LEARNING COMPLEX, EXTENDED SEQUENCES USING THE PRINCIPLE OF HISTORY COMPRESSION

(*Neural Computation*, 4(2):234–242, 1992)

Jürgen Schmidhuber*
Institut für Informatik
Technische Universität München
Arcisstr. 21, 8000 München 2, Germany

Abstract

Previous neural network learning algorithms for sequence processing are computationally expensive and perform poorly when it comes to *long* time lags. This paper first introduces a simple principle for reducing the descriptions of event sequences *without loss of information*. A consequence of this principle is that only *unexpected* inputs can be relevant. This insight leads to the construction of neural architectures that learn to 'divide and conquer' by recursively decomposing sequences. I describe two architectures. The first functions as a self-organizing multi-level hierarchy of recurrent networks. The second, involving only two recurrent networks, tries to collapse a multi-level predictor hierarchy into a single recurrent net. Experiments show that the system can require less computation per time step *and* many fewer training sequences than conventional training algorithms for recurrent nets.

1 INTRODUCTION

Several approaches to on-line supervised sequence learning have been proposed, including back-propagation through time or BPTT, e.g. (Williams and Peng, 1990), the IID- or RTRL-algorithm (Robinson and Fallside, 1987)(Williams and Zipser, 1989), and the recent fast-weight algorithm (Schmidhuber, 1991b)). These approaches are computationally intensive; BPTT is not local in time, RTRL-like algorithms and also their more efficient recent relatives (Schmidhuber, 1991d) are not local in space (Schmidhuber, 1991c). Common to all of these approaches is that they do not try to selectively focus on *relevant* inputs; they waste efficiency and resources by focussing on *every* input. With many applications, a second drawback of these methods is the following: The longer the time lag between an event and the occurrence of a corresponding error the less information is carried by the corresponding back-propagated error signals. (Mozer, 1990) and (Rohwer, 1989) have addressed the latter problem but not the former.

How can a system *learn* to focus on the *relevant* points in time? What does it mean for a point in time to be relevant? How can the system learn to reduce the numbers of inputs to be considered over time *without losing information*? A major contribution of this work is an adaptive method for removing redundant information from sequences. The next section shows that the system ought to focus on *unexpected* inputs and ignore *expected* ones.

2 HISTORY COMPRESSION

Consider a deterministic discrete time predictor (not necessarily a neural network) whose state at time t is described by an environmental input vector $i(t)$, an internal state vector $h(t)$, and an output vector $o(t)$.

*Current address: Dept. of Computer Science, University of Colorado, Campus Box 430, Boulder, CO 80309, USA, yirgan@cs.colorado.edu

The environment may be non-deterministic. At time 0, the predictor starts with $i(0)$ and an internal start state $h(0)$. At time $t \geq 0$, the predictor computes

$$o(t) = f(i(t), h(t)).$$

At time $t > 0$, the predictor furthermore computes

$$h(t) = g(i(t-1), h(t-1)).$$

All information about the input at a given time t_x can be reconstructed from the knowledge about

$$t_x, f, g, i(0), h(0), \textit{ and the pairs } (t_s, i(t_s)) \textit{ for which } 0 < t_s \leq t_x \textit{ and } o(t_s - 1) \neq i(t_s).$$

This is because if $o(t) = i(t+1)$ at a given time t, then the predictor is able to predict the next input from the previous ones. The new input is *derivable* by means of f and g.

Information about the observed input sequence can be even further compressed beyond just the unpredicted input vectors $i(t_s)$. It suffices to know only those *elements* of the vectors $i(t_s)$ that were not correctly predicted.

This observation implies that we can discriminate one sequence from another by knowing *just the unpredicted inputs and the corresponding time steps at which they occurred.* No information is lost if we ignore the expected inputs. We do not even have to know f and g. We call this *the principle of history compression.*

From a theoretical point of view it is important to know at what time an unexpected input occurs; otherwise there will be a potential for ambiguities: Two different input sequences may lead to the same shorter sequence of unpredicted inputs. With many practical tasks, however, there is no need for knowing the critical time steps, as I show later.

