Recursive PCA and the Structure of Time Series.

Thomas Voegtlin
Institute for Theoretical Biology
Humboldt University
Berlin, Germany
E-mail: t.voegtlin@biologie.hu-berlin.de

***Abstract*— A recurrent linear network can be trained with Oja's constrained Hebbian learning rule. As a result, the network learns to represent the temporal context associated to its input sequence. The operation performed by the network is a generalization of Principal Components Analysis (PCA) to time-series, called Recursive PCA. During learning, the weights of the network are adapted to the temporal statistics of its input, in a way that maximizes the information retained by the network. Sequences stored in the network may be retrieved in the reverse order of presentation, thus providing a straight-forward implementation of a logical stack.**

I. Introduction

Time and its representations play a crucial role in most high-level cognitive tasks, such as planning, language capabilities, or the representation of complex objects. Among the most plausible and interesting neural representations of time, are representations based on recurrent networks, where the activities of neurons having recurrent connections represent the temporal context associated with the input. In a recurrent network, the representation of time is implicit. The most influential example illustrating this is the Simple Recurrent Network by Elman [4]. In this case, the values of the recurrent synapses are learned using the supervised back-propagation algorithm. However, it has been demonstrated that gradient-based methods fail to learn complex temporal dependencies in the input [2].

In this paper, we propose an unsupervised approach to training recurrent networks. Instead of training a network to predict future values of a series, we train it to represent the temporal context associated to its input time series. We demonstrate that this method effectively leads to learning temporal dependencies in the input, and that the resulting representation of context are linearly separable.

This paper is organized as follows. In the next section, we explain why optimizing a representation of context can be a way to extract the temporal structure of the associated process. Then we present the Recursive PCA algorithm, and its neural implementation. Finally, we present simulation results obtained with binary time series.

II. Representing temporal context

Consider the problem of representing the temporal context associated with a time-varying input, $\mathbf{x}(t)$, where $\mathbf{x}$ denotes a zero-mean bounded input vector, and t is the time index. Formally, the context at time t is the set of previous vectors $(\mathbf{x}(t-k))_{k\geq 0}$. Since a representation must be finite, only partial knowledge of the infinite sequence of past events is possible in most cases. In addition, the number of possibilities exponentially grows with the number of considered elements. In other words, representations of temporal context face a problem of capacity.

In order to address this issue, it is possible to exploit the temporal statistics of the input. Statistical dependencies introduce redundancy in the input, which should not be included in a compressed representation. For example, if a given input vector $\mathbf{x}_1$ is always preceded by another vector $\mathbf{x}_2$, then $\mathbf{x}(t) = \mathbf{x}_1$ implies that $\mathbf{x}(t-1) = \mathbf{x}_2$. At time t it is not necessary to code both $\mathbf{x}(t-1)$ and $\mathbf{x}(t)$, because any of these vectors can be deduced from the other one.

This generalizes to first-order statistical correlations. For example, if vector $\mathbf{x}_1$ is often preceded by vector $\mathbf{x}_2$, then $P[\mathbf{x}(t-1) = \mathbf{x}_2] \neq P[\mathbf{x}(t-1) = \mathbf{x}_2 | \mathbf{x}(t) = \mathbf{x}_1]$. In that case, depending on the amount of correlation, it might be interesting to use a compressed representation for both events, rather than two independent representations. Such a compressed representation is typically obtained by by using Principal Components Analysis.

Principal Components Analysis (PCA) consists in finding the orthogonal directions of highest variance in the distribution of a random vector. An interesting property of PCA is that it generates an optimal linear encoding of its input vector [1]. PCA also roots several of simple and efficient unsupervised learning methods for neural networks [7], [8], [11]. These methods can adapt their representation to the possibly non-stationnary statistics of the input.

