

# Two stochastic dynamic programming problems by model-free actor-critic recurrent-network learning in non-Markovian settings

Eiji Mizutani

Department of Computer Science  
Tsing Hua University  
Hsinchu, 300 TAIWAN  
E-mail: eiji@wayne.cs.nthu.edu.tw

Stuart E. Dreyfus

Dept. of Industrial Eng. & Operations Research  
University of California, Berkeley  
Berkeley, CA 94720 USA  
E-mail: dreyfus@ieor.berkeley.edu

**Abstract**— We describe two stochastic non-Markovian dynamic programming (DP) problems, showing how the posed problems can be attacked by using actor-critic reinforcement learning with recurrent neural networks (RNN). We assume that the current state of a dynamical system is “completely observable,” but that the rules, unknown to our decision-making agent, for the current reward and state transition depend not only on current state and action, but on possibly the “entire history” of past states and actions. This should not be confused with problems of “partially observable Markov decision processes (POMDPs),” where the current state is only deduced from either partial (observable) state alone or error-corrupted observations [11]. Our actor-critic RNN agent is capable of finding an optimal policy, while learning neither transitional probabilities, associated rewards, nor by how much the current state space must be augmented so that the Markov property holds. The RNN’s recurrent connections or context units function as an “implicit” history memory (or internal state) to develop “sensitivity” to non-Markovian dependencies, rendering the process Markovian implicitly and automatically in a “totally model-free” fashion. In particular, using two small-scale longest-path problems in a stochastic non-Markovian setting, we discuss model-free learning features in comparison with the model-based approach by the classical DP algorithm.

## I. INTRODUCTION

In a stochastic non-Markovian environment, an agent has to manage non-Markovian situations where both reinforcement signals and state transitions may depend *arbitrarily* on the past history of both the states visited and the agent’s outputs (e.g., decisions) during interaction with the dynamic world (i.e., environment). In general, agent’s strategies can be categorized into two major approaches: (1) *model-based* (or *model-building*) *approach*, wherein the agent attempts to learn **explicitly** what prior events matter for making an optimal action, and (2) *model-free approach*, wherein the agent retains some **internal state** (or **implicit 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. In the *temporal-difference reinforcement learning* literature [2], [12], [9], the model-building approach is popular, whereby the agent is assumed to be able to model the mappings from *past and current states and past actions*

to *reinforcement signals and state transitions* by observing states, actions, and reinforcement signals; for example, *Utile Suffix Memory* [6] constructs trees with associated Q-values. Stochastic *finite-state machines* can also be employed to model the environment explicitly. In control engineering, the model-building approach is known as *system identification*.

In a model-free approach, *internal memory* can be realized with *recurrent neural networks* (see [10], [7], [8] and Chap. 7 in [4]) 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. In this paper, we describe two stochastic decision problems with non-Markovian reinforcement signals that can be solved by “model-based” *dynamic programming* if the reinforcement and transition rules are known, and then show our “model-free” *actor-critic reinforcement learning* approaches using a partially recurrent Elman network when the above rules are unknown. (See [13] for *Q-learning* network approaches.) Our model-free approach can be extended to more realistic situations; for instance, a mobile robot tries to intercept a projectile on a windy day, where the observable current state is the relative position of the projectile but its velocity and direction of motion depend on past observables. Our method also might learn to predict (rather than control) the future price of a stock in a non-Markovian model-free way using as input only the *current* price of the stock and the *current* values of suitable other economic indicators rather than *explicitly* including trends, etc.

## II. A STOCHASTIC FOUR-STAGE LONGEST PATH PROBLEM

We describe two stochastic non-Markovian longest path problems. Fig. 1(left) illustrates our triangular four-stage path network in a *stochastic* discrete two-action environment, wherein a transition from vertex (i.e., state) to vertex incurs a reward assigned to each arc. We assume the reward is uniquely determined by a transition, but our simulation method would also work if the rewards were stochastic with the expected values shown in Fig. 1(left). The agent’s objective is to learn

the optimal policy that maximizes the expected total rewards (including the terminal stage value). We assume that the agent always starts at vertex A, choosing either *action d* (try to go diagonally downward) or *action u* (try to go diagonally upward) at each vertex. The environment accordingly informs the agent of the next vertex and the transitional value associated with the action taken. The process is *stochastic* because we assume that when the agent tries to go in a certain direction, it does so with probability  $p$  ( $> 0.5$ ) but it goes in the other direction with probability  $1 - p$ . The model-free agent neither knows nor uses this transitional rule during its learning. For the fourth action, the terminal value 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.

