. 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 $n_I > n_x$ input units and n_y non-input units. A subset of the non-input units, the 'normal' output units, has a cardinality of $n_o < n_y$.

For notational convenience, I will sometimes give *different* names to the real-valued activation of a particular unit at a particular time step. 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 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 $v(t)$ (this is consistent with the last sentence).

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 $(n_I + n_y)n_y = n_{conn}$ 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 definitions will look familiar to the reader knowledgeable about conventional recurrent nets (e.g. [13]). The environment determines the activations of a 'normal' input unit x_k. For a non-input unit y_k we define

$$net_{y_k}(1) = 0, \;\; \forall t \geq 1 : y_k(t) = f_{y_k}(net_{y_k}(t)),$$

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

where f_i is the activation function of unit i.

The current algorithm of the network is given by its current weight matrix (and the current activations). Note, however, that I have not yet specified how the $w_{ij}(t)$ are computed.

'Self-referential' aspects of the net. The following is a list of four unconventional aspects of the system, which should be viewed as just one example of many similar systems.

1. The network sees performance evaluations. 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 which I employ in this paper (but certainly not the only way) 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 next.

3. The network may analyze any of its weights. ana_k is the k-th *analyzing unit* (of n_{ana} such units) in the network. 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. It is possible to endow the analyzing units with enough capacity to address *any* connection, *including connections leading to analyzing units*[1].

One way of doing this is to set

$$n_{ana} = ceil(log_2 n_{conn}) \qquad (2)$$

where $ceil(x)$ returns the first integer $\geq x$.

A special input unit that is used in conjunction with the analyzing units is called val. $val(t)$ is computed according to

$$val(1) = 0, \;\; \forall t \geq 1 : \; val(t+1) =$$
$$= \sum_{i,j} g[\|ana(t) - adr(w_{ij})\|^2] w_{ij}(t), \qquad (3)$$

where $\| \dots \|$ denotes Euclidean length, and g is a 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

[1]Note that we need to have a compact form of addressing connections: One might alternatively think of something like 'one analyzing unit for each connection' to address all weights in parallel, but obviously this would not work — we always would end up with more weights than units and could not obtain 'self-reference'. It should be noted, however, that the binary addressing scheme above is by far not the most compact scheme. This is because real-valued activations allow for representing theoretically unlimited amounts of information in a single unit. For instance, theoretically it is possible to represent *arbitrary simultaneous changes* of all weights within a single unit. In practical applications, however, there is nothing like unlimited precision real values. And the purpose of this paper is not to present the most compact 'self-referential' addressing scheme but to present at least one such scheme.

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 would essentially allow the network to pick out a single connection at a time and obtain its current weight value without receiving 'cross-talk' from other weights.

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} such units) in the network. The modifying units serve to address connections to be modified. Again, it is possible to endow the modifying units with enough capacity to sequentially address *any* connection, *including connections leading to modifying units.* One way of doing this is to set

$$n_{mod} = ceil(log_2 n_{conn}) \tag{4}$$

A special output unit used in conjunction with the modifying units is called $\triangle$. $f_\triangle$ should allow both positive and negative activations of $\triangle(t)$. Together, $mod(t)$ and $\triangle(t)$ serve to explicitly change weights according to

$$w_{ij}(t+1) = w_{ij}(t) + \triangle(t)\ g[\ \|adr(w_{ij}) - mod(t)\|^2\ ]. \tag{5}$$

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. It is straightforward, however, to devise schemes that allow the system to modify more than one weight in parallel.

Together, (1), (3), and (5) make up the hard-wired system dynamics.

2.1 COMPUTATIONAL POWER OF THE NET

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. Although the architecture may influence the state of the environment within such a block of n_r time steps, the changes will not affect its input until the beginning of the next block.