PCA is based on variance; it projects an *input space* onto a *feature space*, where the ordered basis vectors are the orthogonal directions of maximal variance of the input. An appropriate dimension reduction in the feature space reduces the input vector to a set of effective features, which correspond to its projections in the directions of highest variance. This set of effective features is useful because it minimizes the mean-squared error :

$$E = < ||\mathbf{x}(t) - \bar{\mathbf{x}}(t)||^2 > \quad (1)$$

between the input at time t, $\mathbf{x}(t)$ and its projection $\bar{\mathbf{x}}(t)$, where $||\ ||$ denotes the Euclidean norm, and $<\ >$ denotes the

statistical expectancy. In other words, PCA finds a compressed representation, adapted to the statistical dependencies between the components of the input vector.

Here we want to optimize a representation of temporal context. Our hypothesis is that by doing so we will also extract the temporal statistics of the input. In order to capture temporal dependencies, it is possible to perform PCA on both vectors $\mathbf{x}(t)$ and $\mathbf{x}(t-1)$ together (i.e. on a concatenation of both vectors, which implements a time window of size 2 events). By increasing the size of the time window, longer dependencies can be taken into account. However, a solution of this type would correspond to a spatialization of time, which has several major limitations [4]. For example, in such a representation the size of the time window has to be predefined.

A more robust solution is to use a recurrent representation of context, i.e. a representation devised by a recurrent neural network [4]. In this case, the representation of time is implicit. In order to learn the values of recurrent synaptic weights, we propose to use PCA, because of the properties mentioned above.

III. Recursive PCA

Let $\mathbf{y}(t)$ denote a recurrent representation of the temporal context at time t. At time t, $\mathbf{y}(t)$ is computed from vectors $\mathbf{x}(t)$ and $\mathbf{y}(t-1)$. Our goal is to use PCA for this computation. Let n and m denote the respective dimensions of vectors $\mathbf{x}(t)$ and $\mathbf{y}(t)$. Note that m might be higher than n here, which is not the case in the traditional PCA framework,

In order to derive a representation $\mathbf{y}(t)$ of the context at time t, we build an $(n+m)$-dimensional vector, $\mathbf{z}(t)$, by concatenating vectors $\mathbf{x}(t)$ and $\lambda\mathbf{y}(t-1)$, where $0<\lambda<1$ is a gain :

$$\mathbf{z}(t) = \begin{pmatrix} \mathbf{x}(t) \\ \lambda\mathbf{y}(t-1) \end{pmatrix} \tag{2}$$

The idea here is to derive $\mathbf{y}(t)$ from $\mathbf{z}(t)$. This makes sense because $\mathbf{z}(t)$ supposedly contains all the information about the context at time t. Let the components of $\mathbf{y}(t)$ be the m principal components of $\mathbf{z}(t)$:

$$\mathbf{y}(t) = \mathbf{W}\mathbf{z}(t) \tag{3}$$

where $\mathbf{W}$ is an orthogonal m by $(n+m)$ matrix, whose m lines contain vectors that form an orthonormal basis of the m-dimensional dominant eigenvector subspace of the correlation matrix of $\mathbf{z}(t)$. Recursive PCA consists in finding $\mathbf{W}$, $\mathbf{z}$ and $\mathbf{y}$ satisfying (2), (3) and the above condition on $\mathbf{W}$.

Note that this definition does not provide a practical way of finding $\mathbf{W}$, nor does it show that such a matrix does actually exists. Since the distribution of vector $\mathbf{z}$ depends on the weights, the existence of solutions cannot be guaranteed a priori, and is a difficult problem.

In what follows, we develop an iterative method to find a solution to this problem. This method can be implemented in a recurrent neural network. We do not provide a proof of convergence for this method, however. Since the distribution of $\mathbf{z}(t)$ depends on $\mathbf{y}(t)$, finding $\mathbf{W}$ iteratively is a "moving target" problem, and convergence of the procedure is not easy to establish.

Experimentally, however, we observe convergence of our learning procedure for all $\lambda < 1$. Values of the gain equal or higher than one lead to numerical instability. Note that convergence of an iterative method demonstrates the existence of solutions *a posteriori*.

