On using discretized Cohen-Grossberg node dynamics for model-free actor-critic neural learning in non-Markovian domains [*]

Eiji Mizutani
Dept. of Computer Science
National Tsing Hua University
Hsinchu 300 Taiwan R.O.C.
eiji@wayne.cs.nthu.edu.tw

Stuart E. Dreyfus
Department of IEOR
University of California at Berkeley
Berkeley, CA 94720, USA
dreyfus@ieor.berkeley.edu

Abstract

We describe how multi-stage non-Markovian decision problems can be solved using actor-critic reinforcement learning by assuming that a discrete version of Cohen-Grossberg node dynamics describes the node-activation computations of a neural network (NN). Our NN (i.e., agent) is capable of rendering the process Markovian implicitly and automatically in a **totally model-free** fashion without learning by how much the state space must be augmented so that the Markov property holds. This serves as an alternative to using Elman or Jordan-type recurrent neural networks, whose context units function as a history memory in order to develop sensitivity to non-Markovian dependencies. We shall demonstrate our concept using a small-scale non-Markovian **deterministic** path problem, in which our actor-critic NN finds an optimal sequence of actions (but learns neither transitional dynamics nor associated rewards), although it needs many iterations due to the nature of neural model-free learning. This is, in spirit, a neuro-dynamic programming approach.

1 Introduction

Practical multi-stage decision-making problems may involve hidden **non-Markovian** characteristics that are correlated with certain past events; both reinforcement signals and states may depend *arbitrarily* on the past history of the agent's outputs (e.g., decisions) through interaction with the dynamic world (i.e., environment). We assume in what follows that only the reinforcement signal is non-Markovian. In general, the approaches to make the agent manage non-Markovian situations can be roughly categorized into the next two types:

(1) Model-based (or model-building) approach,
wherein the agent attempts to learn **explicitly** what prior events matter;

(2) Model-free approach,
wherein the agent retains some **internal state** (or **memory**) over time that is automatically **sensitive** to non-Markovian dependencies by trial-and-error interaction with the world without attempting to learn a world model.

The classical dynamic programming (DP) algorithm is a model-based approach because it requires an *explicit* state description (see Section 3.1) to allow a DP solution based on the *principle of optimality* [1]. In control engineering, the model-building approach is known as *system identification*, whereby the agent is assumed to be able to model the mapping from *actions* to *reinforcement signals* by observing states, actions, and reinforcement signals. *Utile Suffix Memory* [2] is a typical model-building algorithm that constructs a tree representation with associated Q-values.

In a model-free approach, a typical realization of *internal memory* is to use *recurrent neural networks* (see [3, 4], Chap. 7 in [5], and pages 291–292 in [6]) such as an Elman network or a Jordan network, whose context units *implicitly* and *automatically* encode certain aspects of the past as far back as it goes. Here, two distinct dynamics are entwined:

- **External environmental dynamics**: task-oriented (or problem-specific) time-dependent dynamics that governs the past history as well as the future evolution (e.g., drives the agent to a next stage upon interaction);

- **Internal neural dynamics**: agent's internal neuron-level dynamics, related to neuron-output propagations through networks (e.g., layer-by-layer forward and backward passes in a simple MLP-agent); in general recurrent neural networks (of animals and humans), such dynamics are complicated.

Through interaction with the environment, the agent's internal dynamics will be harmonized in a certain sense with external dynamics, being sensitive to the past history governed by laws of external dynamics (but still unknown to the agent itself because of its model-free nature). In what follows, we describe a model-free *actor-critic reinforcement learning* agent with a partially recurrent network, whose internal neuron dynamics are

[*] This paper appeared in Proc. of the 5th IEEE Int'l Symposium on Computational Intelligence in Robotics and Automation (CIRA 2003), Vol.1., pp. 1–6.

regulated by a discrete version of *Cohen-Grossberg dynamics*. The resulting NN has the ability to solve a non-Markov problem using different internal dynamics from those of Elman and Jordan-type NNs.