Furthermore, we have introduced an *additional reward rule* (or what we call *bonus rule*) to render the process *non-Markovian*. In our previous work [7], [8], we discussed a *deterministic* longest path problem with a *transition-dependent* bonus rule, in which an additional value (*bonus*) was accrued if the transition at any stage matched the transition two stages before. As an extension to a *stochastic* version, we consider the next two stochastic problems: (1) one with prior-transition dependent bonus rules, and (2) another with prior-action dependent bonus rules. In problem (1), an additional value (*bonus*) of 4 is accrued if the actual *transition* at any stage matches the actual *transition* two stages before regardless of the actions taken by the agent. On the other hand, in problem (2), an additional value (*bonus*) of 4 is accrued if the *action* taken at any stage matches the *action* taken two stages before whatever actual transitions followed. The latter problem is more challenging because both action and transition affect the total expected rewards in a more complex manner. In any event, our actor-critic agent needs to estimate the expected reward-to-go at a vertex given a past history, and makes a decision to try to move either diagonally down or up. Needless to say, the bonus rules as well as the incurred reward and the transitional probability used by the environment are unknown to our model-free agent, and are not explicitly learned during the agent’s interactions. Furthermore, the agent is in a non-Markovian domain because the agent at any vertex [see Fig. 1(left)] does not explicitly remember the previous history, which is crucial in making an optimal action due to the bonus rules; it just observes the current state (vertex) alone.

#### A. DP Solution to the Prior-Transition Dependent Problem

The classical dynamic programming (DP) algorithm is a model-based approach, requiring *explicit* world models, such as a transition model and a reward (or cost) model, to allow a DP solution based on the *principle of optimality*. To solve the longest path problem with the prior-transition dependent bonus rule, DP requires that one increase the arguments in the **optimal value function** so that the Markov property holds. That is, we enlarge the state space to make the process Markovian, by choosing proper arguments for the value function, which is in general a matter of art (rather than science); see a variety of

examples in [1], [3]. These arguments must **explicitly** define the appropriate amount of information given the bonus rule; for instance, the current two-dimensional coordinates *plus* the last two consecutive “actual” transitions no matter what actions were taken. Our classical backward DP-formulation needs the following four definitions: (1) Define the *optimal value function*  $V(x, y, z_1, z_2)$  as “maximum expected reward-to-go, starting at vertex  $(x, y)$  (i.e., state  $y$  at stage  $x$ ) to terminal vertices with **transition**  $z_1$  one stage before, and  $z_2$  two stages before.” (2) Define the **recurrence relation** using one-stage reward  $R_z(x, y)$  associated with action  $z$  at vertex  $(x, y)$ :

$$\begin{aligned}
 & V(x, y, u, d) \\
 &= \max \begin{cases} u : p [R_u(x, y) + V(x + 1, y + 1, u, u)] \\ \quad + (1 - p) [R_d(x, y) + V(x + 1, y - 1, d, u) + \text{bonus}] \\ d : p [R_d(x, y) + V(x + 1, y - 1, d, u) + \text{bonus}] \\ \quad + (1 - p) [R_u(x, y) + V(x + 1, y + 1, u, u)] \end{cases} \\
 & V(x, y, u, u) \\
 &= \max \begin{cases} u : p [R_u(x, y) + V(x + 1, y + 1, u, u) + \text{bonus}] \\ \quad + (1 - p) [R_d(x, y) + V(x + 1, y - 1, d, u)] \\ d : p [R_d(x, y) + V(x + 1, y - 1, d, u)] \\ \quad + (1 - p) [R_u(x, y) + V(x + 1, y + 1, u, u) + \text{bonus}] \end{cases}
 \end{aligned}$$

