Sample path-based policy-only learning by actor neural networks

Eiji Mizutani *
Dept. of Industrial Engineering and Operations Research
University of California at Berkeley, Berkeley, CA 94720
Email: eiji@biosys2.me.berkeley.edu

Abstract

This paper highlights a **sample-path-based policy-only learning** algorithm for **regenerative (stochastic) processes** proposed by Marbach and Tsitsiklis [1; 2]. The algorithm attempts to optimize a randomized parameterized policy according to the **average cost** criteria in conjunction with the so-called **infinitesimal perturbation analysis (IPA) gradient estimation** technique.

We present our numerical studies, demonstrating this learning algorithm using small-scale problems; in particular, the parameterized policy-only agent is a **neural network** function approximator in the spirit of **neuro-dynamic programming** (or **reinforcement learning**).

I. Introduction

Policy-only learning is one of the representative architectures of **reinforcement learning** [3], or **neuro-dynamic programming (NDP)** [4; 5] methods, which form a class of **simulation-based learning** techniques. Many architectures usually involve some *parameterized function approximators* (typically, **artificial neural networks**) to allow some approximation/generalization capability. For solving large-scale Markov decision problems, those techniques attempt to overcome the limitations of (classical) *model-based synchronous* **dynamic programming (DP)** [6], by using *simulation* to estimate quantities of interest without involving heavy model-based computations.

Figure 1 pictures an idealized NDP-type policy-only agent realized by a neural network function approximator. Marbach and Tsitsiklis's algorithm nicely fits into this picture, implementing the actor network in the spirit of reinforcement learning.

In the early AI literature, Michie proposed a fairly naive *policy-only* algorithm, called MENACE (Matchbox Educable Naughts And Crosses Engine) algorithm, for playing a tic-tac-toe game [7]. The devices were just a few hundred *matchboxes* with colored beads; the matchboxes corresponded to possible game states and the colored beads specified admissible actions. Michie showed a simple reward-penalty scheme drove MENACE to learn winning trajectories in the tic-tac-toe game, by adjusting the policy alone based on experience (or simulation). (Detailed policy-only view of the MENACE algorithm can be found in ref. [8].)

Such a policy-only learning process can be illustrated in Figure 2 in a quite general setting for seeking an optimal policy. Suppose an agent changes the policy at state i so that transitional probability from i to j (denoted by p_{ij}) increases by a small amount ∇p_{ij}, whereas transitional probability from i to k (denoted by p_{ik}) decreases by the same amount. Then, the agent could expect in gain "$\nabla p_{ij} h_j - \nabla p_{ij} h_k$," where h_j denotes the expected (relative) value at state j with respect to some special state i^* [cf. the second term in Equation (7)]. More elaborate treat-

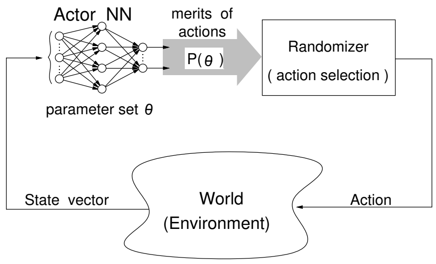


Fig. 1. *An actor neural network that implements policy-only learning. The merits of actions are used to determine probability of choosing actions.*

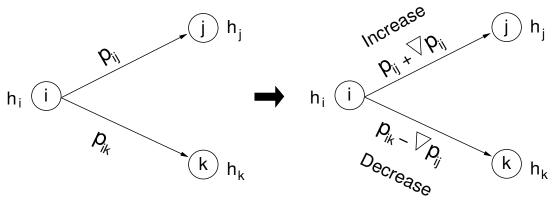


Fig. 2. *The effect of infinitesimal change in the transitional probabilities.*

* This work is greatly attributable to Professor John N. Tsitsiklis (MIT) through his course EE-298 "Neuro-Dynamic Programming," at U.C. Berkeley in Spring, 1998. Of course, any errors/mistakes are purely mine. I also wish to thank Professor Stuart E. Dreyfus (U.C. Berkeley) for his warm encouragement and valuable advice. Special thanks must go to Dr. P. Marbach (Cambridge Univ.) and Prof. Y.C. Ho (Harvard Univ.) for their useful comments via e-mail. [This paper will appear in Proc. of IEEE Int'l Joint Conf. on Neural Networks (July,1999).]