2 Reinforcement Learning with Cohen-Grossberg Dynamics

A variety of dynamical rules could be described for recurrent networks, such as

$$\tau_j \frac{da_j}{dt} = -a_j + f(net_j) = -a_j + f\left(\sum_i \theta_{i,j} a_i\right) \quad (1)$$

where a_j denotes neuron dynamics or activation level (i.e., state) of a particular neuron j; τ_j some time constants; and $\theta_{i,j}$ the weight parameter from node i to node j. This formula is what we call the "Cohen-Grossberg" dynamic formula [7] (see Eq.(3.31), page 54 in [5]), but its variants have been discussed in many different contexts (for example, see Eq.(1) in [8] as well as refs. [9, 3]).

2.1 A Discretized Cohen-Grossberg Dynamical Rule

For our discrete-action problems (see Section 3), we discretize Eq. (1) for reinforcement learning as:

$$\tau_j \left[\frac{a_j(t+\Delta t) - a_j(t)}{\Delta t}\right] = -a_j(t) + f\left(net_j(t+\Delta t)\right),$$

where the last term is special to reinforcement learning, and letting $\Delta t = 1$ and replacing t by $t-1$ (and then calling t "stage" rather than "time") yields (using $\beta_j \equiv 1 - \frac{1}{\tau_j}$):

$$a_j(t) = \beta_j a_j(t-1) + (1-\beta_j) f\left(net_j(t)\right). \quad (2)$$

This is our discretized Cohen-Grossberg dynamical rule; here, β_j is a constant weight for the linear combination of $a_j(t-1)$ and $f\left(net_j(t)\right)$. In everything that follows we shall apply Eq. (2) to "standard" N-layered multilayer perceptrons (MLPs). The resultant NNs can be viewed as an NN with partial recurrence, wherein each neuron (say node i) is assumed to have a *self-loop* connection to itself that has a fixed value (β_j). Of course, one might use another type of self-loop (back to the net input similar to Elman and Jordan-type recurrence) shown below:

$$a_j(t) = f\Big(\beta_j a_j(t-1) + (1-\beta_j) net_j(t)\Big),$$

but the main virtue of the self-loop in Eq. (2) is that it is similar to what a brain presumably does. In any event, such self-looping modification still allows us to use the standard MLP-backpropagation because all β_j are fixed, and $a_j^s(t-1)$, the self-looped activation from the forward pass done at previous stage $t-1$, is just a constant when the node derivatives are evaluated at stage t. Notably, the added constants make the *node outputs sensitive* to past history, whereas Elman-type recurrence renders the *net inputs sensitive* to it. The resultant dynamics are also different from dynamics by *backpropagation through time* [10].

2.2 An Actor-Critic reinforcement learning algorithm

Actor-Critic (AC) reinforcement learning is a class of simulation-based approximate policy iteration algorithms. While generating a number of simulated system trajectories with associated accrued rewards, *Critic* approximates maximum reward-to-go values (i.e., **policy evaluation**), and *Actor* generates a current policy based on the current estimated values (**policy improvement**). Here, the policy is continuously updated by Actor before the values (constantly updated by Critic) converge; hence, called *optimistic* policy iteration [11]. For recent analysis on AC-learning, refer to [12, 13].

When we consider the NN-agent with the new dynamical rule (2) that attempts to solve a multi-stage non-Markov decision-making problem, the agent's decision $\mathbf{u}(t)$ at stage t becomes twofold below in association with both the *external task dynamics* and *internal neural dynamics* (explained in Section 1):

(1) external decision z (scalar, in our example described later) for solving the posed non-Markov problem (i.e., external problem);

(2) internal decision $\boldsymbol{\theta}$ for seeking the optimal *parameters* (denoted by n-dimensional vector $\boldsymbol{\theta}$) of the MLP-agent itself (i.e., internal problem).

