Unsupervised Context-based Learning of Multiple Temporal Sequences

Guilherme de A. Barreto Aluizio F. R. Araújo
Departamento de Engenharia Elétrica – Universidade de São Paulo
Av. Dr. Carlos Botelho, 1465
13560-250 São Carlos, SP, Brasil
{gbarreto, aluizioa}@sel.eesc.sc.usp.br

Abstract

A self-organizing neural network is proposed to handle multiple temporal sequences with states in common. The proposed network combines context-based competitive learning with time-delayed Hebbian learning to encode spatial features and temporal order of sequence items. A responsibility function to avoid catastrophic forgetting, and a redundancy mechanism to provide noise and fault-tolerance increase the reliability of the model. States shared by different sequences are encoded by a single neuron, whereas context information indicates the correct sequence to be recalled in the case of ambiguity. Simulations with trajectories of a PUMA 560 robot are performed to test the network accuracy, robustness to noise and tolerance to faults.

1. Introduction

Many engineering and cognitive tasks require ability to process spatio-temporal sequences. This sequential patterns usually takes the form of a finite, discrete set of items appearing successively in time. Such sequences carry information on the static nature of an individual item, and on the temporal order of their items [1].

A number of memory models for sequential patterns have been proposed in the context of verbal learning [2], but they have similar counterparts in the domain of neural models for robot control. Basically, three paradigms have been applied successfully: associative chaining [3], positional coding [4], and chunking [5]. In the associative chaining hypothesis, each item in a sequence is associated with its precedent items in either a unidirectional or bi-directional fashion. Thus, each item is cued by the preceding ones. In the positional coding hypothesis, each list item is associated with a positional code referring to its location in the sequence. Such an information is used during recall to reconstruct the sequence in the correct order. The chunking hypothesis refers to compressed, categorical, or unitized representations of sequence items which behave functionally as a single item. As mentioned, these paradigms have been used within the framework of artificial neural networks (ANN). For instance, recurrent networks [6], [7]; Hopfield-like associative memory [8], [9]; and unsupervised learning models [10], [11] follow the principle of associative chaining.

Regarding the application of unsupervised models to robot control, Althöfer & Bugmann [12] proposed a two-stage neural network model for planning, learning and retrieval of robot arm movements based on the associative chaining paradigm. The first stage implements a grid-like neural network for path planning, and the second stage learns the trajectories, generated by the first one, through RBF nodes sequentially connected. This model produces smooth movements and accurate final positions for trajectories with no repeated points. Bugmann et al. [13] extended the former model by combining associative chaining and positional coding. This model aims at learning the sequence of positions forming the trajectory of an autonomous wheelchair, and operates by producing the next position for the wheelchair. Positional information is used to disambiguate recall of repeated points. This network is able to handle multiple trajectories provided that a shared point does not occur in the same position. Denham & McCabe [14] uses the chunking paradigm to control an autonomous mobile robot that uses sequences of pairs (sensory stimuli, motor actions) learned over time, to build an internal model of the world in which the robot navigates. The network in Wang & Arbib [15] is suggested as the main controller. However, the proposed system demands additional work to be formalized and implemented in a real-world robot.

As the majority of the existing unsupervised models for sequence processing does not directly address the issue of multiple temporal sequence learning (see Wang and Yuwono [16] as an exception), the aim of this work is to develop an unsupervised neural network, based on chaining hypothesis, to learn and recall temporal patterns. The remaining of the paper is organized as follows. In Section 2, we present the neural algorithm. In Section 3,

0-7803-5529-6/99/$10.00 ©1999 IEEE

we evaluate the model through simulations. In Section 4, we conclude the paper and give possible directions for further work.

2. The Proposed Model

Two characteristics are essential for autonomous learning and reproduction of sequential patterns. For purpose of learning, a mechanism must be implemented to extract and store temporal relationship between patterns. For purpose of recall, the network dynamics must be defined to reproduce the previously observed sequence.