With appropriate constant (time-invariant) $w_{ij}(t)$, simple conventional (threshold or semi-linear) activation functions f_k, sufficient n_h 'hidden' units, and sufficient blocksize n_r, by repeated application of (1), the network can compute any function (or combination of functions)

$$f : \{0,1\}^{n_x+1+n_{eval}+n_o+n_{ana}+n_{mod}+1} \rightarrow$$

$$\rightarrow \{0,1\}^{n_o+n_{ana}+n_{mod}+1} \tag{6}$$

computable within a constant finite number n_{cyc} of machine cycles by a given algorithm running on a given conventional digital (sequential or parallel) computer with limited temporal and storage resources. This is because information processing in conventional computers can be described by the repeated application of boolean functions that can easily be emulated in recurrent nets as above.

With the particular set-up of section 2.2, at least the $\triangle$ output unit and the *val* input unit should take on not only binary values but real values. It is not difficult, however, to show that the range $\{0,1\}$ in (6) may be replaced by $\mathbf{R}$ for any unit (by introducing appropriate simple activation functions).

We now can clearly identify the storage constraint n_h and the time constraint n_r with two parameters, without having to take care of any additional hardware-specific limitations constraining the computational power of the net[2].

3 EXACT GRADIENT-BASED LEARNING ALGORITHM

Hardwired aspects of the learning algorithm. Why does the sub-title of this paragraph refer to *hardwired learning algorithms*, although this paper is inspired by the idea of *learning* learning algorithms? Because certain aspects of the initial learning algorithm may not be modified. There is no way of making *everything* adaptive — for instance, algorithms leading to desirable evaluations must always be favored over others. *We may not allow the system to change this basic rule of the game.* The *hardwired* aspects of the *initial* learning algorithm will favor algorithms that modify themselves in a useful manner (a manner that leads to 'more desirable' evaluations).

This section derives an exact gradient-based algorithm for supervised sequence learning tasks. For the purposes of this section, f_k and g must be differentiable. This will allow us to compute gradient-based *directions* for the search in algorithm space. For another variant of the architecture, [8] describes a more general but less informed and less complex reinforcement learning algorithm.

With supervised learning, the eval units provide information about the desired outputs at certain time steps. Arbitrary time lags may exist between inputs and later correlated outputs. For $o_k(t)$ there may be a target value, $d_k(t)$, specified at time t. We set $n_{eval} = n_o$, which means that there are as many eval units as there are 'normal' output units. The current activation of a particular eval unit provides information about the error of the corresponding

[2]It should be mentioned that since n_{conn} grows with n_h, n_{ana} and n_{mod} also grow with n_h (if they are not chosen large enough from the beginning). However, n_{ana} and n_{mod} grow much slower than n_h.

output unit at the previous time step (see equation (10)). We assume that inputs and target values do not depend on previous outputs (via feedback through the environment).

To obtain a better overview, let us summarize the system dynamics in compact form. In what follows, *unquantized variables are assumed to take on their maximal range:*

$$net_{y_k}(1) = 0, \quad \forall t \geq 1: \quad x_k(t) \leftarrow environment,$$

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

$$\forall t > 1: \quad net_{y_k}(t) = \sum_l w_{y_k l}(t-1)l(t-1), \tag{7}$$

$$\forall t \geq 1: \quad w_{ij}(t+1) = w_{ij}(t) + \triangle(t)\, g[\, \|adr(w_{ij}) - mod(t)\|^2\,], \tag{8}$$

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

$$= \sum_{i,j} g[\|ana(t) - adr(w_{ij})\|^2] w_{ij}(t). \tag{9}$$

The following aspect of the system dynamics is specific for supervised learning and therefore has not yet been defined in previous sections:

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

$$= d_k(t) - o_k(t) \;\; if \; d_k(t) \; exists, \; and \; 0 \; else. \tag{10}$$

Objective function. As with typical supervised sequence-learning tasks, we want to minimize

$$E^{total}(n_r n_s), \quad where \; E^{total}(t) = \sum_{\tau=1}^{t} E(\tau),$$

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

Note that elements of algorithm space are evaluated solely by a conventional evaluation function[3].

The following algorithm for minimizing E^{total} is partly inspired by (but more complex than) conventional recurrent network algorithms (e.g. [4], [10], [2], [3], [13]).

