LEARNING TO CONTROL FAST-WEIGHT MEMORIES: AN ALTERNATIVE TO DYNAMIC RECURRENT NETWORKS

(*Neural Computation*, 4(1):131–139, 1992)

Jürgen Schmidhuber*
Institut für Informatik
Technische Universität München
Arcisstr. 21, 8000 München 2, Germany
schmidhu@tumult.informatik.tu-muenchen.de

Abstract

Previous algorithms for supervised sequence learning are based on dynamic recurrent networks. This paper describes an alternative class of gradient-based systems consisting of two *feedforward* nets that learn to deal with temporal sequences using *fast weights*: The first net *learns* to produce context dependent weight *changes* for the second net whose weights may vary very quickly. The method offers the potential for STM storage efficiency: A single weight (instead of a full-fledged unit) may be sufficient for storing temporal information. Various learning methods are derived. Two experiments with unknown time delays illustrate the approach. One experiment shows how the system can be used for *adaptive* temporary variable binding.

1 The Task

A training sequence p with n_p discrete time steps (called an episode) consists of n_p ordered pairs $(x^p(t), d^p(t)) \in R^n \times R^m$, $0 < t \leq n_p$. At time t of episode p a learning system receives $x^p(t)$ as an input and produces the output $y^p(t)$.

*Current address: Dept. of Computer Science, University of Colorado, Campus Box 430, Boulder, CO 80309, USA

The goal of the learning system is to minimize

$$\hat{E} = \frac{1}{2} \sum_p \sum_t \sum_i (d_i^p(t) - y_i^p(t))^2,$$

where $d_i^p(t)$ is the ith of the m components of $d^p(t)$, and $y_i^p(t)$ is the ith of the m components of $y^p(t)$.

In general, this task requires storage of input events in a short-term memory. Previous solutions to this problem have employed gradient-based dynamic recurrent nets (e.g., (Robinson and Fallside, 1987), (Pearlmutter, 1989), (Williams and Zipser, 1989)). In the next section an alternative gradient-based approach is described. For convenience, we drop the indices p which stand for the various episodes.

The gradient of the error over all episodes is equal to the sum of the gradients for each episode. Thus we only require a method for minimizing the error observed during one particular episode:

$$\bar{E} = \sum_t E(t),$$

where $E(t) = \frac{1}{2} \sum_i (d_i(t) - y_i(t))^2$. (In the practical *on-line* version of the algorithm below there will be no episode boundaries; one episode will 'blend' into the next (Williams and Zipser, 1989).)

2 The Architecture and the Algorithm

The basic idea is to use a slowly learning feed-forward network S (with a set of randomly initialized weights W_S) whose input at time t is the vector $x(t)$ and whose output is transformed into immediate weight *changes* for a second 'fast-weight' network F. The input to F at time t is also $x(t)$, its m-dimensional output is $y(t)$, and its set of weight variables is W_F. F serves as a short term memory: At different time steps, the same input event may be processed in different ways depending on the time-varying state of W_F.

The standard method for processing temporal sequences is to employ a recurrent net with feedback connections. The feedback connections allow for a short-term memory of information earlier in a sequence. The present work suggests a novel approach to building a short-term memory by employing fast weights that can be set and reset by the 'memory controller' S. Fast weights can hold on to information over time because they remain essentially invariant unless they are explicitly modified.

One potential advantage of the method over the more conventional recurrent net algorithms is that it does not necessarily require full-fledged units – experiencing some sort of feedback – for storing temporal information. A single weight

may be sufficient. Because there are many more weights than units in most networks, this property represents a potential for storage efficiency. For related reasons, the novel representation of past inputs is well-suited for solving certain problems involving *temporary variable binding* in a natural manner: F's current input may be viewed as a representation of the *addresses* of a set of variables; F's current output may be viewed as the representation of the current *contents* of this set of variables. In contrast with recurrent nets, temporary bindings can be established very naturally by temporary *connectivity patterns* instead of temporary *activation patterns* (see section 3.2 for an illustrative experiment).