Sequences are presented to the network item-by-item. Then, the neural model should store each item individually as well as the temporal order of such input items. In order to use memory resources efficiently, each repetition of a state, called *shared state*, is stored in a single neuron. During recall, the network has to produce the successor of each input item. The network architecture is presented in Figure 1.

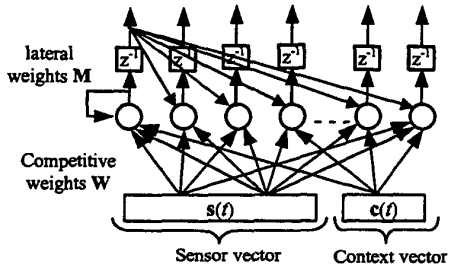

Figure 1. The topology of the proposed network. Only some connections are shown.

Each input pattern, $\mathbf{v}(t) \in \boldsymbol{R}^{p+q}$, is comprised of two parts. The *sensor vector*, $\mathbf{s}(t) \in \boldsymbol{R}^{p}$, corresponds to an observed sequence item at time t. In this work, it takes the values of the spatial position of the end-effector, joint angles and applied torques. The *context vector*, $\mathbf{c}(t) \in \boldsymbol{R}^{q}$, is used to handle ambiguities during recall.

Associated with the j-th output neuron there is a feedforward weight vector, $\mathbf{w}_j(t) = \{\mathbf{w}_j^s(t), \mathbf{w}_j^c(t)\}$, $j = 1, ..., n$, responsible for storing each one of the sequence items $\mathbf{v}(t)$. Such a vector is divided into a sensor, $\mathbf{w}_j^s(t)$, and a context, $\mathbf{w}_j^c(t)$, part that encodes $\mathbf{s}(t)$ and $\mathbf{c}(t)$ respectively. Also, there is a lateral weight vector, $\mathbf{m}_j(t)$, which should encode the temporal order of the sequence by connecting consecutive items. Finally, it is defined an activation state, $a_j(t) \in \boldsymbol{R}$, that identifies the neuron whose weight vector $\mathbf{w}_j^s(t)$ is the most similar to the current sensor vector $\mathbf{s}(t)$, and an output value, $y_j(t) \in \boldsymbol{R}$, to indicate the neuron whose weight vector contains the next item in the stored sequence. The learning processes are described next.

Context-based competitive learning: After the presentation of the input vector $\mathbf{v}(t)=\{\mathbf{s}(t), \mathbf{c}(t)\}$, the output neurons "compete" for encoding it. The competition process uses Euclidean distances as similarity measures. Hence, we define a *sensor distance*, $D_j^s(t) \in \boldsymbol{R}$, and a *context distance*, $D_j^c(t) \in \boldsymbol{R}$, for each output unit j:

$$D_j^s(t) = \left\| \mathbf{s}(t) - \mathbf{w}_j^s(t) \right\| \quad \text{and} \quad D_j^c(t) = \left\| \mathbf{c}(t) - \mathbf{w}_j^c(t) \right\| \tag{1}$$

where $\| \mathbf{x} \|^2 = x_1^2 + ... + x_{p+q}^2$. The distance $D_j^s(t)$ is used to find the winners of the current competition, while $D_j^c(t)$ is used to solve ambiguities during recall.

It is worth remembering that, to guarantee accurate recall of all states of a given sequence, the network should avoid the situation in which a neuron wins the competition more than once. This can be accomplished by introducing a function $R_j(t)$, called *responsibility function*, that indicates whether a neuron is already responsible for encoding a given sequence item. If $R_j(t) > 0$, neuron j is excluded from subsequent competitions for sequence components. If $R_j(t) = 0$, neuron j is allowed to compete.

In sequel, we define the following function:

$$f_j(t) = \begin{cases} D_j^s(t), & \text{if } D_j^s(t) < \text{DOS or } R_j(t) = 0 \\ R_j(t), & \text{otherwise} \end{cases} \tag{2}$$

where DOS is an acronym for degree of similarity. This constant specifies a criterion to allow neurons to be chosen winners even if they have already won before. This mechanism contemplates situations in which repeated or shared states occur, allowing them to be encoded by the same neuron.