For our convenience, the *state* and *stage* in the external task/problem are denoted by y and t (i.e., external time t), respectively, while the *state* and *stage* in the internal problem by a and s (i.e., layer s), respectively. Since these two problems have their own dynamics that are discretized with different clocks (i.e., different time ticks), we specifically call t the *task-stage* and s the *layer-stage* without using the term "time-stage" common to both cases. Hence, the agent's twofold decision $\mathbf{u}(t)$ can be expressed as $\mathbf{u}(t)$=[$z(t)$, $\boldsymbol{\theta}(t)$] at task-stage t, but internal decision $\boldsymbol{\theta}(t)$ consists of a sequence of actions $\boldsymbol{\theta}^s$ at each layer-stage s (s=$1, \cdots, N-1$). Our notations are summarized in the next table:

	External problem	Internal problem
State	y	a
Stage	t (task)	s (layer)
Action	z	θ

Accordingly, the value function approximated by Critic can be written as $V\left(y(t), \mathbf{a}(t-1)\right)$ and the randomized policy function approximated by Actor can be written as $A\left(y(t), \mathbf{a}(t-1)\right)$, wherein the agent's state at task-stage t is written as [$y(t)$,$\mathbf{a}(t-1)$] because the agent plugs $y(t)$ into the first input layer and uses $\mathbf{a}(t-1)$ (activations at previous task-stage $t-1$) for *propagating activations* with Eq. (2). In other words, our NN-agent employs a multiple-layered version of Eq. (2), given at task-stage t by:

$$\begin{aligned} a_j^s(t) &= \beta_j a_j^s(t-1) + (1-\beta_j) f^s\left(net_j^s(t)\right) \quad (3)\\ &= \beta_j a_j^s(t-1) + (1-\beta_j) f^s\left(\textstyle\sum_i \theta_{i,j}^{s-1}(t) a_i^{s-1}(t)\right) \end{aligned}$$

to generate Critic's value and Actor's random action choice. For **temporal difference** (**TD**) reinforcement learning [14, 11], the agent needs to compute $\mathbf{a}(t+1)$ at the next task-stage $t+1$ using the same internal decision $\boldsymbol{\theta}(t)$ as at task-stage t because $\boldsymbol{\theta}(t+1)$ is not determined before the TD error is computed (as described below). For this purpose, the agent uses the next equation to produce node outputs:

$$\begin{aligned} a_j^s(t+1) &= \beta_j a_j^s(t) + (1-\beta_j) f^s\left(net_j^s(t+1)\right) \qquad (4) \\ &= \beta_j a_j^s(t) + (1-\beta_j) f^s\left(\sum_i \theta_{i,j}^{s-1}(t) a_i^{s-1}(t+1)\right). \end{aligned}$$

Next, we explain how to implement AC-learning with Eqs. (3) and (4) for our NN-agent that attempts to solve a non-Markov problem (i.e., external task).

We now consider a situation where the agent is at state $[y(t), \mathbf{a}(t-1)]$ (at task-stage t), and assume using Eq. (3) that the agent has computed approximate value $V(y(t), \mathbf{a}(t-1))$ by Critic and also obtained external decision $z(t)$ from the policy function $A(y(t), \mathbf{a}(t-1))$ by Actor. The agent exercises action $z(t)$, and observes the next successor (external) state $y(t+1)$ plus the *current-stage* (*history-dependent*) *reward* (between task-stages t and $t+1$) $R(y^t, z(t))$, wherein $y^t \overset{\text{def}}{=} \{y(1), y(2), \cdots, y(t)\}$, resulting from decision $z(t)$ through interaction with external dynamics. The agent then uses that new state $[y(t+1), \mathbf{a}(t)]$ as input to do **forward pass** with Eq. (4) through the network: (a) plugging the current state $y(t+1)$ as input into the nodes at the first input layer, (b) propagating the node activations $a^s(t+1)$ with $\mathbf{a}^s(t)$ used for Eq. (4) through all N-layers ($s = 1, \cdots, N$), (c) producing the final outputs $a^N(t+1)$; this gives Critic's value $V(y(t+1), \mathbf{a}(t))$, an *estimated value* [i.e., value incurred by that selected action $z(t)$] of the next state $y(t+1)$ at task-stage $t+1$. The agent now computes the TD error (i.e., residual for Critic): $R(y^t, z(t)) + V(y(t+1), \mathbf{a}(t)) - V(y(t), \mathbf{a}(t-1))$, and converts the error to the associated residual (or internal reinforcement) for Actor. The agent then propagates those residuals backward (i.e., **backward pass**) to obtain internal decision $\boldsymbol{\theta}^s(t+1)$ at each layer-stage s (s=$N-1, \cdots, 1$) in an attempt to solve the N-stage *internal* problem. Consequently, $y(t+1)$ and $\boldsymbol{\theta}(t+1)$ are now determined. After advancing to the successor state $y(t+1)$, our agent repeats the whole procedures at that new state.

