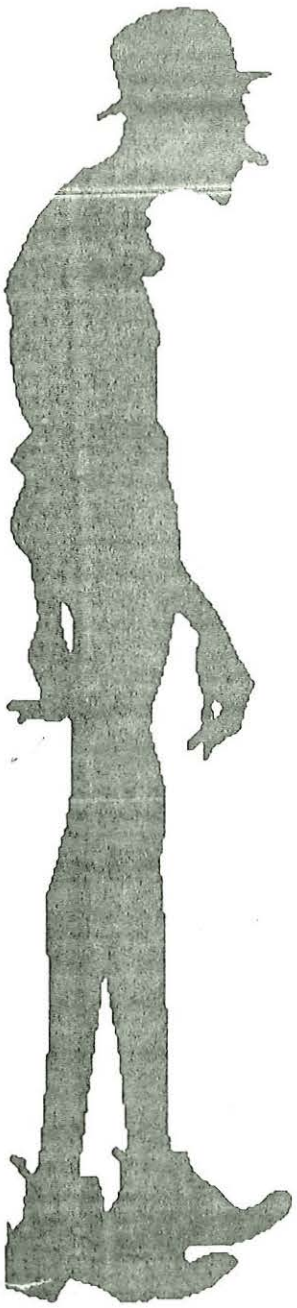
FORSCHUNGSBERICHTE KÜNSTLICHE INTELLIGENZ



Learning to Control Fast-Weight Memories: An Alternative to Dynamic Recurrent Networks

Jürgen Schmidhuber

Report FKI-147-91

März 1991

T U M

TECHNISCHE UNIVERSITÄT MÜNCHEN
Institut für Informatik, Arcisstr. 21, 8000 München 2, Germany

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

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

Technical Report FKI-147-91, March 26, 1991

Abstract

Previous algorithms for supervised sequence learning are based on dynamic recurrent networks. This paper describes alternative gradient-based systems consisting of two *feed-forward* nets which learn to deal with temporal sequences by 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 a potential for STM storage efficiency: A simple 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 $(x^p(t), d^p(t)) \in R^n x R^m$, $0 < t \leq n_p$. At time t of episode p a learning system receives $x^p(t)$ as an input and produces the output $y^p(t)$. 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 to memorize input events in short term memory. Previous approaches to solving this problem employed dynamic recurrent nets (e.g. [2], [12], [1], [13], [6]). In the next section an alternative gradient-based approach is described. For convenience, in what follows we will drop the indices p which stand for various episodes: The gradient of the error sum over all episodes is equal to the sum of the corresponding gradients. Thus we are interested in 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 we will not have any episode boundaries at all; all episodes will 'blend into each other' [13].)

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 (potentially very significant) weight *changes* for a second 'fast-weight' network F. F's input at time t is $x(t)$, its m-dimensional output is $y(t)$, and the set of its 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.

One potential advantage of the method over the more conventional recurrent net algorithms is the following: It does not necessarily occupy full-fledged units (experiencing some sort of feedback) for storing information over time. A simple weight may be sufficient for storing temporal information. Since with most networks there are many more weights than units, this property represents a potential for storage efficiency.

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 $\Box w_{ab}(0)$ (to be computed by 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. [10]). 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' 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 to be specified below (we will consider two alternatives). $\Box w_{ab}(t)$ is S' contribution to the modification of w_{ab} at time step t.

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

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

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. [10]). 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 [2] and [13]: 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, for solving (2) we compute

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

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

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

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 is originating. 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 is leading. 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.

(4) and (5) differ in the way that error signals are obtained at 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 $\mid W_S \mid *c$ variables (here c is the number of units in $FROM \cup TO$). This makes it easy to solve (6) in a second pass.

The algorithm is *local in time*, its update-complexity per time step is $O(\mid W_F \mid\mid W_S \mid)$. But, it is not local in space (see [5] 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 [13]. An interesting property of the *on-line* version is that we do not have to specify episode boundaries ('all episodes blend into each other' [13]).

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 [3] [11]. 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' 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' 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 of 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 Recurrent Slow-Weight and Fast-Weight Networks

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 a task that only recurrent nets were supposed to solve.

3 Experiments

The following experiments were conducted by Klaus Bergner, a student at TUM.

3.1 An Experiment With Unknown Time Delays

F had to learn to behave like a flip-flop as described in [13]. F saw a continuous stream of input events. The task was to switch on the single output unit whenever an event 'B' occurred for the first time after the last event 'A' had happened. In all other cases the output unit had to be switched off.

One difficulty with the problem was that there could be arbitrary time lags between relevant events. An additional difficulty was that no information about 'episode boundaries' was 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 disturbing influence on activations and weights appearing during 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 normal 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. None of the networks needed any 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

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

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.

3.2 Learning Temporary Variable Binding

When connectionism was young some people have claimed that neural nets are not capable of variable binding. Others, however, have argued for the potential usefulness of 'dynamic links' (e.g. [9]), which may be useful for variable binding. With the method above it is possible to *train* an appropriate system to use its dynamic links (= fast weights) for temporarily binding 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 learned to bind a variable for storing the position of a car to time-varying parking slots.

Neither F nor S needed any hidden units for this task. The activation function of all output units was the identity function. All inputs to the system were binary, and so 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 contents of WHERE-IS-MY-CAR?). S had three output units, one for each fast weight, and six input units (here we 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, the remaining three were randomly activated with binary values at each time step. These random activations were interpreted 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. He notices the name I_i of the parking slot (this takes him one time step, during which input unit I_i is briefly activated, while the other slot-detectors remain switched off). Then he makes

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 memorizing the position of the car. It was trained by activating the WHERE-IS-MY-CAR?-unit at randomly chosen time steps in the life of the car owner 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 stood 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.

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 (differentiable with respect to all its parameters) for changing a vector-valued memory structure and which employ some parameterized *retrieval* function (again differentiable with respect to all its parameters) for *processing* the content of the memory structure and the current input.

Such systems can work because of the existence of the *chain rule*. Results as above (as well as other novel applications of the chain rule [7][8][4]) indicate that there may be additional interesting (yet undiscovered) ways of applying the chain rule for temporal credit assignment in adaptive systems.

References

[1] B. A. Pearlmutter. Learning state space trajectories in recurrent neural networks. *Neural Computation*, 1:263–269, 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 D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing*, volume 1, pages 318–362. MIT Press, 1986.

[4] J. H. Schmidhuber. Dynamische neuronale Netze und das fundamentale raumzeitliche Lernproblem. Dissertation, Institut für Informatik, Technische Universität München, 1990.

[5] J. H. Schmidhuber. Learning algorithms for networks with internal and external feedback. In *Proc. of the 1990 Connectionist Models Summer School*, pages 52–61. San Mateo, CA: Morgan Kaufmann, 1990.

[6] J. H. Schmidhuber. A local learning algorithm for dynamic feedforward and recurrent networks. *Connection Science*, 1(4):403–412, 1990.

[7] J. H. Schmidhuber. Towards compositional learning with dynamic neural networks. Technical Report FKI-129-90, Institut für Informatik, Technische Universität München, 1990.

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

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

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

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

[12] R. J. Williams. Toward a theory of reinforcement-learning connectionist systems. Technical Report NU-CCS-88-3, College of Comp. Sci., Northeastern University, Boston, MA, 1988.

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