IV. Recovering previous events

In order to reconstruct the input, it is possible to apply the transpose $\mathbf{W}^T$ of $\mathbf{W}$ to the output vector. This generates a reconstruction $\bar{\mathbf{z}}(t)$ of $\mathbf{z}(t)$:

$$\bar{\mathbf{z}}(t) = \begin{pmatrix} \bar{\mathbf{x}}(t) \\ \lambda\bar{\mathbf{y}}(t-1) \end{pmatrix} = \mathbf{W}^T\mathbf{y}(t) \tag{4}$$

In addition to $\bar{\mathbf{x}}(t)$, $\bar{\mathbf{z}}(t)$ contains the reconstruction $\bar{\mathbf{y}}(t-1)$ of $\mathbf{y}(t-1)$. This allows us to recursively reconstruct the context. Given an estimation $\bar{\mathbf{y}}(t-k)$ of $\mathbf{y}(t-k)$, $\mathbf{W}^T$ can be applied in order to reconstruct $\mathbf{y}(t-k-1)$, for all $k \geq 0$:

$$\bar{\mathbf{z}}(t-k) = \begin{pmatrix} \bar{\mathbf{x}}(t-k) \\ \lambda\bar{\mathbf{y}}(t-k-1) \end{pmatrix} = \mathbf{W}^T\bar{\mathbf{y}}(t-k) \tag{5}$$

With the successive applications of $\mathbf{W}^T$, previous events are retrieved in the reverse order of presentation. This implements a logical stack, where a "push" operation corresponds to reading a new input with matrix $\mathbf{W}$, and a "pop" corresponds to reconstructing the previous input with $\mathbf{W}^T$.

We may define mean-squared errors associated to the reconstructions of previous events. At each time step, it is possible to reconstruct a sequence of previous events. Thus, for each value of k, there is an expectancy of the corresponding squared reconstruction error. Let e_k denote the mean squared reconstruction error:

$$e_k = < ||\mathbf{x}(t-k) - \bar{\mathbf{x}}(t-k)||^2 > \tag{6}$$

This error reflects how well a past event that happened k steps in the past can be reconstructed from the current input.

V. Neural Implementation

Recursive PCA can be performed by a recurrent neural network with two layers, encoding the input and the context representation, and denoted by X and Y respectively. The activity $\mathbf{y}_i(t)$ of neuron i in layer Y results from the combination of feed-forward connections, denoted by $\mathbf{w}$, from X to Y and recurrent connections, denoted by $\mathbf{w}'$, from Y to Y :

$$\mathbf{y}_i(t) = \sum_{j=1}^{n} \mathbf{w}_{ij}\mathbf{x}_j(t) + \lambda \sum_{k=1}^{m} \mathbf{w}'_{ik}\mathbf{y}_k(t-1) \tag{7}$$

where $\mathbf{x}_j$ is the activity of neuron j in X, $\mathbf{w}_{ij}$ is the weight from j to i and $\mathbf{w}'_{ik}$ is the weight between units k and i in Y. Note that matrix $\mathbf{W}$ is entirely defined by the set of weights $(\mathbf{w})$ and $(\mathbf{w}')$. The gain λ controls the influence of the recurrent connections. The time difference between the feed-forward term and the recurrent term can be interpreted either as a time delay in the recurrent connections, or as an explicit "context" layer as in [4]. The operation of the network

is illustrated in Figure 1, for both the forward mode (push), and the backward mode (pop).

Both feed-forward and recurrent connections are updated every time step, using Oja's constrained Hebbian learning rule [8] applied to vectors $\mathbf{x}(t)$ and $\lambda\mathbf{y}(t-1)$ simultaneously :

$$\Delta\mathbf{w}_{ij} = \eta\mathbf{y}_i(t)\left(\mathbf{x}_j(t) - \sum_{k=1}^{m}\mathbf{w}_{kj}\mathbf{y}_k(t)\right) \quad (8)$$

$$\Delta\mathbf{w}'_{ij} = \eta'\mathbf{y}_i(t)\left(\lambda\mathbf{y}_j(t-1) - \sum_{k=1}^{m}\mathbf{w}'_{kj}\mathbf{y}_k(t)\right) \quad (9)$$

where η and η' are learning rates. From a biological perspective, the transpose of $\mathbf{W}$ used in the previous section can be implemented by additional synapses symmetric to $(\mathbf{w}_{ij})$ and $(\mathbf{w}'_{ij})$.