Furthermore, in *finite-horizon* decision problems with the total (discounted) reward criteria, we can employ **activation-resetting** $\mathbf{a}(0)$=0 whenever the agent returns to the initial task-stage t=1; that is, $V(y(1), \mathbf{a}(0)) = V(y(1), 0)$ intuitively because node activations at the first task-stage have nothing to do with what they were before at the terminal task-stage (see Section 3.2). A similar idea can be found in *infinite-horizon* decision problems with the average-reward criteria to reset the *eligibility traces* (see Chapter 7 in [14]) whenever the agent returns to certain designated states (i.e., whenever a "renewal" occurs); see details on such resetting schemes in [15, 13, 16].

3 Experiments: A Four-Stage Longest Path Problem

We consider a triangular four-stage (i.e., four task-stage) path network in a *deterministic* discrete two-action environment, wherein a transition from vertex (i.e., state) to vertex incurs a reward assigned to each arc, as shown in Figure 1(left). The agent's objective is to learn the optimal sequence of four actions that maximizes the total rewards (including the terminal task-stage value). We assume that the agent always starts at vertex A, choosing either *action* d (going diagonally downward) or *action* u (going diagonally upward) at each vertex. The environment accordingly informs the agent of the next vertex and the transitional value associated with the action taken. Since the process is *deterministic*, by action d an agent *always* goes diagonally downward, although the model-free agent neither knows nor uses this fact during its learning. For the fourth action, the terminal value (all zeros in our example) provided by the environment is used as the value of the next state rather than the agent's outputs. This is the realization of the boundary conditions.

Moreover, the process is *non-Markovian* because of the *additional reward rule* (or *bonus-rule*, in short); that is, "an additional value (*bonus*) of 9" is accrued if the transition at any stage matches the transition two stages before." For instance, an action sequence "d-u-d-d" yields the total (reward) value 27 [$= 4+2+(6+\text{bonus } 9)+6$]. Of course, the bonus-rule and the incurred-reward data used by the environment are unknown to our model-free agent, and are not explicitly learned during the procedure. The agent is in a non-Markovian domain because the agent at vertex E [see Figure 1(left)] does not know the previous vertex (B or C) just by observing the current state (vertex E) alone, yet it is crucial in making an optimal action due to the bonus-rule. In other words, our NN agent has no explicit memory other than memory of the current and the next states plus the associated one-stage reward. It does, of course, have an implicit memory encoded in its hidden node activations. This is a very simplified situation, but is related to complicated real-life situations; for instance, a situation where an autonomous mobile agent (or robot) should be able to deal with incomplete state descriptions generated by limited sensors (e.g., one can perceive the current vertex alone).

3.1 Model-based approach: Dynamic-Programming (DP) algorithm

We first show a model-based approach by classical DP-algorithm [1], which uses world models, such as a tran-