3 A SELF-ORGANIZING MULTI-LEVEL PREDICTOR HIERARCHY

Using the principle of history compression we can build a self-organizing hierarchical neural 'chunking' system. The system detects causal dependencies in the temporal input stream and learns to attend to unexpected inputs instead of focussing on every input. It learns to reflect both the relatively local and the relatively global temporal regularities contained in the input stream.

The basic task can be formulated as a prediction task. At a given time step the goal is to predict the next input from previous inputs. If there are external target vectors at certain time steps then they are simply treated as another part of the input to be predicted.

The architecture is a hierarchy of predictors, the input to each level of the hierarchy is coming from the previous level. P_i denotes the ith level network which is trained to *predict its own next input from its previous inputs*[1]. We take P_i to be a conventional dynamic recurrent neural network (Robinson and Fallside, 1987)(Williams and Zipser, 1989)(Williams and Peng, 1990)(Schmidhuber, 1991d); however, it might be some other adaptive sequence processing device as well[2].

At each time step the input of the lowest-level recurrent predictor P_0 is the current external input. We create a new higher-level adaptive predictor P_{s+1} whenever the adaptive predictor at the previous level, P_s, stops improving its predictions. When this happens the weight-changing mechanism of P_s is switched off (to exclude potential instabilities caused by ongoing modifications of the lower-level predictors). If at a given time step P_s ($s \geq 0$) fails to predict its next input (or if we are at the beginning of a training sequence which usually is not predictable either) then P_{s+1} will receive as input the concatenation of this next input of P_s *plus a unique representation of the corresponding time step*[3]; the activations of P_{s+1}'s hidden and

[1] Recently I became aware that Don Mathis had some related ideas (personal communication). A hierarchical approach to sequence *generation* was pursued by (Miyata, 1988).

[2] For instance, we might employ the more limited feedforward networks and a 'time window' approach. In this case, the number of previous inputs to be considered as a basis for the next prediction will remain fixed.

[3] A unique time representation is theoretically necessary to provide P_{s+1} with unambiguous information about when the failure occurred (see also the last paragraph of section 2). A unique representation of the time that went by since the *last* unpredicted input occurred will do as well.

output units will be updated. Otherwise P_{s+1} will not perform an activation update. This procedure ensures that P_{s+1} is fed with an *unambiguous* reduced description[4] of the input sequence observed by P_s. This is theoretically justified by the principle of history compression.

In general, P_{s+1} will receive fewer inputs over time than P_s. With existing learning algorithms, the higher-level predictor should have less difficulties in learning to predict the critical inputs than the lower-level predictor. This is because P_{s+1}'s 'credit assignment paths' will often be short compared to those of P_s. This will happen if the incoming inputs carry global temporal structure which has not yet been discovered by P_s.

This method is a simplification and an improvement of the recent chunking method described by (Schmidhuber, 1991a).

Often a multi-level predictor hierarchy will be the fastest way of learning to deal with sequences with multi-level temporal structure (e.g speech). Experiments have shown that multi-level predictors can quickly learn tasks which are practically unlearnable by conventional recurrent networks, e.g. (Hochreiter, 1991). One disadvantage of a predictor hierarchy, however, is that it is not known in advance how many levels will be needed. Another disadvantage is that levels are explicitly separated from each other. It can be possible, however, to collapse the hierarchy into a single network as described next.

4 COLLAPSING THE HIERARCHY INTO A SINGLE RECURRENT NET

4.1 OUTLINE

I now describe an architecture consisting of two conventional recurrent networks: The *automatizer* A and the *chunker* C. At each time step A receives the current external input. A's error function is threefold: One term forces it to emit certain desired target outputs at certain times. If there is a target, then it becomes part of the next input. The second term forces A at every time step to predict its own next non-target input. The third (crucial) term will be explained below.

If and only if A makes an error concerning the first and second term of its error function, the unpredicted input (including a potentially available teaching vector) *along with a unique representation of the current time step* will become the new input to C. Before this new input can be processed, C (whose last input may have occurred many time steps earlier) is trained to predict this higher-level input from its current internal state and its last input (employing a conventional recurrent net algorithm). After this, C performs an activation update which contributes to a higher level internal representation of the input history. Note that according to the principle of history compression C is fed with an *unambiguous reduced description of the input history.* The information deducable by means of A's predictions can be considered as *redundant.* (The beginning of an episode usually is not predictable, therefore it has to be fed to the chunking level, too.)