with obvious modifications for  $V(x, y, d, u)$  and  $V(x, y, d, d)$ . (3) Define the **boundary conditions**:  $V(5, 0, -, -) = -1$ ;  $V(5, 2, -, -) = 3$ ;  $V(5, 4, -, -) = V(5, 6, -, -) = 0$ ;  $V(5, 8, -, -) = 1$ . (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** and  $p = 0.9$ , the DP-algorithm yields the optimal policy, which can be summarized with the eight possible transitional paths (or trajectories) in Table I; see the first three columns therein. Note in Paths 3 and 4 that action  $d$  is not optimal at vertex E even if the “actual” transitions were A-B-E (i.e., “down-up” transition) because the arc reward between vertices E and I is 7 and we chose a relatively small bonus value as 4. If the bonus value is large (e.g., 9), then the optimal action always favors the decision to receive the bonus.

#### B. DP Solution to the Prior-Action Dependent Problem

Next, for solving the prior-action dependent problem, the arguments of the value function must *explicitly* define the current two-dimensional coordinates *plus* the last two consecutive actions taken whatever actual transitions followed. Our classical backward DP-algorithm needs the four definitions: (1) Define the *optimal value function*  $V(x, y, z_1, z_2)$  as “maximum expected reward-to-go, starting at vertex  $(x, y)$  (i.e., state  $y$  at stage  $x$ ) to terminal vertices with **action**  $z_1$  one stage before, and  $z_2$  two stages before.” (2) Define the *recurrence relation* using one-stage reward  $R_z(x, y)$  associated with action  $z$  at vertex  $(x, y)$ :