Finally, the output neurons are ranked, as follows:

$$f_{\mu_1}(t) < f_{\mu_2}(t) < ... < f_{\mu_{n-1}}(t) < f_{\mu_n}(t) \tag{3}$$

where $\mu_i(t)$, $i = 1,..., n$, is the index of the i-th closest output neuron to $\mathbf{s}(t)$. We choose K neurons, $\boldsymbol{\mu}(t) = (\mu_1(t), \mu_2(t),...,\mu_K(t))$, $0 < K << n$, as winners of the current competition. They will represent the current input vector $\mathbf{v}(t)$. During learning, the redundancy degree K is often set

to a value greater than one in order to guarantee tolerance to faults and distortions in the input sequence caused by noise. During recall, it is *always* set to K=1.

The corresponding activation values decay linearly from a maximum value $a_{max} \in \boldsymbol{R}$, for $\mu_1(t)$, to a minimum $a_{min} \in \boldsymbol{R}$, for $\mu_K(t)$, according to the following equation:

$$a_{\mu_i}(t) = a_{\max} - \left(\frac{a_{\max} - a_{\min}}{\max(1, K-1)} \right)(i-1), \qquad i = 1, \ldots, K \quad (4)$$

where a_{max} and a_{min} are user-defined constants. The responsibility function $R_j(t)$ is updated every time a new activation pattern, $\mathbf{a}(t) = (a_1(t), \ldots, a_n(t))^T$, is computed:

$$R_j(t+1) = R_j(t) + \beta a_j(t) \qquad j = 1, \ldots, n \quad (5)$$

where $\beta >> 0$ is the exclusion constant.

The competitive weights are adjusted according to the following equation:

$$\mathbf{w}_j(t+1) = \mathbf{w}_j(t) + \eta a_j(t)[\mathbf{v}(t) - \mathbf{w}_j(t)] \quad (6)$$

where $0 < \eta \leq 1$ is the learning rate. Only neurons with activation different of zero are allowed to learn. If we adopt $\eta \cong 1$, an input vector item $\mathbf{v}(t)$ is learned very quickly, allowing an entire sequence to be learned in only one pass of its components.

Time-delayed Hebbian Learning: To encode transitions between consecutive states, we employ a lateral coupling structure. Such unidirectional connections are adjusted when the winners of each competition are determined, indicating the correct temporal order of appearance of the sequence items. The learning rule is [17]:

$$m_{jl}(t+1) = m_{jl}(t) + \alpha a_j(t) a_l(t-1) \qquad j, l = 1, \ldots, n \quad (7)$$

where $\alpha > 0$ is the lateral learning rate. Equation (7) updates the weights from winners at time t-1 to winners at time t. In other words, state transitions are learned by the lateral connections between consecutive winners. During recall, the resulting lateral coupling structure takes the network to respond with the neurons that encoded the successor of the current input vector.

Kopecz [11], Montague & Sejnowski [18], Girolami & Fyfe [19], and Wallis [20] have also used time-delayed Hebbian rules to model temporal aspects of sequences.

The network output values are calculated as follows:

$$y_j(t) = \left(1 - \frac{D_j^c(t)}{\sum_{l=1}^{n} D_l^c(t)} \right) \left(\sum_{l=1}^{n} m_{jl}(t) a_l(t) \right) \quad (8)$$

where the weight vector of the neuron with the largest value for $y_j(t)$ contains the next configuration for the robot arm. This weight vector is sent to the robot controllers.

The role of the first factor on the right hand side of Equation (8) is to solve ambiguities. If a shared state has been reached, there are various possibilities for the next state. This uncertainty is resolved by that factor, because the correct neuron is the one with the smallest context distance $D_j^c(t)$ which yields larger values for $y_j(t)$.