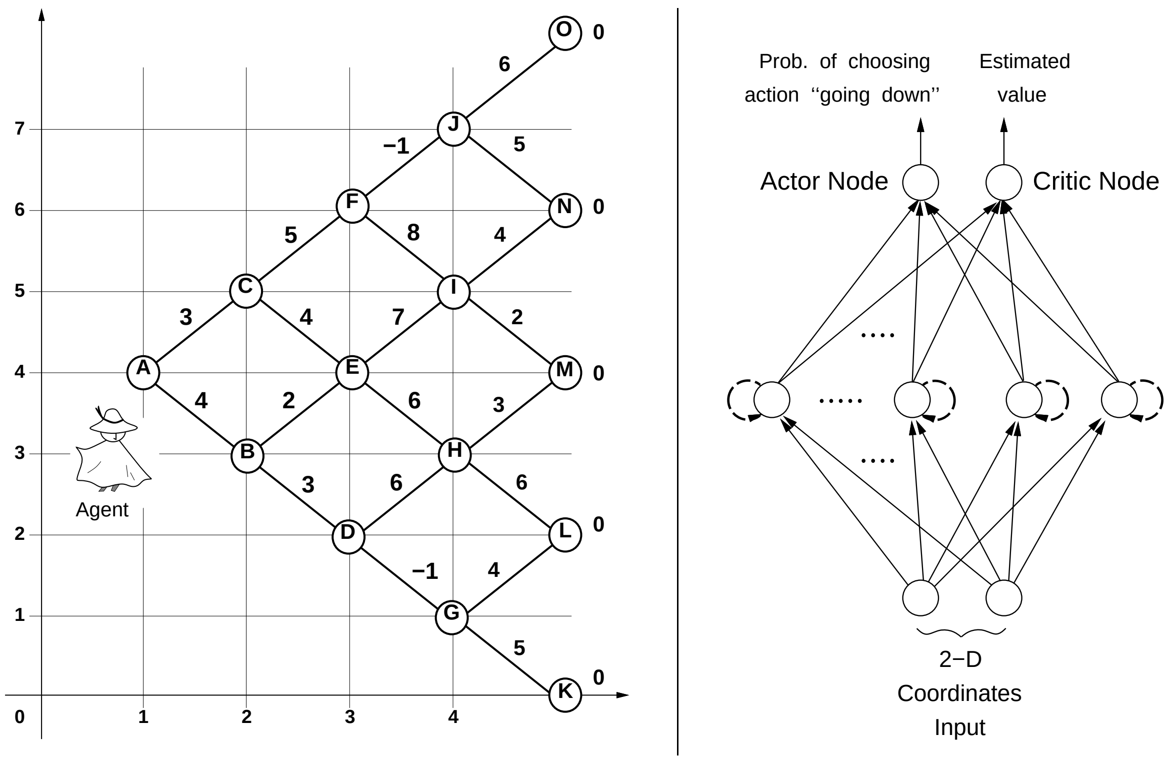

Figure 1. A four stage longest path problem (left) in the x-y coordinate system, and our actor-critic NN agent (right) wherein discretized Cohen-Grossberg dynamical formulas [see Eqs. (3) and (4)] describe the hidden-node activations.

sition model and a reward (or cost) model. To solve our path problem with the bonus-rule, DP requires that one increase the arguments in the **optimal value function** so that the Markov property holds. Choosing proper arguments for the value function corresponds to enlarging the state space to make the process Markovian, and those arguments must **explicitly** define the appropriate amount of information including the bonus-rule; for instance, the current two-dimensional coordinates [e.g., vertex H expressed as (4,3)] *plus* the last two consecutive actions. Here is our classical backward DP-formulation that needs the following four definitions: (1) Define the *optimal value function* $V(x, y, z_1, z_2)$ as "maximum reward-to-go, starting at vertex (x, y) (i.e., state y at stage x) to terminal vertices with action z_1 at stage $x-1$, and z_2 at stage $x-2$." (2) Define the **recurrence relation** using one-stage reward $R_z(x, y)$ associated with action z at vertex (x, y):

$$V(x,y,U,D) = \max \begin{cases} u : R_u(x,y) + V(x+1, y+1, U, U) \\ d : R_d(x,y) + V(x+1, y-1, D, U) + \text{bonus} \end{cases}$$

$$V(x,y,U,U) = \max \begin{cases} u : R_u(x,y) + V(x+1, y+1, U, U) + \text{bonus} \\ d : R_d(x,y) + V(x+1, y-1, D, U) \end{cases}$$

with obvious modifications for $V(x, y, D, U)$ and $V(x, y, D, D)$. (3) Define the **boundary conditions**: $V(5, 0, -, -) = V(5, 2, -, -) = V(5, 4, -, -) = V(5, 6, -, -) = V(5, 8, -, -) = 0$. (4) Define an **optimal policy function**, which is clear from the above recurrence relation: $\pi(x, y, -, -) = z$, where z is action d or u, such that z maximizes $V(x, y, -, -)$.