For initialization reasons we introduce an additional time step 0 at the beginning of an episode. At time step 0 each weight variable $w_{ab} \in W_F$ of a directed connection from unit a to unit b is set to $\Box w_{ab}(0)$ (a function of S's outputs as described below). At time step $t > 0$, the $w_{ab}(t-1)$ are used to compute the output of F according to the usual activation spreading rules for back-propagation networks (e.g. (Werbos, 1974)). After this, each weight variable $w_{ab} \in W_F$ is altered according to

$$w_{ab}(t) = \sigma(w_{ab}(t-1), \Box w_{ab}(t)), \tag{1}$$

where σ (e.g. a sum-and-squash function) is differentiable with respect to all its parameters and where the activations of S's output units (again computed according to the usual activation spreading rules for back-propagation networks) serve to compute $\Box w_{ab}(t)$ by a mechanism specified below. $\Box w_{ab}(t)$ is S's contribution to the modification of w_{ab} at time step t.

Equation (1) is essentially identical to Möller and Thrun's equation (1) in (Möller and Thrun, 1990). Unlike (Möller and Thrun, 1990), however, the current paper derives an exact gradient descent algorithm for time-varying inputs and outputs for this kind of architecture.

For all weights $w_{ij} \in W_S$ (from unit i to unit j) we are interested in the increment

$$\triangle w_{ij} = -\eta \frac{\partial \bar{E}}{\partial w_{ij}} = -\eta \sum_{t>0} \frac{\partial E(t)}{\partial w_{ij}} = -\eta \sum_{t>0} \sum_{w_{ab} \in W_F} \frac{\partial E(t)}{\partial w_{ab}(t-1)} \frac{\partial w_{ab}(t-1)}{\partial w_{ij}}. \tag{2}$$

Here η is a constant learning rate. At each time step $t > 0$, the factor

$$\delta_{ab}(t) = \frac{\partial E(t)}{\partial w_{ab}(t-1)}$$

can be computed by conventional back-propagation (e.g. (Werbos, 1974)). For $t > 0$ we obtain the recursion

$$\frac{\partial w_{ab}(t)}{\partial w_{ij}} = \frac{\partial \sigma(w_{ab}(t-1), \Box w_{ab}(t))}{\partial w_{ab}(t-1)} \frac{\partial w_{ab}(t-1)}{\partial w_{ij}} + \frac{\partial \sigma(w_{ab}(t-1), \Box w_{ab}(t))}{\partial \Box w_{ab}(t)} \frac{\partial \Box w_{ab}(t)}{\partial w_{ij}}.$$

We can employ a method similar to the one described in (Robinson and Fallside, 1987) and (Williams and Zipser, 1989): For each $w_{ab} \in W_F$ and each $w_{ij} \in W_S$

we introduce a variable p_{ij}^{ab} (initialized to zero at the beginning of an episode) which can be updated at each time step $t > 0$:

$$p_{ij}^{ab}(t) = \frac{\partial \sigma(w_{ab}(t-1), \Box w_{ab}(t))}{\partial w_{ab}(t-1)} p_{ij}^{ab}(t-1) + \frac{\partial \sigma(w_{ab}(t-1), \Box w_{ab}(t))}{\partial \Box w_{ab}(t)} \frac{\partial \Box w_{ab}(t)}{\partial w_{ij}}. \tag{3}$$

$\frac{\partial \Box w_{ab}(t)}{\partial w_{ij}}$ depends on the interface between S and F. With a given interface (two possibilities are given below) an appropriate back-propagation procedure for each $w_{ab} \in W_F$ gives us $\frac{\partial \Box w_{ab}(t)}{\partial w_{ij}}$ for all $w_{ij} \in W_S$. After having updated the p_{ij}^{ab} variables, (2) can be computed using the formula

$$\frac{\partial E(t)}{\partial w_{ij}} = \sum_{w_{ab} \in W_F} \delta_{ab}(t) p_{ij}^{ab}(t-1).$$

A simple interface between S and F would provide one output unit $s_{ab} \in S$ for each weight variable $w_{ab} \in W_F$, where

$$\Box w_{ab}(t) := s_{ab}(t), \tag{4}$$

$s_{ab}(t)$ being the output unit's activation at time $t \geq 0$.