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

$$w_{ab}(1) \leftarrow w_{ab}(1) - \eta \frac{\partial E^{total}(n_r n_s)}{\partial w_{ab}(1)}, \tag{11}$$

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 (7)-(10). To reduce writing effort, I introduce some short-hand notation partly inspired by [11]:

For all units u and all weights w_{ab} we write

$$p^u_{ab}(t) = \frac{\partial u(t)}{\partial w_{ab}(1)}. \tag{12}$$

For all pairs of connections (w_{ij}, w_{ab}) we write

$$q^{ij}_{ab}(t) = \frac{\partial w_{ij}(t)}{\partial w_{ab}(1)}. \tag{13}$$

First note that

$$\frac{\partial E^{total}(1)}{\partial w_{ab}(1)} = 0, \quad \forall t > 1: \; \frac{\partial E^{total}(t)}{\partial w_{ab}(1)} =$$

$$= \frac{\partial E^{total}(t-1)}{\partial w_{ab}(1)} - \sum_k eval_k(t+1) p^{o_k}_{ab}(t). \tag{14}$$

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.

At time step 1 we have

$$p^{z_k}_{ab}(1) = 0. \tag{15}$$

Now let us focus on time steps after the first one. For $t \geq 1$ we obtain the recursion

$$p^{x_k}_{ab}(t+1) = 0, \tag{16}$$

$$p^{eval_k}_{ab}(t+1) = -p^{o_k}_{ab}(t), \;\; if \; d_k(t) \; exists, \; and \; 0 \; otherwise, \tag{17}$$

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

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

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

$$\times 2 \sum_m (ana_m(t) - adr_m(w_{ij})) p^{ana_m}_{ab}(t) \;\;] \;\; \} \tag{18}$$

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

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

$$f'_{y_k}(net_{y_k}(t+1)) \sum_l w_{y_k l}(t) p^l_{ab}(t) + l(t) q^{y_k l}_{ab}(t), \tag{19}$$

where

$$q^{ij}_{ab}(1) = 1 \; if \; w_{ab} = w_{ij}, \; and \; 0 \; otherwise, \tag{20}$$

$$\forall t > 1: \quad q^{ij}_{ab}(t) =$$

[3]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 [7][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.

$$= \frac{\partial}{\partial w_{ab}(1)} \left[w_{ij}(1) + \sum_{\tau < t} \triangle(\tau) g(\|mod(\tau) - adr(w_{ij})\|^2) \right] =$$

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

$$q_{ab}^{ij}(t-1) + p_{ab}^{\triangle}(t-1) g(\|mod(t-1) - adr(w_{ij})\|^2) +$$
$$2 \triangle (t-1)\ g'(\|mod(t-1) - adr(w_{ij})\|^2) \times$$
$$\times \sum_m [mod_m(t-1) - adr_m(w_{ij})] p_{ab}^{mod_m}(t-1). \qquad (21)$$

According to equations (15)-(21), the $p_{ab}^{j}(t)$ and $q_{ab}^{ij}(t)$ can be updated incrementally at each time step. This implies that (14) 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 is $O(n_{conn}^2 log n_{conn})$.

Again: The initial learning algorithm uses gradient descent to find weight matrices that minimize a conventional error function. This is partly done just like with conventional recurrent net weight changing algorithms. Unlike with conventional networks, however, the network algorithms themselves may choose to change some of the network weights in a manner unlike gradient descent (possibly by doing something smarter than that) — but only if this helps to minimize E^{total}. In other words, the 'self-referential' aspects of the architecture *may* be used by certain 'self-modifying' algorithms to generate desirable evaluations. Therefore the whole system may be viewed as a 'self-referential' *augmentation* of conventional recurrent nets. Further speed-ups (like the ones in [11], [13], [6]) are possible but not essential for the purposes of this paper.

4 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.