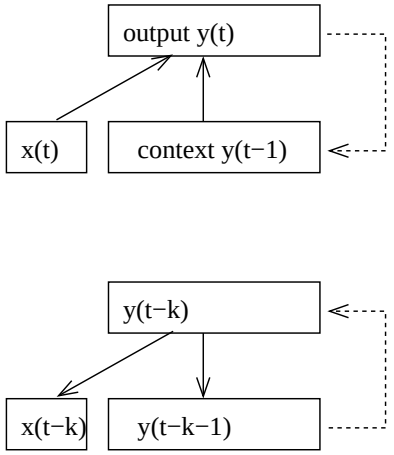

Fig. 1. Function of a simple recurrent network performing Recursive PCA. Connections are linear. Top: Forward mode. The input $x(t)$ is fed at the bottom. Both the input $x(t)$ and context $y(t-1)$ are combined together in order to form the output $y(t)$, using feed-forward connections (represented by full arrows). Dashed arrows indicate recurrent time-delayed one-to-one connections. Bottom: Backward (reconstruction) mode. The transpose of the connection matrix is used to reconstruct the input $x(t-k)$ and the context $y(t-k-1)$, from the current output $y(t-k)$. Recurrent one to one connections (dashed arrow) are used to iterate over many events.

As mentioned above, we do not present a proof of convergence for (8) and (9) here; the classical convergence results from [8] cannot be easily generalized here, due to the recurrent connections in our architecture. However, convergence is experimentally observed for λ below one. Experimental results are presented in the next section.

VI. Experiment: statistics of a simple automaton

In order to evaluate the performance of our learning method, we compared too conditions: In a control condition (A), the input time-series was i.i.d., and thus no dependencies could be learned. In a test condition (B), the input had temporal structure, and we expected the network to learn these time dependencies.

We used binary time-series in both conditions. The first series (A) was random i.i.d, and was was generated by picking $+1$ or -1 randomly with equal probabilities 0.5 (coin toss). The second series (B) was a simple Markov process, generated by a finite machine with two states, labeled $+1$ and -1. On each time step the probability of transition from one state to the other was equal to 0.3, while the probability of staying in a given state was 0.7. Note that in both the control condition (A) and the test condition (B), the input series have zero mean and variance equal to one.

The input was encoded using a single unit ($n = 1$), whose value was either $+1$ or -1. The representation of context and the output used $m = 20$ units.

In order to evaluate how well temporal context is encoded, the mean-squared reconstruction errors of previous events, e_k, were computed for $0 \leq k < 40$. Reconstructions of previous events were computed on each time step, by exploiting the stack property described in section IV. On each time step, 40 previous inputs were reconstructed, and the corresponding mean-squared error terms e_k were estimated. These reconstructions were performed using separate memory buffers, so that the reconstruction process did not interfere with the function of the network.

The mean-squared reconstruction errors corresponding to the 40 previous events is plotted on Figure 2, for several conditions. Curve A corresponds to the input time series with no temporal dependencies. Its shape was independent from the value of the gain λ. Curves $B(\lambda)$ correspond to the reconstruction error of sequences generated by the state machine. Several curves are plotted, corresponding to different values of the gain between 0.7 and 0.9.