A disadvantage of (4) is that the number of output units in S grows in proportion to the number of weights in F. An alternative is the following: Provide an output unit in S for each unit in F from which at least one fast weight originates. Call the set of these output units *FROM*. Provide an output unit in S for each unit in F to which at least one fast weight leads. Call the set of these output units *TO*. For each weight variable $w_{ab} \in W_F$ we now have a unit $s_a \in FROM$ and a unit $s_b \in TO$. At time t, define $\Box w_{ab}(t) := g(s_a(t), s_b(t))$, where g is differentiable with respect to all its parameters. As a representative example we will focus on the special case of g being the multiplication operator:

$$\Box w_{ab}(t) := s_a(t) s_b(t). \tag{5}$$

Here the fast weights in F are manipulated by the outputs of S in a Hebb-like manner, assuming that σ is just a sum-and-squash function as employed in the experiments described below.

One way to interpret the *FROM/TO* architecture is to view S as a device for creating temorary associations by giving two parameters to the short term memory: The first parameter is an activation pattern over *FROM* representing a *key* to a temporary association pair, the second parameter is an activation pattern over *TO* representing the corresponding *entry*. Note that both key and entry may involve hidden units.

(4) and (5) differ in the way that error signals are obtained at S's output units: If (4) is employed, then we use conventional back-propagation to compute $\frac{\partial s_{ab}(t)}{\partial w_{ij}}$ in (3). If (5) is employed, note that

$$\frac{\partial \Box w_{ab}(t)}{\partial w_{ij}} = s_b(t) \frac{\partial s_a(t)}{\partial w_{ij}} + s_a(t) \frac{\partial s_b(t)}{\partial w_{ij}}. \tag{6}$$

Conventional back-propagation can be used to compute $\frac{\partial s_a(t)}{\partial w_{ij}}$ for each output unit a and for all w_{ij}. The results can be kept in $\mid W_S \mid * \mid FROM \cup TO \mid$ variables. This makes it easy to solve (6) in a second pass.

The algorithm is *local in time*, its update-complexity per time step is $O(\mid W_F \mid\mid W_S \mid)$. However, it is not local in space (see (Schmidhuber, 1990b) for a definition of locality in space and time).

2.1 On-Line Versus Off-Line Learning

The *off-line* version of the algorithm would wait for the end of an episode to compute the final change of W_S as the sum of all changes computed at each time step. The *on-line* version changes W_S at every time step, assuming that η is small enough to avoid instabilities (Williams and Zipser, 1989). An interesting property of the *on-line* version is that we do not have to specify episode boundaries ('all episodes blend into each other' (Williams and Zipser, 1989)).

2.2 Unfolding in time

An alternative of the method above would be to employ a method similar to the 'unfolding in time'-algorithm for recurrent nets (e.g. (Rumelhart et al., 1986)). It is convenient to keep an activation stack for each unit in S. At each time step of an episode, some unit's new activation should be pushed onto its stack. S's output units should have an additional stack for storing sums of error signals received over time. With both (4) and (5), at each time step we essentially propagate the error signals obtained at S's output units down to the input units. The final weight change of W_S is proportional to the sum of all contributions of all errors observed during one episode. The complete gradient for S is computed at the end of each episode by successively popping off the stacks of error signals and activations analogously to the 'unfolding in time'-algorithm for recurrent networks. A disadvantage of the method is that it is not local in space.

2.3 Limitations and Extensions

When both F and S are feedforward networks, the technique proposed above is limited to only certain types of time-varying behavior. With σ being a sum-and-squash function, the only kind of interesting time-varying output that can be produced is in response to variations in the input; in particular, autonomous dynamical behavior like oscillations (e.g. (Williams and Zipser, 1989)) cannot be performed while the input is held fixed.

