

LEARNING TEMPORARY VARIABLE BINDING WITH DYNAMIC LINKS

In Proc. International Joint Conference on Neural Networks, Singapore, volume 3, pages 2075-2079. IEEE, 1991.

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

Abstract

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. [8]) for variable binding. With the novel method described in this paper it is possible to *train* a system with time-varying inputs and outputs to use its dynamic links for temporarily binding variable contents to variable names (or ‘fillers’ to ‘slots’) as long as it is necessary for solving a particular task. Various learning methods for non-stationary environments are derived. Two experiments with unknown time delays illustrate the approach. A by-product of this work is the demonstration that a system consisting of two feed-forward networks can solve tasks that only dynamic recurrent networks were supposed to solve.

1 INTRODUCTION

This paper describes a novel gradient-based system for processing sequential time-varying inputs and outputs. The system consists of two *feed-forward* nets which learn to deal with temporal sequences by using *dynamic links*: The first net *learns* to produce context dependent weight *changes* for the second net whose weights may vary very quickly. The second net serves as a short term memory: Its purpose is to temporarily bind *values* to *variables* as long as necessary for solving particular tasks.

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 $y^p(t)$. 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 i th of the m components of $d^p(t)$, and $y_i^p(t)$ is the i th 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 gradient-based dynamic recurrent nets (e.g. [3], [2], [10]). One potential advantage of the alternative gradient-based approach described in the next section over the 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

*This work has been done at Technische Universität München, Germany

information. Since with most networks there are many more weights than units, this property represents a potential for storage efficiency.

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’ [10].)

2 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 network F with dynamic links. 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 .

At a given time, W_F may be interpreted as a *temporary binding pattern*: F ’s current input pattern may be interpreted as the (distributed) name of a variable or of a set of variables; F ’s corresponding output pattern may be interpreted as the current value(s) of this variable (or set of variables). How new inputs are processed depends on the current binding pattern which may have been caused by previous input events.

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 $\square w_{ab}(0)$ (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. [9]). After this, each weight variable $w_{ab} \in W_F$ is altered according to

$$w_{ab}(t) = \sigma(w_{ab}(t-1), \square w_{ab}(t)), \quad (1)$$

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 $\square w_{ab}(t)$ by a mechanism to be specified below (we will consider two alternatives). $\square w_{ab}(t)$ is 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 [1]. Unlike [1], 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

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

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. [9]). For $t > 0$ we obtain the recursion

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

We can employ a method similar to the one described in [3] and [10]: 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 $t > 0$:

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

$\frac{\partial \square w_{ab}(t)}{\partial w_{ij}}$ 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 $\frac{\partial \square w_{ab}(t)}{\partial w_{ij}}$ 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

$$\square w_{ab}(t) = s_{ab}(t). \quad (4)$$

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 'dynamic link' 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 'dynamic link' 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 $\square w_{ab}(t) = g(s_a(t), s_b(t))$, 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:

$$\square w_{ab}(t) = s_a(t) s_b(t). \quad (5)$$

Here the dynamic links 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

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

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 $|W_S| * c$ 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 $O(|W_F| |W_S|)$. But, it is not local in space (see [6] 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 [10]. An interesting property of the *on-line* version is that we do not have to specify episode boundaries ('all episodes blend into each other' [10]).

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. [4]). 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.

3 EXPERIMENTS

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

1. F had to learn to behave like a flip-flop as described in [10]. 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 dynamic link 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 dynamic links was given by

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

Here T determines the maximal steepness of the logistic function used to bound the dynamic links 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 dynamic link changes based on (4), $T = 10$ and $\eta = 1.0$ the system learned to solve the task within 300 time steps. With dynamic link 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.

After learning, at a given time step the current activation of F ’s output unit could be interpreted as the current *value* of a variable which memorized whether there has been an A since the last occurrence of B . One could say that the input unit for B -events evolved to become the *address* of this variable: By activating the B -input unit and watching the result at F ’s output it was possible to tell whether there has been an A since the last occurrence of B or not.

2. 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 dynamic link, 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), dynamic link 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 dynamic links.

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][5]) indicate that there may be additional interesting (yet undiscovered) ways of applying the chain rule for temporal credit assignment in adaptive systems.

5 ACKNOWLEDGEMENTS

Thanks to Klaus Bergner for conducting the experiments.

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] 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.
- [4] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In *Parallel Distributed Processing*, volume 1, pages 318–362. MIT Press, 1986.
- [5] J. Schmidhuber. Dynamische neuronale Netze und das fundamentale raumzeitliche Lernproblem. Dissertation, Institut für Informatik, Technische Universität München, 1990.

- [6] J. Schmidhuber. Learning algorithms for networks with internal and external feedback. In D. S. Touretzky, J. L. Elman, T. J. Sejnowski, and G. E. Hinton, editors, *Proc. of the 1990 Connectionist Models Summer School*, pages 52–61. San Mateo, CA: Morgan Kaufmann, 1990.
- [7] J. Schmidhuber. Learning to generate sub-goals for action sequences. In T. Kohonen, K. Mäkisara, O. Simula, and J. Kangas, editors, *Artificial Neural Networks*, pages 967–972. Elsevier Science Publishers B.V., North-Holland, 1991.
- [8] C. v.d. Malsburg. Technical Report 81-2, Abteilung für Neurobiologie, Max-Planck Institut für Biophysik und Chemie, Göttingen, 1981.
- [9] P. J. Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. PhD thesis, Harvard University, 1974.
- [10] R. J. Williams and D. Zipser. Experimental analysis of the real-time recurrent learning algorithm. *Connection Science*, 1(1):87–111, 1989.