3. Simulations and Discussions

The proposed model is applied as part of a larger control system for trajectory tracking of a 6-DOF PUMA 560 robot. This task aims at converting a description of a desired motion into a trajectory defined as a time sequence of intermediate configurations between an origin and a destination [21]. The desired motion is that of an industrial robot arm consisting of joints driven by individual actuators. The robot is required to follow a prescribed path, so that its controllers must coordinate the movements of the robot individual joints to achieve the desired overall movement along the path. In this sense, our model acts as a trajectory planner, outputting a sequence of arm configurations that forms the input to the arm control system.

The trajectories in this work have one state in common, and they are grouped into 3 types according to the position of the shared states in the sequence. Trajectories sharing the same origin but with different destinations are called 1-m (one-to-many) trajectories. This group of trajectories, considered in the reverse temporal order, are called m-1 (many-to-one), since now they share the same destination. Trajectories that share intermediate states (other than the origin and destination ones) are called of type mm-wc (many-to-many with crossing). These trajectories have already been used to evaluate supervised neural models for robot control [22], [23].

The fundamental problem in processing trajectories with repeated or shared items is to determine which trajectory the arm should follow when it reaches such a bifurcation point. In this paper, trajectories were constructed with the toolbox robotics of Matlab [24], and to visualize the learned trajectories, a robot simulator was developed.

The network parameters and constants for all simulations are the following: $p = 15$, $q = 3$, $n = 250$, DOS = 10^{-4}, $a_{max} = 1$, $a_{min} = 0.98$, $\beta = 100$, $\eta = 1$, $\alpha = 0.8$. The weights and functions are initialized only in the beginning of training: $w_{ji}(0) = \text{rand}[0, 1]$, $m_{jr}(0) = 0$, and $R_j(0) = 0$, for all i, j and r. The activation and output values are initialized every time a new sequence item is observed: $a_j(0) = y_j(0) = 0$, for all j. The context $\mathbf{c}(t)$ is always set to the spatial position of the destination, (x_t, y_t, z_t), of the trajectory being processed and remain unchanged for each trajectory. For the present simulations, this information is sufficient to solve any ambiguity. The network is evaluated in terms of mean square error (MSE) values obtained from the comparison between the desired and the retrieved spatial positions of the arm.

For the first test, three 1-m trajectories (with 11 states) are presented to the network sequentially, one after the other. These trajectories share a point at spatial position (0.60,0.10,0.00). Recall is performed to verify if the sequences were encoded accurately and in the correct order. The results in Figure 2 illustrate a correct recall procedure. Both, learning and recall, are performed using noise-free inputs. Under these conditions, the MSE errors during recall are very small, of the order of 10^{-5}.

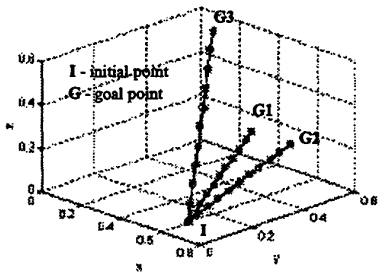

Figure 2. Desired 'o' and retrieved '.' spatial trajectories with same starting point (0.6, 0.1, 0.0).

Figure 3 allows the visualization in a simulated robot configuration space of one of the retrieved trajectories (I-G1) of type 1-m.

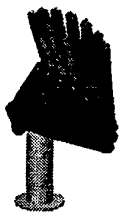

Figure 3. Visualization of a retrieved trajectory with 11 states.

In the next test, for a given origin and destination, we evaluate the effect of varying the number of intermediate states on the network performance under noisy conditions. For this, we fixed the redundancy degree K and used trajectories of type m-1 with 11, 21, 41, and 81 points respectively. A typical result for K=3 is shown in Figure 4. For the sake of clarity, only the results for trajectories with 11 and 81 points are presented.