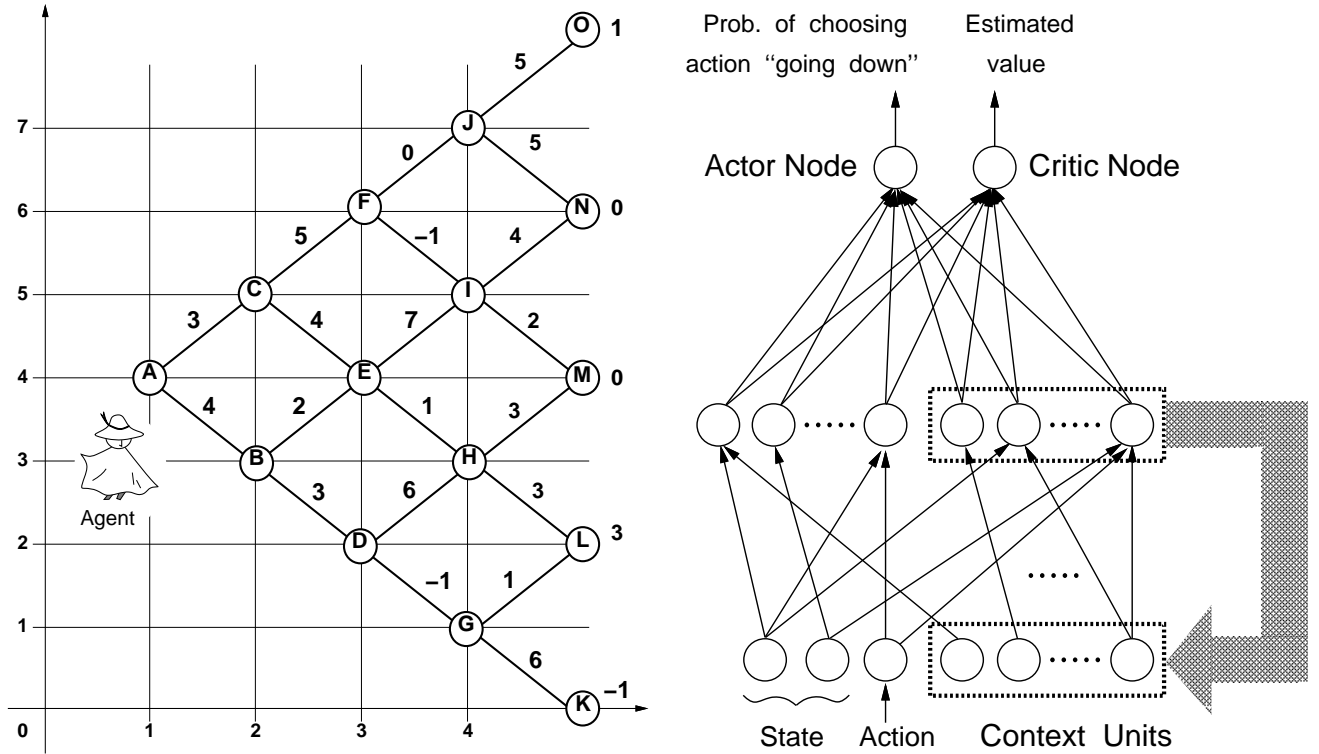


Fig. 1. A four-stage stochastic longest path problem (left) in the  $x$ - $y$  coordinate system, and our actor-critic Elman-network agent (right) for solving a stochastic longest path problem with our action-dependent bonus rule (see Section II-B).

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

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

with obvious modifications for  $V(x, y, d, u)$  and  $V(x, y, d, d)$ . The rest of definitions, (3) boundary conditions and (4) optimal policy function, are the same as those defined in Sec. II-A.

### III. SIMULATION BY MODEL-FREE ACTOR-CRITIC LEARNING

As a model-free approach to the aforementioned two stochastic longest path problems, we shall demonstrate our AC-learning using a two-output single-hidden-layer Elman-network with a subset of hidden nodes used as the context units. Specifically, in our experiment, we used seven nodes as context units among 11 hidden nodes, and the standard AC-learning algorithm (see details in [7], [8]) was employed. We introduced an **activation-resetting** method; that is, whenever the agent returns to vertex A at the first stage, we reset all the

context-unit outputs equal to 1.0. The intuitive reason for this is that neurons' activations at the moment a new excursion starts at vertex A have nothing to do with what they were when ending up at a certain vertex at the terminal stage before. The two final output nodes suffice to solve the posed "two-action" problems: *Critic node* estimates the expected reward-to-go at the current vertex given a past history, and *Actor node* generates  $P_{down}$ , the current probability of action  $d$  "try to go down." In particular, 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 (see [5] for convergence issues in AC-learning), although Critic and Actor share a subset of parameters (hidden parameters) between the first input layer and the hidden layer in our RNN. In the following results, we used a fixed transition probability  $p = 0.9$ .

#### A. The Prior-Transition Dependent Problem

For solving the prior-transition dependent problem (see Sec. II-A), our model-free agent has no explicit memory other than memory of the current and the next states plus the associated one-stage reward. Only the current vertex  $(x, y)$  (i.e., observable state) is used as input to our actor-critic learning agent using **sample backups**. This is essentially the same as the model-free agent for the deterministic case [7]. In our simulation, the procedure worked better when the vertex

TABLE I

The eight possible transitional paths (columns 1 and 2), the associated optimal policy (column 3), and the estimated expected value-to-go and the action (in parentheses) generated by our RNN agent (the last large column) for the prior-transition dependent problem (in Sec. II-A); see Fig. 1 (left) for definitions of vertices and transition values.

	Transitions	Policy	Estimated values (action)			
Path 1	A-B-D-G	u-d-u-d	21.67 (0.10)	16.75 (0.92)	14.20 (0.09)	7.65 (0.80)
Path 2	A-B-D-H	u-d-u-d	21.67 (0.10)	16.75 (0.92)	14.20 (0.09)	8.91 (0.92)
Path 3	A-B-E-H	u-d-u-u	21.67 (0.10)	16.75 (0.92)	12.48 (0.23)	6.97 (0.32)
Path 4	A-B-E-I	u-d-u-u	21.67 (0.10)	16.75 (0.92)	12.48 (0.23)	5.64 (0.20)
Path 5	A-C-F-J	u-d-u-u	21.67 (0.10)	18.89 (0.90)	11.67 (0.09)	9.14 (0.09)
Path 6	A-C-F-I	u-d-u-u	21.67 (0.10)	18.89 (0.90)	11.67 (0.09)	5.89 (0.25)
Path 7	A-C-E-I	u-d-u-d	21.67 (0.10)	18.89 (0.90)	15.38 (0.10)	5.63 (0.88)
Path 8	A-C-E-H	u-d-u-d	21.67 (0.10)	18.89 (0.90)	15.38 (0.10)	8.50 (0.87)