It is straight-forward to extend the system above to the case where both S and F are recurrent. In the experiment below S and F are *non-recurrent*, mainly to demonstrate that even a feed-forward system employing the principles above can solve certain tasks that only recurrent nets were supposed to solve.

The method can be accelerated by a procedure analogous to the one presented in (Schmidhuber, 1991b).

3 Experiments

The following experiments were conducted in collaboration with Klaus Bergner, a student at Technische Universität München.

3.1 An Experiment With Unknown Time Delays

In this experiment, the system was presented with a continuous stream of input events and F's task was to switch on the single output unit the first time an event 'B' occurred following an event 'A'. At all other times, the output unit was to be switched off. This is the flip-flop task described in (Williams and Zipser, 1989).

One difficulty with this task is that there can be arbitrary time lags between relevant events. An additional difficulty is that no information about 'episode boundaries' is given. The on-line method was employed: The activations of the networks were never reset. Thus, activations caused by events from past 'episodes' could have a harmful effect on activations and weights in later episodes.

Both F and S had the topology of standard feedforward perceptrons. F had 3 input units for 3 possible events 'A', 'B', and 'C'. Events were represented in a local manner: At a given time, a randomly chosen input unit was activated with a value of 1.0, the others were de-activated. F's output was one-dimensional. S also had 3 input units for the possible events 'A', 'B', and 'C', as well as 3 output units, one for each fast weight of F. Neither of the networks needed hidden units for this task. The activation function of all output units was the identity function. The weight-modification function (1) for the fast weights was given by

$$\sigma(w_{ab}(t-1), \Box w_{ab}(t)) = (1 + e^{-T(w_{ab}(t-1)+\Box w_{ab}(t)-0.5)})^{-1}. \qquad (7)$$

Here T determines the maximal steepness of the logistic function used to bound the fast weights between 0 and 1.

The weights of S were randomly initialized between -0.1 and 0.1. The task was considered to be solved if for 100 time steps in a row F's error did not exceed 0.05. With fast-weight changes based on (4), $T = 10$ and $\eta = 1.0$ the system learned to solve the task within 300 time steps. With fast-weight changes based on the *FROM/TO*-architecture and (5), $T = 10$ and $\eta = 0.5$ the system learned to solve the task within 800 time steps.

The typical solution to this problem has the following properties: When an A-signal occurs, S responds by producing a large weight on the B input line of F (which is otherwise small), thus enabling the F network as a B detector.

When a B signal occurs, S 'resets' F by causing the weight on the B line in F to become small again, thereby making F unresponsive to further B signals until the next A is received.

3.2 Learning Temporary Variable Binding

Some researchers have claimed that neural nets are incapable of performing variable binding. Others, however, have argued for the potential usefulness of 'dynamic links' (e.g. (v.d. Malsburg, 1981)), which may be useful for variable binding. With the fast-weight method, it is possible to *train* a system to use fast weights as dynamic links in order to temporarily bind variable contents to variable names (or 'fillers' to 'slots') as long as it is necessary for solving a particular task.

In the simple experiment described next, the system learns to remember where in a parking lot a car has been left. This involves binding a value in a variable that represents the car's location.

Neither F nor S needed hidden units for this task. The activation function of all output units was the identity function. All inputs to the system were binary, as were F's desired outputs. F had one input unit which stood for the name of the variable WHERE-IS-MY-CAR?. In addition, F had three output units for the names of three possible parking slots P_1, P_2, and P_3 (the possible answers to WHERE-IS-MY-CAR?). S had three output units, one for each fast weight, and six input units. (Note that S need not always have the same input as F.) Three of the 6 input units were called the parking-slot detectors – I_1, I_2, I_3. These detectors were activated for one time step when the car was parked in a given slot (while the other slot-detectors remained switched off). The three additional input units were randomly activated with binary values at each time step. These random activations served as distracting time varying inputs from the environment of a car owner whose life looks like this: He drives his car around for zero or more time steps (at each time step the probability that he stops driving is 0.25). Then he parks his car in one of three possible slots. Then he conducts business outside the car for zero or more time steps during which all parking-slot-detectors are switched off again (at each time step the probability that he finishes business is 0.25). Then he remembers where he has parked his car, goes to the corresponding slot, enters his car and starts driving again etc.