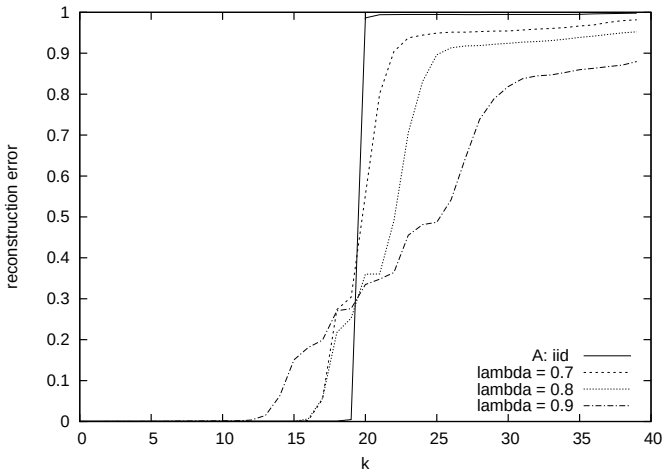

Fig. 2. Mean-squared reconstruction error, e_k, versus k, for binary random sequences. A: Reconstruction of a sequence with no temporal structure. B: Reconstruction of a sequence generated by the state machine, using different values for the gain λ: $0.7, 0.8$ and 0.9.

Curve A corresponds to the i.i.d. sequence. It has the shape of a step function. This indicates that if there are no temporal dependencies in the input, exactly 20 events ($0 \leq k < 20$) can be recovered from the activity of the 20 neurons, with a reconstruction error that is nearly equal to zero. For $k \geq 20$, the reconstruction error is equal to the variance of the input, which means that performance is not better than chance. In fact, we observe that the reconstructions performed by the

network for $k \geq 20$ are nearly equal to zero. Hence, the weight matrix that was learned by the network is almost a rotated permutation matrix.

Curves B(λ) correspond to the sequence generated by the state machine, for different values of the gain. Their general shape is a sigmoid, with a variable slope that depends on the gain. This means that if there are temporal dependencies, temporal context can be retrieved beyond the former "horizon" of 20 events. This implies that temporal dependencies are exploited by the network, in order to represent more than 20 events.

The counterpart of this ability to represent more previous events, is that reconstructions for $0 \leq k < 20$ are not perfect. The error increases with k, at a rate that depends on λ. Hence, the gain controls a trade-off between quality of the reconstructions and depth of the neural stack. Interestingly, when λ decreases, the shape of the sigmoid $B(\lambda)$ approaches the shape of curve A.

Despite this trade-off between quality and depth, we observe that the overall result is a net gain of information. For example, it is possible to measure the area between these curves and the chance level (equal to 1). This measure reflects the average number of events that can be correctly reconstructed. For $\lambda = 0.9$ this measure is equal to 28.3, while the reference value is 20 for the i.i.d. series.

This experiment demonstrates that temporal dependencies are exploited by the network, in order to gain information about past events. If there are no temporal dependencies, the representation is restricted to m events because no information can be gained from the statistics of the input. If there are dependencies, the network learns to exploit them in order to represent more than m events. The progressive loss of quality of the reconstructions reflects incorrect reconstructions that occur when low-probability transitions are encountered.

For the iid. input series, the number of correctly reconstructed events is exactly equal to the number of neurons used in the representation. This observation is general, and we observed that it still holds for networks of size up to 50 (data not shown). The only limiting factor seems to be the time required to train those networks.

VII. Conclusion

Recusive PCA generates a linear encoding of the temporal context associated to a time series. The purpose of the algorithm is not to predict the future values of an input, but to represent past values. However, by optimizing this representation, one imposes that the temporal correlations of the input time series are extracted. In that sense, Recursive PCA generalizes PCA to time-series. The resulting representations exploit the temporal structure of the input.

Recursive PCA is easily implemented in a linear recurrent neural network trained with Oja's unsupervised learning rule. In a previous work, we demonstrated that Sanger's generalized Hebbian algorithm can also be used in a recurrent network [13]. In these networks, PCA is performed recursively, using linear recurrent connections. We observed that the memory of the network increases linearly with its size. Moreover, the internal representation of context that is learned by the network exploits the temporal statistics of the input, in order to gain information about past events. As a result, the capacity of the network is higher when the input has temporal dependencies.

In a recurrent network, time has an implicit representation. Here, the orthogonality of the weights matrix makes it possible to reconstruct the sequence of past events. For this reason, the network presented here combines the advantages of implicit representations with potential explicit recovery.