(state) was linearly transformed to  $(2x - 5, y - 4)$  as input to the RNN agent rather than  $(x, y)$  as input.

The last column in Table I shows the results obtained after epoch 1,000,000: For instance, along Path 1, the actual trajectory was A-B-D-G and the optimal action at those four vertices A, B, D, and G were u-d-u-d; in this case, Critic node produced the estimated reward-to-go values: 21.67, 16.75, 14.20, and 7.65 at vertices A, B, D, and G (while the exact values given by the DP algorithm in Sec. II-A are 22.80, 17.74, 14.92, and 8.5), and Actor node generated actions 0.10, 0.92, 0.09, and 0.80 (while the exact values are 0, 1, 0, and 1) as the probability of trying to go down. All decisions, when rounded to the nearest integer, are correct. The least accurate decision probability (0.32 in Table I) is for the situation least likely to occur during simulation, since all the three transitions occurring on Path 3 had probability 0.1 given the probable decision. Overall, our AC-agent 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.

### B. The Prior-Action Dependent Problem

Since the current vertex is an input to our model-free actor-critic learning RNN agent at each stage of the process, the activations of the context units are *sensitive to past trajectory*. For the action-dependent problem (in Section II-B), our RNN agent needs an additional input of the *most recently taken action* in order to make the activations of the context units *sensitive to past actions* also. That is, not only the observable state [i.e., current vertex  $(x, y)$  after linearly transformed as  $(2x - 5, y - 4)$ ] but also one prior action (i.e., most recent action) need to be plugged in as input to our RNN agent; specifically, we employed input value 1 for action *up*,  $-1$  for action *down*, and 0 at vertex A. Fig. 1(right) depicts our actor-critic Elman-network agent specially-gearred to the posed action-dependent problem. Our AC-learning network architecture looks similar to the Q-learning network with both state and action as input, but it is purely the AC-learning

network because it has only the most recent action as input. Therefore, our AC-learning agent won't pursue how bad the bad decisions are unlike the Q-learning agent; in other words, our AC-agent needs a lesser number of value evaluations than the Q-learning agent.

There are too many action sequences and resulting transition sequences to fully report numerical results after 1,000,000 epochs; hence we report our RNN outputs for some interesting related cases: In particular, Fig. 2 shows Critic-node outputs and Actor-node outputs at vertex E (3,4) in the four different situations below:

- (1) Actions  $u$  and then  $d$ , but actual transitions were “down-up,” hence, vertex sequence A-B-E; see solid line in Figures 2(a) and (b).
- (2) Actions  $u$  and then  $d$ , and actual transitions were “up-down,” hence, vertex sequence A-C-E; see dotted line in Figures 2(a) and (b).
- (3) Actions  $d$  and then  $u$ , and actual transitions were “down-up,” hence, vertex sequence A-B-E; see solid line in Figures 2(c) and (d).
- (4) Actions  $d$  and then  $u$ , but actual transitions were “up-down,” hence, vertex sequence A-C-E; see dotted line in Figures 2(c) and (d).

In situations (1) and (2), in spite of different trajectories, the values were gradually converging to the same expected “reward-to-go” function value, denoted by  $V(3, 4, d, u)$  in the notation used in Section II-B, and the agent was slowly learning the optimal decision  $u$ . Likewise, in situations (3) and (4), the agent was slowly learning the optimal decision  $u$  while the values were gradually converging to  $V(3, 4, u, d)$ . In this way, the values and decisions became more or less independent of transitions but dependent on actions.