ments of this perturbation mechanism lead to the so-called *infinitesimal perturbation analysis* (IPA) [9; 10; 11], which is employed in Marbach and Tsitsiklis's policy-only algorithm. The next section details their algorithm that overcomes the two major limitations of the MENACE agent's inherent constitutions; namely, "goal-driven" and "memory-intensive" natures.

II. Marbach and Tsitsiklis's Policy-Only Learning Algorithm

This section briefly introduces Marbach and Tsitsiklis's policy-only learning, describing how IPA serves the development of the *on-line* policy-only algorithm with *implicit* estimates of relative values or relative Q-factors. Details can be found in ref. [1; 2].

A. Parameterized Markov Reward Processes

A **regenerative stochastic process** probabilistically restarts itself at regeneration points. From this viewpoint, we consider a discrete-time N(finite)-state Markov chain to be controlled by a parameter vector $\boldsymbol{\theta}$, assuming the parameterized Markov chain has a **recurrent state** i^* (at a regeneration point). When we consider a simulated sample path i_n of the Markov chain, the mth *renewal* (or *regenerative*) *cycle* can be represented as a state sequence $i_{t_m}, i_{t_m+1}, \ldots, i_{t_{m+1}}$, in which t_m denotes the time of the mth visit to the recurrent state i^*. We assume the renewal cycle lengths $T_m (= t_{m+1} - t_m)$ are *i.i.d* random variables with a finite mean $E_{\boldsymbol{\theta}}[T]$.

To evaluate different policies, we use the average cost criteria, which can be expressed, due to the **renewal reward theorem** [12; 13], as

$$\begin{aligned} \lambda(\boldsymbol{\theta}) &= \textstyle\sum_{i=0}^{N} \pi_i(\boldsymbol{\theta}) g_i(\boldsymbol{\theta}), \\ &= \lim_{m\to\infty} \frac{1}{t_m} E_{\boldsymbol{\theta}} \left[\textstyle\sum_{k=0}^{t_m} g_{i_k}(\boldsymbol{\theta}) \right], \\ &= \frac{E_{\boldsymbol{\theta}}\left[\sum_{k=t_m}^{t_{m+1}-1} g_{i_k}(\boldsymbol{\theta})\right]}{E_{\boldsymbol{\theta}}[t_{m+1}-t_m]}, \end{aligned} \qquad (1)$$

where $g_i(\boldsymbol{\theta})$ and $\pi_i(\boldsymbol{\theta})$ denote the expected cost per stage and the steady state probabilities (for any state i), respectively; the steady state probabilities $\pi_i(\boldsymbol{\theta})$ satisfy the balance equations:

$$\begin{aligned} \textstyle\sum_{i=0}^{N} \pi_i(\boldsymbol{\theta}) p_{ij}(\boldsymbol{\theta}) &= \pi_j(\boldsymbol{\theta}), \quad j = 1, 2, \ldots, N-1, \\ \textstyle\sum_{i=0}^{N} \pi_i(\boldsymbol{\theta}) &= 1. \end{aligned} \qquad (2)$$

Note that $E_{\boldsymbol{\theta}}$ implies that the expectation is taken with respect to the distribution of the Markov chain with transition probability $p_{ij}(\boldsymbol{\theta})$.