Since C's 'credit assignment paths' will often be short compared to those of A, C will often be able to develop useful internal representations of previous unexpected input events. Due to the final term of its error function, A will be forced to reproduce these internal representations, *by predicting C's state.* Therefore A will be able to create useful internal representations by itself in an *early* stage of processing a given sequence; it will often receive meaningful error signals long before errors of the first or second kind occur. These internal representations in turn must carry the discriminating information for enabling A to improve its low-level predictions. Therefore the chunker will receive fewer and fewer inputs, since more and more inputs become predictable by the automatizer. This is the *collapsing operation.* Ideally, the chunker will become obsolete after some time.

It must be emphasized that unlike with the incremental creation of a multi-level predictor hierarchy described in section 3 there is no formal proof that the 2-net *on-line* version is free of instabilities. For instance, one can imagine situations where A unlearns previously learned predictions because of the third term of its error function. Relative weighting of the different terms in A's error function represents an

[4] In contrast, the reduced descriptions referred to by (Mozer, 1990) are not unambiguous.

vector	description (referring to time t)	dimension
$x(t)$	'normal' environmental input	n_I
$d(t)$	teacher-defined target	n_D
$i_A(t) = x(t) \circ d(t)$	A's input	$n_I + n_D$
$h_A(t)$	A's hidden activations	n_{H_A}
$d_A(t)$	A's prediction of $d(t)$	n_D
$p_A(t)$	A's prediction of $x(t)$	n_I
$time(t)$	unique representation of t	n_{time}
$h_C(t)$	C's hidden activations	n_{H_C}
$d_C(t)$	C's prediction of C's next target input	n_D
$p_C(t)$	C's prediction of C's next 'normal' input	n_I
$s_C(t)$	C's prediction of C's next 'time' input	n_{time}
$o_C(t)$	$d_C(t) \circ p_C(t) \circ s_C(t)$	$n_{O_C} = n_D + n_I + n_{time}$
$q_A(t)$	A's prediction of $h_C(t) \circ o_C(t)$	$n_{H_C} + n_{O_C}$
$o_A(t)$	$d_A(t) \circ p_A(t) \circ q_A(t)$	$n_{O_A} = n_D + n_I + n_{H_C} + n_{O_C}$

Table 1: *Definitions of symbols representing time-dependent activation vectors. 'o' is the concatenation operator. $h_A(t)$ and $o_A(t)$ are based on previous inputs and are computed without knowledge about $d(t)$ and $x(t)$.*

ad-hoc remedy for this potential problem. In the experiments (presented in section 5) relative weighting was not necessary.

4.2 DETAILS OF THE 2-NET CHUNKING ARCHITECTURE

The system described below is the *on-line* version of a representative of a number of variations of the basic principle described in 4.1. See (Schmidhuber, 1991c) for various modifications.

Table 1 gives an overview of various time-dependent activation vectors relevant for the description of the algorithm. *Additional notation*: 'o' is the concatenation operator; $\delta_d(t) = 1$ if the teacher provides a target vector $d(t)$ at time t and $\delta_d(t) = 0$ otherwise. If $\delta_d(t) = 0$ then $d(t)$ takes on some default value, e.g. the zero vector.

——————INSERT TABLE 1 HERE——————-

A has $n_I + n_D$ input units, n_{H_A} hidden units, and n_{O_A} output units (see table 1). With pure prediction tasks $n_D = 0$. C has n_{H_C} hidden units, and n_{O_C} output units. All of A's input and hidden units have directed connections to all of A's hidden and output units. All input units of A have directed connections to all hidden and output units of C. This is because A's input units *serve as input units for C at certain time steps*. There are additional n_{time} input units for C for providing unique representations of the current time step. These additional input units also have directed connections to all hidden and output units of C. All hidden units of C have directed connections to all hidden and output units of C.