Even in the former transition-dependent problem, we can make the same observation if we consider five or more stages. In the five-stage prior-transition dependent problem, for instance, getting to vertex M (requiring the fifth decision) by way of transition  $u-d-u-d$  (with double bonus) or transition  $d-u-u-d$  (with no bonus) should result in the same expected reward-to-go  $V(5, 4, d, u)$  and decision.

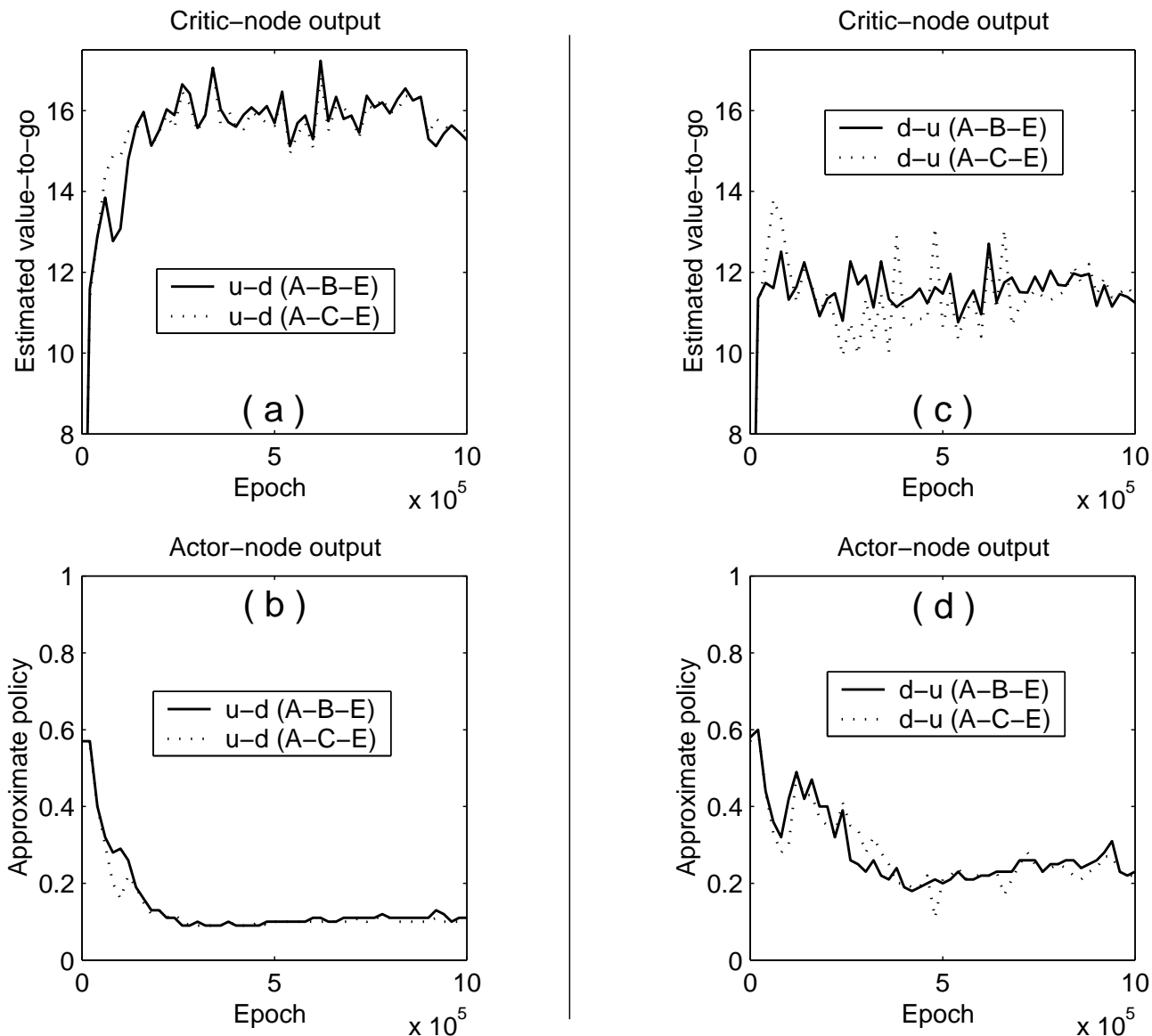


Fig. 2. Critic-node outputs and Actor-node outputs at vertex  $E$  in an AC-learning curve till epoch 1,000,000: (a) Estimated expected reward-to-go at vertex  $E$  after action  $u$  and then  $d$  were taken consecutively (denoted by  $u-d$ ) to reach vertex  $E$ ; here, the solid line shows the value-to-go when actual transitions were vertices  $A$ ,  $B$ , and then  $E$  (denoted by  $A-B-E$ ), whereas the dotted line shows the value-to-go when actual transitions were  $A-C-E$ . (b) Approximate policy, given by Actor’s output  $P_{down}$  at vertex  $E$  after action  $u$  and then  $d$  were taken consecutively ( $u-d$ ) to reach vertex  $E$ ; here, the solid line shows  $P_{down}$  when actual transitions were  $A-B-E$ , whereas the dotted line shows  $P_{down}$  when actual transitions were  $A-C-E$ . (c) Estimated expected reward-to-go when action sequence  $d-u$  led to vertex  $E$  with actual transitions  $A-B-E$  (solid line), and with actual transitions  $A-C-E$  (dotted line). (d) Approximate policy when action sequence  $d-u$  led to vertex  $E$  with actual transitions  $A-B-E$  (solid line), and with actual transitions  $A-C-E$  (dotted line).

#### IV. DISCUSSION

Our AC-learning recurrent-network procedure is applicable both (1) when a world-model is known (yet unknown to our model-free agent) and is used to generate sample paths as in our toy example problems; and (2) when the model is not known and sample paths are observed as is the case with real-world human behavior.

Besides those two model-free cases, if the model is known to the agent, model-based optimization methods such as dynamic programming are applicable. Suppose we are dealing with a process with  $m$  possible current states,  $q$  possible