For our convenience, we use the **relative cost** function $h_i(\boldsymbol{\theta})$ for any $\boldsymbol{\theta}$ and state i as

$$h_i(\boldsymbol{\theta}) = E_{\boldsymbol{\theta}} \left[\sum_{k=0}^{T-1} \{ g_{i_k}(\boldsymbol{\theta}) - \lambda(\boldsymbol{\theta}) \} \mid i_0 = i \right], \qquad (3)$$

where $T = \min\{k > 0 \mid i_k = i^*\}$; i.e., the first future time of visit to i^*. Each time the process visits the specific state i^*, we say that a **renewal** occurs, and we reset $h_{i^*}(\boldsymbol{\theta}) = 0$, since $h_i(\boldsymbol{\theta})$ $(i \neq i^*)$ expresses a relative cost of starting the *renewal reward process* in state i with respect to the recurrent state i^*. The relative cost $h_i(\boldsymbol{\theta})$ $(i \neq i^*)$ can be obtained by solving the following linear simultaneous equations [4]:

$$h_i(\boldsymbol{\theta}) = g_i(\boldsymbol{\theta}) - \lambda(\boldsymbol{\theta}) + \sum_j p_{ij}(\boldsymbol{\theta}) h_j(\boldsymbol{\theta}). \qquad (4)$$

This is related to the average-cost Bellman's optimality equation for a given fixed policy [4; 5].

B. IPA-based Derivative Estimates

Infinitesimal Perturbation Analysis (IPA) [10; 11; 14] is a *technique for obtaining estimates of derivatives of performance with respect to parameters from sample paths or simulation.*

Recall that our objective is to minimize the average cost function $\lambda(\boldsymbol{\theta})$ by adjusting parameters $(\boldsymbol{\theta})$ of policy in a direction $\mathbf{d}$ iteratively:

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \eta_k \mathbf{d}_k \quad (k = 1, 2, 3, \ldots), \qquad (5)$$

where k denotes the current iteration number, and η is some positive step size. The $\boldsymbol{\theta}_k$ is intended to converge to a (local) minimum $\boldsymbol{\theta}^*$. When a direction $\mathbf{d}$ is set equal to the negative **gradient** [i.e., $-\nabla\lambda(\boldsymbol{\theta})$]; that is, when our objective function is directly differentiable with respect to $\boldsymbol{\theta}$, we can resort to gradient-based algorithms in a straightforward fashion. However, under a situation where an estimated average cost $\hat{\lambda}$ is only available [i.e., $\hat{\lambda} \neq \lambda(\boldsymbol{\theta})$], we need to *estimate* $\nabla\lambda(\boldsymbol{\theta})$ in place of $\mathbf{d}_k$. The theory of IPA allows us to obtain an estimate $\mathbf{d}_k$ of $\nabla\lambda(\boldsymbol{\theta})$ by simulating *regenerative processes*.

The gradient of the average cost $\lambda(\boldsymbol{\theta})$ [in Equation (1)] with respect to $\boldsymbol{\theta}$ is given by

$$\nabla\lambda(\boldsymbol{\theta}) = \sum_i \pi_i(\boldsymbol{\theta}) \nabla g_i(\boldsymbol{\theta}) + B(\boldsymbol{\theta}), \qquad (6)$$

where $B(\boldsymbol{\theta}) \stackrel{\text{def}}{=} \sum_i \nabla\pi_i(\boldsymbol{\theta}) g_i(\boldsymbol{\theta})$. Using the relative cost $h_i(\boldsymbol{\theta})$, the gradient can be expressed as

$$\begin{aligned} \nabla\lambda(\boldsymbol{\theta}) &= \textstyle\sum_i \pi_i(\boldsymbol{\theta}) \left[\nabla g_i(\boldsymbol{\theta}) + \sum_j \nabla p_{ij}(\boldsymbol{\theta}) h_j(\boldsymbol{\theta}) \right] \\ &= \textstyle\sum_i \pi_i(\boldsymbol{\theta}) \left[\nabla g_i(\boldsymbol{\theta}) + \sum_j p_{ij}(\boldsymbol{\theta}) \left\{ \frac{\nabla p_{ij}(\boldsymbol{\theta})}{p_{ij}(\boldsymbol{\theta})} h_j(\boldsymbol{\theta}) \right\} \right]. \end{aligned} \qquad (7)$$