Biological plausibility. It seems that I cannot explicitly tell each of my 10^{15} synapses to adopt a certain value. I seem able only to affect my own synapses indirectly — for instance, by somehow actively creating 'keys' and 'entries' to be associated with each other. Therefore, at first glance, the neural net in my head appears to embody a different kind of self-reference than the artificial net of section 2.1 and 2.2. But does it really? The artificial net also does not have a concept of its n-th weight. All it can do is to find out how to talk about weights in terms of activations — without really knowing what a weight is (just like humans who did not know for a long time what synapses are). Therefore I cannot see any evidence that brains use fundamentally different kinds of 'introspective' algorithms. On the other hand, I am not aware of any biological evidence supporting the theory that brains have some means for addressing single synapses[4].

Ongoing and future research. Due to the complexity of the activation dynamics of the 'self-referential' network, one would expect the error function derived in section 3 to have many local minima. [9] 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. In addition, its initial learning algorithm has lower computational complexity than the one from section 3. In fact, this opens the door for another interesting motivation of recurrent networks that can actively change their own weights: It is possible to derive exact gradient-based learning algorithms for such networks that have a lower ratio between learning complexity and number of time varying variables than existing recurrent net learning algorithms [9].

A major criticism of the learning algorithm in section 3 is that it is based on the concept of fixed interaction sequences. All the hard-wired learning algorithms does is find *initial* weights leading to 'desirable' cumulative evaluations. After each interaction sequence, the *final* weight-matrix (obtained through self-modification) is essentially thrown away. *A simple alternative would be to run (after each interaction sequence) the final weight matrix against the best algorithm so far and keep it if it is better*[5]. Again, performance cannot get worse but can only improve over time. I would, however, prefer a hypothetical 'self-referential' learning system that is not initially based on the concept of training sequences at all. Instead, the system should be able to learn to actively segment a single continuous input stream into useful training sequences. Future research will be directed towards building provably working, hard-wired initial-learning-algorithms for such hypothetical systems.

Although the systems described in this paper have a mechanism for 'self-referential' weight changes, they must still learn to use this mechanism. Experiments are needed to discover how practical an approach this is. This paper[6] does not focus on experimental evaluations; it is intended only to show the theoretical possibility of certain kinds

[4]As mentioned before, however, this paper does not insist on addressing every weight in the system individually. (See again footnote 1.) There are many alternative, sensible ways of choosing g and redefining equations (3) and (5) (e.g. [9]).

[5]With the algorithms of section 3, the weight changes for the *initial* weights (at the beginning of a training sequence) are hard-wired. The alternative idea of testing the final weight matrix (at the end of some sequence) against the best previous weight matrix corresponds to the idea of letting the system change its initial weights, too.

[6]This paper is partly inspired by some older ideas about 'self-referential learning' — [5] describes a 'self-referential' genetic algorithm, as well as a few other 'introspective' systems.

of 'self-referential' weight change algorithms. Experimental evaluations of alternative 'self-referential' architectures will be left for the future.

5 ACKNOWLEDGMENT

Thanks to Mark Ring, Mike Mozer, Daniel Prelinger, Don Mathis, and Bruce Tesar, for helpful comments on drafts of this paper. This research was supported in part by a DFG fellowship to the author, as well as by NSF PYI award IRI–9058450, grant 90–21 from the James S. McDonnell Foundation, and DEC external research grant 1250 to Michael C. Mozer.

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] B. A. Pearlmutter. Learning state space trajectories in recurrent neural networks. *Neural Computation*, 1(2):263–269, 1989.

[3] F. J. Pineda. Time dependent adaptive neural networks. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, pages 710–718. San Mateo, CA: Morgan Kaufmann, 1990.

[4] 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.

[5] J. Schmidhuber. Evolutionary principles in self-referential learning, or on learning how to learn: the meta-meta-... hook. Institut für Informatik, Technische Universität München, 1987.

[6] 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.

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

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

[9] 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.

[10] P. J. Werbos. Generalization of backpropagation with application to a recurrent gas market model. *Neural Networks*, 1, 1988.

[11] 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.

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

[13] R. J. Williams and D. Zipser. Gradient-based learning algorithms for recurrent networks and their computational complexity. In *Back-propagation: Theory, Architectures and Applications*. Hillsdale, NJ: Erlbaum, 1992.