decisions in each state, and  $m$  possible next states as a result. Suppose further that, according to the known model, the past history of the then-current state and the decision taken in that state during the previous  $K$  stages affect the generation mechanism of the current transition or of the stage cost. Then the number of dynamic programming states at each stage is  $O(m(mq)^K)$  and could easily become infeasible for even moderately sized  $K$ .

If, instead, one used reinforcement learning with a non-recurrent feedforward neural network as function approximator with *past history* of  $K$  pairs of prior “state and action” (hence,

$2K$  features in total) explicitly represented, then  $2K + 1$  (or  $2K + 2$  for Q-learning) input features would be required, still potentially a sizeable number. In contrast, our recurrent-network procedure would require only two input features (i.e., the current state and the decision one stage previous) even when  $K$  prior states and decisions are relevant. Of course, whenever a non-linear function approximator is used instead of a table of values, reinforcement learning, even Q-learning, cannot guarantee convergence to an optimal solution as one can with dynamic programming, but at least it offers a method of plausible successive improvement on a nominal guessed policy.

If a world-model is not known and one must rely on real-world samples, two alternatives to our type of procedure present themselves. First, one can use a reinforcement-learning type algorithm implemented with a non-recurrent feedforward neural network with past history explicitly input as described above for which one guesses how many prior stages  $K$  need be explicitly input into the neural network. However, it would be very difficult to statistically determine when an adequate  $K$  had been assumed. Second, one can attempt to learn the world-model based on sampling and then use the model as described earlier. Then one must guess a parameterized form of the world-model despite the fact that there are, literally, an infinite number of such forms. On the other hand, our recurrent procedure being sensitive to the entire past in a model-free fashion avoids those issues and still requires just two input features.