Using **full backups**, the DP-algorithm yields the optimal sequence of action "u-d-u-d" with the *exact* maximum total value "34" (including double bonus), which corresponds to the following optimal path:

$$\text{A} \underset{3}{\overset{up}{\Longrightarrow}} \text{C} \underset{4}{\overset{down}{\Longrightarrow}} \text{E} \underset{7+\text{bonus}}{\overset{up}{\Longrightarrow}} \text{I} \underset{2+\text{bonus}}{\overset{down}{\Longrightarrow}} \text{M}.$$

In the next model-free approach, only the current vertex (x, y) (i.e., observable state) is used as input to our actor-critic learning agent using **sample backups**.

3.2 Model-free approach: Actor-Critic algorithm

As a model-free approach to the posed path problem, we shall demonstrate our AC-learning algorithm described in Section 2.2, using a partially-recurrent AC-network [see Figure 1(right)], which is obtained by applying Eqs. (3) and (4) to describe the hidden-node outputs of an MLP. Furthermore, we used *activation-resetting* as defined and justified in Section 2.2. Note in the AC-network architecture [*] that the following two final out-

[*] An alternative NN model is a two-output Elman-network with a subset of hidden nodes used as the context units; see details in [4].

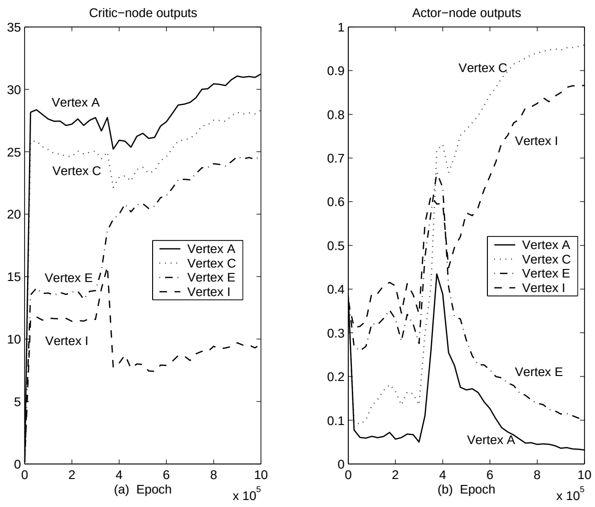

Figure 2. A sample AC-learning curve till epoch 1,000,000, obtained at (left) Critic node and (right) Actor node: At an early stage, our AC-agent happened to dictate a sub-optimal sequence of actions "u-u-u-u." After many epochs, however, the agent found the optimal sequence "u-d-u-d." The estimated values at epoch 2,000,000 at vertices A, C, E, and I, respectively, with all the values averaged over the last 1,000 epochs and the bracketed values denoting the desired ones, were: Actor's outputs (P_{down}): 0.02 [0.0], 0.99 [1.0], 0.06 [0.0], and 0.91 [1.0], and Critic's outputs (total rewards): 32.4 [34.0], 29.4 [31.0], 25.6 [27.0], and 10.3 [11.0].

put nodes suffice to solve the "two-action" problem: Critic node that produces the current estimated value, and Actor node that generates P_{down}, the current probability of action d "going down." Actor node has the sigmoidal logistic activation function constrained to lie between zero and one, whereas Critic node has the linear identity function. Hence, Critic's residuals are likely to be larger than Actor's; so, Critic's value-updates tend to be *faster* than Actor's policy updates, although Critic and Actor share a subset of parameters connecting to the first input layer. Our AC-net has 25 context nodes (hence, a 2-25-2 NN), receiving only two inputs (two-dimensional coordinates of the current vertex alone), and is trained by the backpropagation (or incremental gradient) method with a fixed momentum term (0.8) using TD(0). At vertex A (task-stage 1), just as hidden-node activations $\mathbf{a}(0)$ were reset to 0, likewise the momentum term was reset to 0. In addition, all β_i were set equal to the same constant (0.5 in the posed problem).