Since the reconstruction process is a linear operation, being able to correctly reconstruct a sequence of inputs implies that its internal representation is linearly separable from the representations of other sequences. Linear separability ensures that the representation of temporal context is non-ambiguous, in which case it may be used as the input of a classifier. In our network, linear separability results from increasing the dimension of the representation. In that aspect, our approach is similar to so-called "liquid state" computing [6]. However, a key difference is that the network presented here, although it is linear, is able to learn and optimize its representation.

Previous events are recovered in the reverse order of presentation. This constitutes a neural implementation of a logical stack. This property has theoretical implications, concerning the type of operations that are possible in a neural network. Several authors have proposed neural implementations of logical stacks, to demonstrate that neural networks can perform the same type of operations as a Turing Machine [12], [9]. The stack property is sufficient for that, because a Turing Machine can be simulated by a Push-Down Automaton with two stacks [5].

In [12], the demonstration uses linear neurons. In order to push and pop arbitrary high number of elements in the stack, arbitrary precision of real numbers is invoked. However, physical systems are subject to noise and cannot afford arbitrary precision. In the Sequential RAAM [10], memory depends on the numbers of neurons rather than on precision, which makes it a more robust alternative. However, the number of binary patterns that can be efficiently encoded in Sequential RAAM tends to increase linearly with the size of the representation. In contrast, there is no such limitation in our model. For a binary i.i.d. input, we demonstrated that the resulting stack is of depth m, which means that it can store 2^m different binary patterns. Capacity is further improved if the input has statistical dependencies, because they are learned.

In practice, the stack property can be used for representing hierarchical structures [10], or for language processing, where nested grammatical structures typically require such a mechanism. An example of a combination of a neural network and an external stack is the Neural Network Pushdown Automaton [3], which learns to infer context-free grammars from examples. In that model, however, it is required that the operation of the stack is continuous, while the stack model presentd here is discrete.

ACKNOWLEDGMENT

This work was supported by the Alexander von Humboldt Foundation.

REFERENCES

[1] P. Baldi and K. Hornik, "Neural networks and principal component analysis: Learning from examples without local minima", *Neural Networks* **2**(1) pp. 53-58, 1988

[2] Bengio, Y., Simard, P., Frasconi, P. "Learning long-term dependencies with gradient descent is difficult." *IEEE Trans. on Neural Networks* **5**(2) pp. 157-166, 1994

[3] Das, S., Giles, C.L., Sun, G.Z. "Learning Context Free Grammars: Capabilities and Limitations of a Recurrent Neural Network with an External Stack Memory," *Proceedings of the 14 Annual Conference of the Cognitive Science Society*, Morgan Kaufmann, San Mateo, p. 79, 1992.

[4] Elman, J.L. "Finding structure in time." *Cognitive Science* **14** pp. 179-211, 1990

[5] Hopcroft, J.E. & Ullman, J.D. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, New York, 1979

[6] W. Maass and H. Markram "Temporal integration in recurrent microcircuits." In: *The Handbook of Brain Theory and Neural Networks* (2nd edition), MIT Press, pp. 1159-1163, 2003

[7] Oja, E. "A simplified neuron model as principal component analyzer." *Journal of Mathematical Biology* **15**(3) pp. 267-273, 1982

[8] Oja, E. "Neural networks, principal components and subspaces." *International Journal of Neural Systems* **1**(1) pp. 61-68, 1989

[9] Pollack, J. B. "The induction of dynamical recognizers. " *Machine Learning* vol. 7, pp. 227-252, 1991

[10] Pollack, J.B. "Recursive Distributed Representations." *Artificial Intelligence* **46**(1-2) pp. 77-105, 1990

[11] Sanger, T.D. *Optimal unsupervised learning in a single-layer linear feedforward neural network.* Neural Networks **2**(6) pp. 459-473. 1989

[12] Siegelman, H. T. & Sontag, E. D. "Turing-Computability with Neural Nets." *Applied Mathematics Letters*, **4**(6) (1991) pp. 77-80.

[13] Voegtlin, T. "Learning principal components in a contextual space." *Proceedings of the ESANN*, Bruges, 2000, pp. 359-364.