Our system focussed on the problem of remembering the position of the car. It was trained by activating the WHERE-IS-MY-CAR? unit at randomly chosen time steps and by providing the desired output for F, which was the activation of the unit corresponding to the current slot P_i, as long as the car was parked in one of the three slots.

The weights of S were randomly initialized between -0.1 and 0.1. The task was considered to be solved if for 100 time steps in a row F's error did not exceed 0.05. The on-line version (without episode boundaries) was employed. With the weight-modification function (7), fast-weight changes based on (4), $T = 10$ and

$\eta = 0.02$ the system learned to solve the task within 6000 time steps. As it was expected, S learned to 'bind' parking slot units to the WHERE-IS-MY-CAR?-unit by means of strong temporary fast-weight connections. Due to the local output representation, the binding patterns were easy to understand: At a given time there was a large fast weight on the connection leading from the WHERE-IS-MY-CAR?-unit to the appropriate parking slot unit (given the car was currently parked). The other fast-weights remained temporarily suppressed.

4 Concluding Remarks

The system described above is a special case of a more general class of adaptive systems (which also includes conventional recurrent nets) which employ some parameterized *memory* function for changing a vector-valued memory structure and which employ some parameterized *retrieval* function for *processing* the contents of the memory structure and the current input. The only requirement is that the memory and retrieval functions be differentiable with respect to their internal parameters.

Such systems work because of the existence of the *chain rule*. Results as above (as well as other novel applications of the chain rule (Schmidhuber, 1991a)(Schmidhuber, 1990a)) indicate that there may be additional interesting (yet undiscovered) ways of applying the chain rule for temporal credit assignment in adaptive systems.

5 Acknowledgements

I wish to thank Klaus Bergner for conducting the experiments. Furthermore I wish to thank Mike Mozer, Bernhard Schätz, and Jost Bernasch for providing comments on a draft of this paper.

References

Möller, K. and Thrun, S. (1990). Task modularization by network modulation. In Rault, J., editor, *Proceedings of Neuro-Nimes '90*, pages 419–432.

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

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

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning internal representations by error propagation. In Rumelhart, D. E. and McClelland,

J. L., editors, *Parallel Distributed Processing*, volume 1, pages 318–362. MIT Press.

Schmidhuber, J. H. (1990a). Dynamische neuronale Netze und das fundamentale raumzeitliche Lernproblem. Dissertation, Institut für Informatik, Technische Universität München.

Schmidhuber, J. H. (1990b). Learning algorithms for networks with internal and external feedback. In Touretzky, D. S., Elman, J. L., Sejnowski, T. J., and Hinton, G. E., editors, *Proc. of the 1990 Connectionist Models Summer School*, pages 52–61. San Mateo, CA: Morgan Kaufmann.

Schmidhuber, J. H. (1991a). Learning to generate sub-goals for action sequences. In Simula, O., editor, *Proceedings of the International Conference on Artificial Neural Networks ICANN 91, to appear*. Elsevier Science Publishers B.V.

Schmidhuber, J. H. (1991b). An $O(n^3)$ learning algorithm for fully recurrent networks. Technical Report FKI-151-91, Institut für Informatik, Technische Universität München.

v.d. Malsburg, C. (1981). Internal Report 81-2, Abteilung für Neurobiologie, Max-Planck Institut für Biophysik und Chemie, Göttingen.

Werbos, P. J. (1974). *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. PhD thesis, Harvard University.

Williams, R. J. and Zipser, D. (1989). Experimental analysis of the real-time recurrent learning algorithm. *Connection Science*, 1(1):87–111.