4 Discussion

The simulation results show that our AC-net can produce outputs close to the desired values but converged *very slowly* because the agent learned in a totally model-free fashion with current observable states only as input without explicitly learning what enlargement of state would render the situation Markovian. Learning acceleration algorithms described in [13, 17] might be worth trying after suitable modifications if applicable. In addition, the parameter setups (e.g., β and learning rates) greatly affected learning behaviors, and we observed that the performance appeared better when β was set closer to 0.5. Choosing proper parameters are still a matter of art and found by the process of trial and error. Some choices appeared to lock-in on non-optimal solutions, although the possibility of finding the optimal path after additional epochs cannot be ruled out; see Figure 2 for a sample AC-learning curve with $\beta = 0.5$.

Furthermore, we compared an agent comprising two *separate* networks: a single-output Actor Elman-net (7-5-1 NN) with five context units, and a single-output Critic net (2-15-1 NN) with Eqs. (3) and (4) applied (with β=0.5) to the 15 hidden nodes with the activation-resetting. In this agent, Actor and Critic have no shared parameters; hence, Actor's policy-updates can be readily made *slower* (e.g., by using smaller learning rates) than Critic's value-updates in an attempt to achieve better convergence [12, 13]. In fact, we feel that the performance improvement was relatively more easily accomplished for the two-net agent than that for the single-net agent in Figure 1(right). As a result, among 100 trials, the two-net agent found an optimal sequence 39 times by epoch 2,000,000, whereas the single-net agent found it only 10 times. Of course, if the learning rates are carefully fine-tuned specially for each trial, then the success rates of both nets may not differ greatly.

5 Conclusion and Future Work

Discretized Cohen-Grossberg dynamic formulas (3) and (4) are readily applicable to an MLP (without changing its size), resulting in a partially-recurrent NN. In consequence, the NN has the same ability as Elman and Jordan-type networks in developing *sensitivity* of *value* and *action* to whatever in the prior path history is potentially relevant to environmental dynamics and reinforcement, and thus can solve non-Markov decision problems in a *totally model-free fashion*.

We have chosen a toy, discrete decision variable, deterministic problem with non-Markovian reinforcement but Markovian dynamics to illustrate our *model-free approach* even though this problem could have been solved merely by trying all 16 possible sequences of four decisions. While it is ideal for determining by hand the optimal solution and for presenting results, this type of problem (involving *discrete* values and decisions) is difficult for neural networks, which perform best when learning smooth continuous data.

We have also applied our method to simulation of a baseball outfielder learning from experience to move to the proper location for catching a batted ball (see [18] for a reinforcement learning approach). The fielder observes the angle of the ball's elevation. In [18], it is also assumed that the fielder observes such stimuli as the *second derivative* of the tangent of the ball's elevation angle, a feature that is not directly observable. Using only the angle plus the fielder's velocity as inputs but using a past-sensitive recurrent network as we have in this paper, our simulated fielder learned fairly successful behavior. The work remains unpublished since our fielder always starts moving in the same direction independent of whether he should run in toward the batter or out toward the outfield fence, before correcting himself, and because our current procedure often leads to the fielder accelerating until his velocity exceeds what is humanly possible. We feel, however, that the baseball problem, involving fielder acceleration decisions at about 50 discretized slices of time, shows that our procedure can, in principle, solve large problems with suitably large, complex neural networks and perhaps better learning rules.