A will try to make $d_A(t)$ equal to $d(t)$ if $\delta_d(t) = 1$, and it will try to make $p_A(t)$ equal to $x(t)$, thus trying to predict $x(t)$. Here again the target prediction problem is defined as a special case of an input prediction problem. C will try to make $d_C(t)$ equal to the externally provided teaching vector $d(t)$ if $\delta_d(t) = 1$ and if A failed to emit $d(t)$. Furthermore, it will always try to make $p_C(t) \circ s_C(t)$ equal to the next non-teaching input to be processed by C. This input may be many time steps ahead. Finally, and most importantly, A will try to make $q_A(t)$ equal to $h_C(t) \circ o_C(t)$, thus trying to predict the state of C. The activations of C's output units are considered as part of its state.

Both C and A simultaneously are trained by a conventional algorithm for recurrent networks in an on-line fashion. Both the IID-Algorithm and BPTT are appropriate. In particular, computationally inexpensive variants of BPTT (Williams and Peng, 1990) are interesting: There are tasks with hierarchical temporal structure where only a few iterations of 'back-propagation back into time' per time step are in principle sufficient to bridge *arbitrary* time lags (see section 5).

I now describe the (quite familiar) procedure for updating activations in a net.

Repeat for a constant number of iterations (typically one or two):

1. For each non-input unit j of N compute $\hat{a}_j = f_j(\sum_i a_i w_{ij})$, where a_j is the current activation of unit j, f_j is a semilinear differentiable function and w_{ij} is the weight on the directed connection from unit i to unit j.
2. For all non-input units j: Set a_j equal to $\hat{a}_j$.

I now specify the input-output behavior of the chunker and the automatizer as well as the details of error injection:

INITIALIZATION: All weights are initialized randomly. In the beginning, at time step 0, make $h_C(0)$ and $h_A(0)$ equal to zero, and make $i_A(0)$ equal $d(0) \circ x(0)$. Represent time step 0 in time(0). Update C to obtain $h_C(1)$ and $o_C(1)$.

FOR ALL TIMES $t > 0$ UNTIL INTERRUPTION DO:
1. Update A to obtain $h_A(t)$ and $o_A(t)$. A's error $e_A(t)$ is defined as

$$2e_A(t) = (p_A(t) \circ q_A(t) - x(t) \circ h_C(t) \circ o_C(t))^T (p_A(t) \circ q_A(t) - x(t) \circ h_C(t) \circ o_C(t)) + \delta_d(t)(d_A(t) - d(t))^T (d_A(t) - d(t)).$$

Use a gradient descent algorithm for dynamic recurrent nets to change each weight w_{ij} of A in proportion to (the approximation of) $-\frac{\partial e_A(t)}{\partial w_{ij}}$. Set $i_A(t)$ to $d(t) \circ x(t)$. Uniquely represent t in time(t).
2. If A's low-level error

$$2e_P(t) = (p_A(t) - x(t))^T (p_A(t) - x(t)) + \delta_d(t)(d_A(t) - d(t))^T (d_A(t) - d(t))$$

is less or equal to a small constant $\beta \geq 0$, then set $h_C(t+1) = h_C(t)$, $o_C(t+1) = o_C(t)$. Else define C's prediction error $e_C(t)$ as

$$2e_C(t) = (p_C(t) - x(t))^T (p_C(t) - x(t)) + \delta_d(t)(d_C(t) - d(t))^T (d_C(t) - d(t)) + (s_C(t) - time(t))^T (s_C(t) - time(t)),$$

use a gradient descent algorithm for dynamic recurrent nets to change each weight w_{ij} of C in proportion to (the approximation of) $-\frac{\partial e_C(t)}{\partial w_{ij}}$, and update C to obtain $h_C(t+1)$ and $o_C(t+1)$.

5 AN EXPERIMENT

Josef Hochreiter (a student at TUM) tested a chunking system against a conventional recurrent net algorithm. See (Hochreiter, 1991) and (Schmidhuber, 1991c) for details. A prediction task with a 20-step time lag was constructed. There were 22 possible input symbols $a, x, b_1, b_2, \ldots, b_{20}$. The learning systems observed one input symbol at a time. There were only two possible input sequences: $ab_1 \ldots b_{20}$ and $xb_1 \ldots b_{20}$. These were presented to the learning systems in random order. At a given time step, one goal was to predict the next input (note that in general it was not possible to predict the first symbol of each sequence due to the random occurrence of x and a). The second (and more difficult) goal was to make the activation of a particular output unit (the 'target unit') equal to 1 whenever the last 21 processed input symbols were $a, b_1, \ldots, b_{20}$ and to make this activation 0 whenever the last 21 processed input symbols were $x, b_1, \ldots, b_{20}$. No episode boundaries were used: Input sequences were fed to the learning systems without providing information about their beginnings and their ends. Therefore there was a continuous stream of input events.