## V. CONCLUSION AND FUTURE WORK

Our AC-learning RNN agent develops *sensitivity of value and action* to whatever in the prior path history is potentially relevant to environmental dynamics and reinforcement, without identifying which recent-past stimuli should be used in addition to the current ones in determining actions, and thus can solve non-Markov decision problems in a *totally model-free fashion*. Presumably, other (partially) recurrent networks can accomplish the same task; for instance, a partially recurrent network, whose internal neuron dynamics are regulated by a discrete version of *Cohen-Grossberg node dynamics* [8]. The resulting NN has the ability to solve a non-Markov problem using different internal dynamics from those of the Elman network, and from those by backpropagation through time (BPTT) [14].

We have illustrated our *model-free learning approaches* to toy, stochastic problems with non-Markovian reinforcement but Markovian dynamics. This type of problem (involving *discrete* values and decisions) may be difficult for neural networks, which perform best when learning smooth continuous data. We are currently investigating our model-free AC-learning to simulate a baseball outfielder's learning from experience to move to the proper location for catching a batted ball using only the currently observable angle of elevation of ball as input to a past-sensitive RNN. In principle, our method is applicable to *large* stochastic problems with continuous decision variables and non-Markovian dynamics as well as

reinforcement. It is also applicable to problems with arbitrarily complex non-Markovian dynamical rules where there is no hope of learning the model based on observations.

## REFERENCES

- [1] Richard E. Bellman and Stuart E. Dreyfus. *Applied Dynamic Programming*. Princeton University Press, 1962.
- [2] Dimitri P. Bertsekas and John N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, Belmont, MA, 1996.
- [3] Stuart E. Dreyfus and Averill M. Law. *The Art and Theory of Dynamic Programming*. Mathematics in Science and Engineering, Vol. 130, Academic Press Inc. 1977.
- [4] J. Hertz and A. Krogh and R. G. Palmer. *Introduction to the theory of neural computation*. Addison-Wesley, MA. 1991.
- [5] Vijaymohan R. Konda. "Actor-Critic Algorithms" Ph.D Thesis, Dept. of EECS, Massachusetts Institute of Technology, 2002.
- [6] A. K. McCallum. "Reinforcement Learning with Selective Perception and Hidden State" Ph.D Thesis, Dept. of Computer Science, University of Rochester, 1995 (Revised in 1996).
- [7] 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 Joint Conference on Neural Networks, part of the World Congress on Computational Intelligence (Wcci'98)*, pp. 2016–2021, Alaska USA, May 1998. (available at <http://www.ieor.berkeley.edu/People/Faculty/dreyfus-pubs/Wcci98.ps> ).
- [8] Eiji Mizutani and Stuart E. Dreyfus. "On using discretized Cohen-Grossberg node dynamics for model-free actor-critic neural learning in non-Markovian domains" In *Proceedings of the IEEE Int'l Symposium on Computational Intelligence in Robotics and Automation (CIRA 2003)*, Vol.1, pages 1–6, Kobe JAPAN, July 16–20, 2003.
- [9] Eiji Mizutani. "Chapter 10: Learning from Reinforcement." In *Neuro-Fuzzy and Soft Computing: A Computational Approach to Learning and Machine Intelligence*. (J.-S. R. Jang, C.-T. Sun and E. Mizutani), pages 258–300, Prentice Hall, 1997. (<http://neural.cs.nthu.edu.tw/jang/book/>.)
- [10] Barak A. Pearlmutter. "Gradient Calculations for Dynamic Recurrent Neural Networks: A Survey" In *IEEE Transactions on Neural Networks*, Vol. 6, No. 5, pages 1212–1228, 1995.
- [11] Fernando Mark Pendrith. "On reinforcement learning of control actions in noisy and non-Markovian domains." Technical Report: UNSW-CSE-TR-9410, School of Computer Science and Engineering, The University of New South Wales, Sydney, Australia, August 1994.
- [12] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, 1998.
- [13] Steven D. Whitehead and Long-Ji Lin. "Reinforcement learning of non-Markov decision processes." *Artificial Intelligence*, pages 271–306, Vol. 73, 1995.
- [14] Paul J. Werbos. "Backpropagation Through Time: What It Does and How to Do It" In *Proceedings of the IEEE*, pages 1550–1560, Vol. 78, No. 10, Oct. 1990.