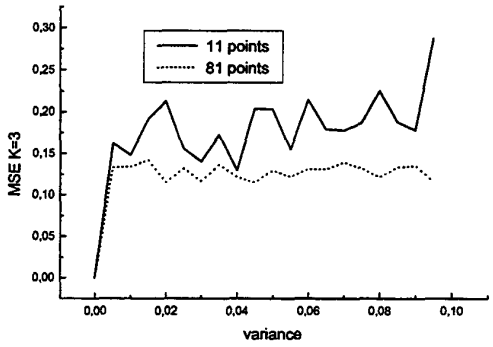

Figure 4. Effect of varying the number of states of a given trajectory, for a fixed K, on the network recall performance under the addition of Gaussian noise N(0,σ^2) to the input.

From Figure 4, one can conclude that, for a fixed K, a higher number of intermediate points reduces the MSE levels. The results for 21 and 41-point trajectories (not shown) also confirm this assertive.

The last simulation evaluates the effect of varying the redundancy degree K, for a fixed number of intermediate states, on the network performance under noisy conditions. For this, we used trajectories of type mm-wc with 81 points, and varied K from 1 to 5. A typical result is shown in Figure 5. For the sake of clarity, only the results for K=3 and 4 are presented.

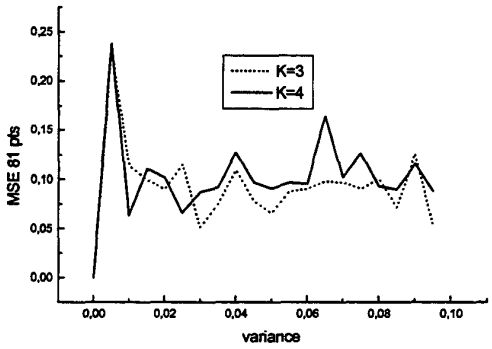

Figure 5. Effect of varying the redundancy degree K, for a fixed number of states, of a given trajectory, on the network recall performance under the addition of Gaussian noise N(0,σ^2) to the input.

From Figure 5, one can infer that there is an upper bound for values of K, since on average $K = 3$ gives better results for MSE that $K=4$. The results for $K = 5$ (not shown) are poorer than that for $K \leq 4$. This is an interesting result because memory requirements can be roughly estimated by analyzing the values of K for each stored sequence.

4. Conclusions

In the proposed model, multiple sequences are stored in a simple network in such a way that the patterns of activity for items representation encodes both the individual items that have occurred and the temporal order in which they have occurred.

Our model behave similarly to the Grossberg's outstar avalanche model [10], in the sense that our method treats both simple and complex sequences in the same way, and a complex sequence can be recalled as easily as a simple one. The basic difference is that outstar avalanche stores repeated or shared items as different copies, while our model stores them as a unique copy which can still correctly recall the sequence due to context information. As a result, our model yields an efficient use of memory space. Also, the proposed model is more reliable because, being redundant, it recalls stored sequences even if neurons or their connections are damaged. The problem of *ad hoc* wiring for temporal coupling, existing, for example, in the outstar avalanche, Bugmann & Althöfer and Bugmann et al. models has been avoided here. Compared with the model proposed by Kopecz [11], our model is substantially faster and can handle shared or repeated items. Further work has to be developed in order to compare the proposed model with other existing ones.

Acknowledgments

The authors thanks FAPESP for its financial support.

References

[1] Ray, S.R. & Kargupta, H. (1996). A temporal sequence processor based on the biological reaction-diffusion process. *Complex Systems*, Vol. 9, No. 4, 305-327.

[2] Kahana, M. J. & Jacobs, J. (1998). Inter-response times in serial recall: effects of intra-serial repetition. *Submitted to Journal of Exp. Psyc.: Learning, Memory, and Cognition.*

[3] Crowder, R. G. (1968). Intraserial repetition effects in immediate memory. *Journal of Verbal Learning and Verbal Behavior*, Vol. 7, 446-451.

[4] Young, R. K. (1968). Serial Learning. In: *Verbal Behavior and General Behavior Theory*, T. R. Dixon and D. L. Horton (Eds.), pp. 122-148, NJ: Prentice Hall.

[5] Miller, G. A. (1956). The magical number seven, plus or minus two: some limits on our capacity for processing information. *Psychological Review*, vol. 63, 81-97.