Furthermore, in principle, our method is applicable to large **stochastic** problems with continuous decision variables and non-Markovian dynamics as well as reinforcement. Extension of our longest path problem to *stochastic* version leads to at least two distinct problems: one with prior-action dependent bonus rules, and another with prior-transition dependent bonus rules. We are currently investigating the generalization of our longest path problem to the version with a prior-transition dependent bonus rule: A decision at a vertex only determines the probability of moving diagonally up or down in Figure 1 (left). Needless to say, our fundamental concept is not confined to the actor-critic learning with total reward criteria; so, it would be of interest to apply it in other contexts (e.g., actor-only learning [15, 13, 16] with average-reward criteria and stochastic problems) for performance comparison in the future.

References

[1] Stuart E. Dreyfus and Averill M. Law. *The Art and Theory of Dynamic Programming*, volume 130 of *Mathematics in Science and Engineering*. Academic Press Inc., 1977.

[2] A. K. McCallum. *Reinforcement Learning with Selective Perception and Hidden State*. PhD thesis, Department of Computer Science, University of Rochester, Rochester, NY, 1995. revised in 1996.

[3] Barak A. Pearlmutter. Gradient calculations for dynamic recurrent neural networks: A survey. *IEEE Transactions on Neural Networks*, 6(5):1212–1228, 1995.

[4] Eiji Mizutani and Stuart E. Dreyfus. Totally model-free reinforcement learning by actor-critic Elman networks in non-Markovian domains. In *Proceedings of the IEEE International Conference on Neural Networks, part of the World Congress on Computational Intelligence (Wcci'98)*, pages 2016 – 2021, Alaska, USA, May 1998.

[5] J. Hertz, A. Krogh, and R. G. Palmer. *Introduction to the theory of neural computation*. Addison-Wesley, Reading, MA, 1991.

[6] Eiji Mizutani. Chapter 10: Learning from Reinforcement. In *Neuro-Fuzzy and Soft Computing: a computational approach to learning and machine intelligence*, pages 258–300. J.-S. Roger Jang, C.-T. Sun and E. Mizutani. Prentice Hall, 1997.

[7] Michael A. Cohen and Stephen Grossberg. Absolute stability of global pattern formation and parallel memory storage by competitive neural networks. *IEEE Trans. on Systems, Man, and Cybernetics*, 13(5):815–826, 1983.

[8] Fernando J. Pineda. Generalization of back-propagation to recurrent neural networks. *Physical Review Letters*, 59(19):2229–2232, November 1987.

[9] Shun-Ichi Amari. Characteristics of random nets of analog neuron-like elements. *IEEE Trans. on Systems, Man, and Cybernetics*, 2(5):643–657, November 1972.

[10] Paul J. Werbos. Backpropagation through time: What it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, October 1990.

[11] Dimitri P. Bertsekas and John N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientifi c, Belmont, MA, 1996.

[12] V. R. Konda and J. N. Tsitsiklis. Actor-critic algorithms. In *Advances in Neural Information Processing Systems (NIPS 1999)*, volume 12, pages 1008–1014. MIT Press, 2000.

[13] Vijaymohan R. Konda. *Actor-Critic Algorithms*. PhD thesis, EECS Department, Massachusetts Institute of Technology, Cambridge, MA, 2002.

[14] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA., 1998.

[15] Peter Marbach and John N. Tsitsiklis. Simulation-based optimization of Markov reward processes. *IEEE Transactions on Automatic Control*, 46(2):191–209, 2001.

[16] Eiji Mizutani. Sample path-based policy-only learning by actor neural networks. In *Proceedings of the IEEE International Conference on Neural Networks (vol. 2)*, pages 1245–1250, Washington, D.C., July 1999.

[17] Michail G. Lagoudakis and Ronald Parr. Model-free least squares policy iteration. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems (NIPS 2002)*, volume 14. MIT Press, 2002.

[18] Rajarshi Das and Sreerupa Das. Using reinforcement learning to catch a baseball. In *Proceedings of the IEEE International Conference on Neural Networks, part of the World Congress on Computational Intelligence (Wcci'94)*, volume 5, pages 2808 – 2818, Orlando, Florida, USA, July 1994.