With the conventional algorithm, with various learning rates, and with more than 1,000,000 training

sequences *it was not possible to obtain a significant performance improvement concerning the target unit.* A similar task involving time lags of as few as 5 steps required many hundreds of thousands of training sequences.

But, a chunking system was able to solve the 20-step task rather quickly, using an efficient approximation of the BPTT-method where error was propagated a maximum of 3 steps into the past (although there was a 20 step time lag!). No unique representations of time steps were necessary for this task. 13 out of 17 test runs required fewer than 5000 training sequences. The remaining test runs required fewer than 35000 training sequences.

Typically, A quickly learned to predict the 'easy' symbols $b_2, \ldots, b_{20}$. This led to a greatly reduced input sequence for C which now did not have many problems in learning to predict the target values at the end of the sequences. After a while A was able to mimic C's internal representations, which in turn allowed it to learn correct target predictions by itself. A's final weight matrix often looked like one one would hope to get from the conventional algorithm: There were hidden units which learned to bridge the 20-step time lags by means of strong self-connections. The chunking system needed less computation per time step than the conventional method. Still it required many fewer training sequences.

6 CONCLUDING REMARKS

It seems that people tend to memorize and focus on atypical or unexpected events and that they often try to explain new atypical events in terms of previous atypical events. In the light of the principle of history compression this makes a lot of sense.

Once events become expected, they tend to become 'subconscious'. There is an obvious analogy to the chunking algorithm: The chunker's attention is removed from events that become expected; they become 'subconscious' (automatized) and give rise to even higher-level 'abstractions' of the chunker's 'consciousness'.

The chunking systems described in (Schmidhuber, 1991a), (Schmidhuber, 1991c) and the current paper try to detect temporal regularities and learn to use them for identifying *relevant* points in time. A general criticism of more conventional algorithms can be formulated as follows: These algorithms do not try to selectively focus on *relevant* inputs, they waste efficiency and resources by focussing on *every* input.

Speech is a good example of a domain involving multi-level temporal structure. Ongoing research will explore the application of chunking systems to speech recognition.

The principle of history compression is not limited to neural networks. Any adaptive sequence processing device could make use of it.

7 ACKNOWLEDGEMENTS

Thanks to Josef Hochreiter for conducting the experiments. Thanks to Mike Mozer for useful comments on an earlier draft of this paper.

References

Hochreiter, J. (1991). Diploma thesis. Institut für Informatik, Technische Universität München.

Miyata, Y. (1988). An unsupervised PDP learning model for action planning. In *Proc. of the Tenth Annual Conference of the Cognitive Science Society, Hillsdale, NJ*, pages 223–229. Erlbaum.

Mozer, M. C. (1990). Connectionist music composition based on melodic, stylistic, and psychophysical constraints. Technical Report CU-CS-495-90, University of Colorado at Boulder.

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.

Rohwer, R. (1989). The 'moving targets' training method. In Kindermann, J. and Linden, A., editors, *Proceedings of 'Distributed Adaptive Neural Information Processing', St.Augustin, 24.-25.5,*. Oldenbourg.

Schmidhuber, J. H. (1991a). Adaptive decomposition of time. In Kohonen, T., Mäkisara, K., Simula, O., and Kangas, J., editors, *Artificial Neural Networks*, pages 909–914. Elsevier Science Publishers B.V., North-Holland.

Schmidhuber, J. H. (1991b). Learning to control fast-weight memories: An alternative to recurrent nets. Technical Report FKI-147-91, Institut für Informatik, Technische Universität München.

Schmidhuber, J. H. (1991c). Neural sequence chunkers. Technical Report FKI-148-91, Institut für Informatik, Technische Universität München.

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

Williams, R. J. and Peng, J. (1990). An efficient gradient-based algorithm for on-line training of recurrent network trajectories. *Neural Computation*, 4:491–501.

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