[6] Elman, J. L. (1990). Finding structure in time. *Cognitive Science*, vol. 14, 179-211.

[7] Jordan, M. I. (1986). Attractor dynamics and parallelism in a connectionist sequential machine, *Proc. of the 8th Annual Conf. of the Cognitive Sci. Society*, Amherst, MA, 531-546.

[8] Sompolinsky, H. & Kanter, I. (1986). Temporal association in asymmetric neural networks. *Phys. Rev. Lett.*, vol. 57, 2861-2864.

[9] Guyon, I., Personnaz, L., Nadal, J. & Dreyfus, G. (1988). Storage and retrieval of complex sequences in neural networks. *Physical Review A*, vol. 38, no. 12, 6365-6372.

[10] Grossberg, S. (1969). Some networks that can learn, remember, and reproduce any number of complicated space-time patterns, I. *Journal of Math. and Mech.*, vol. 19, 53-91.

[11] Kopecz, K. (1995). Unsupervised learning of sequences on maps with lateral connectivity. *Proc. of the International Conf. on Artificial Neural Networks*, vol. 2, 431-436.

[12] Althöfer, K. & Bugmann, G. (1995). Planning and learning goal-directed sequences of robot arm movements. In: *Fogelman-Soulié F. & Gallinari (Eds.), Proceedings of the International Conference on Artificial Neural Networks (ICANN'95)*, Paris, France, vol. 1, 449-454.

[13] Bugmann, G., Koay, K. L., Barlow, N., Phillips, M. & Rodney, D. (1998). Stable encoding of robot trajectories using normalized radial basis functions: Application to an autonomous wheelchair. *Proc. of the 29th International Symp. on Robotics (ISR'98)*,Birmingham, UK, 232-235.

[14] Denham, M. J. & McCabe, S. L. (1995). Robot control using temporal sequence learning. *Proceedings of the World Congress on Neural Networks*, vol. II, 346-348.

[15] Wang, D. L. & Arbib, M. A. (1993). Timing and chunking in processing temporal order. *IEEE Transactions on Systems, Man, Cybernetics*, vol. 23, 993-1009.

[16] Wang, D. L. & Yuwono, B. (1996). Incremental learning of complex temporal patterns. *IEEE Transactions on Neural Networks*, vol. 7, no. 6, 1465-1481.

[17] Barreto, G. A. & Araújo, A. F. R. (1999). Fast learning of robot trajectories via unsupervised neural networks. *Proc. of the 14th IFAC World Congress*, Beijing, China.

[18] Montague, R. & Sejnowski, T. J. (1994). The predictive brain: temporal coincidence and temporal order in synaptic learning mechanisms. *Learning & Memory*, no. 1, 1-33.

[19] Girolami, M. & Fyfe, C. (1996). A temporal model of linear anti-Hebbian learning. *Neural Proc. Lett.*, no. 4, 139-148.

[20] Wallis, G. (1998). Spatio-temporal influences at the neural level of object recognition. *Network: Computation in Neural Systems*, vol. 9, 265-278.

[21] Thompson, S. E. & Patel, R. V. (1987). Formulation of joint trajectories for industrial robots using B-splines. *IEEE Trans. on Industrial Electronics*, vol. IE-34, no. 2, 192-200.

[22] Araújo, A. F. R. & Vieira, M. (1998). Associative memory used for trajectory generation and inverse kinematics problem. *Proc. of the International Joint Conference on Neural Networks*, Anchorage, AK, USA, 2057-2052.

[23] Araújo, A. F. R. & D'Arbo, H. (1998). Partially recurrent neural network to perform trajectory planning, inverse kinematics, inverse dynamics. *Proc. of the IEEE Int. Conf. on System, Man, and Cybernetics*, San Diego, USA, 1784-1789.

[24] Corke, P. I. (1996). A robotics toolbox for MATLAB. *IEEE Robotics and Automation Magazine*, vol. 3, no. 1, 24-32.