This is the *essence* of the algorithm; a variety of estimates of $\nabla\lambda(\boldsymbol{\theta})$ were described in different contexts [15; 14; 16]. Notice that this expression does not involve any $\nabla\pi_i(\boldsymbol{\theta})$ terms unlike Equation (6), and thus simulation-based estimate of $\nabla\lambda(\boldsymbol{\theta})$ is obtainable [2; 1; 15]. Here is some algebra required to derive the "key" term $\sum_i \sum_j \pi_i(\boldsymbol{\theta}) \nabla p_{ij}(\boldsymbol{\theta}) h_j(\boldsymbol{\theta})$ in Equation (7) from $B(\boldsymbol{\theta})$ in Equation (6):

$$\begin{aligned}
&B(\theta) = \textstyle\sum_i \nabla \pi_i(\theta) g_i(\theta) \\
&= \textstyle\sum_i \nabla \pi_i(\theta) \left[g_i(\theta) - \lambda(\theta)\right], \text{ due to } \sum_i \nabla \pi_i(\theta) = 0 \text{ for } \forall \theta, \\
&= \textstyle\sum_i \nabla \pi_i(\theta) \left[h_i(\theta) - \sum_j p_{ij}(\theta) h_j(\theta)\right], \text{ due to Eq. (4)}, \\
&= \textstyle\sum_i \nabla \pi_i(\theta) h_i(\theta) - \sum_i \sum_j \left\{\nabla \pi_i(\theta) p_{ij}(\theta)\right\} h_j(\theta), \\
&= \textstyle\sum_i \nabla \pi_i(\theta) h_i(\theta) - \sum_i \sum_j \left\{\nabla \left[\pi_i(\theta) p_{ij}(\theta)\right]\right. \\
&\quad \left. - \pi_i(\theta)\, \nabla p_{ij}(\theta)\right\} h_j(\theta), \text{ due to the product rule}, \\
&= \textstyle\sum_i \nabla \pi_i(\theta) h_i(\theta) - \sum_j \nabla \left[\sum_i \pi_i(\theta) p_{ij}(\theta)\right] h_j(\theta) \\
&\quad + \textstyle\sum_i \sum_j \pi_i(\theta) \nabla p_{ij}(\theta) h_j(\theta), \\
&= \textstyle\sum_i \nabla \pi_i(\theta) h_i(\theta) - \sum_j \nabla \left[\pi_j(\theta)\right] h_j(\theta) \\
&\quad + \textstyle\sum_i \sum_j \pi_i(\theta) \nabla p_{ij}(\theta) h_j(\theta), \text{ due to Eq. (2)}.
\end{aligned}$$

Now, the first two terms are cancelled out.

With a current estimate of average cost $\hat{\lambda}$, an estimate of $h_i(\boldsymbol{\theta})$ in Equation (3), denoted by $\hat{h}_i(\boldsymbol{\theta}, \hat{\lambda})$, can be written, due to Equation (1), as

$$\hat{h}_i(\theta, \hat{\lambda}) = \begin{cases} 0 & \text{if } n = t_m \text{ (i.e., } i_n = i^*) \\ \sum_{k=n}^{t_{m+1}-1} \left\{g_{i_k}(\theta) - \hat{\lambda}\right\} & \text{if } t_m < n \le t_{m+1} - 1. \\ = E_\theta[T] \left\{\lambda(\theta) - \hat{\lambda}\right\} & \end{cases} \tag{8}$$

This shows that the expected update direction for $\hat{\lambda}$ is in the direction of making $\hat{\lambda}$ closer to $\lambda(\boldsymbol{\theta})$. By accumulating the aforementioned estimates over a renewal cycle, an estimate of $\nabla \lambda(\boldsymbol{\theta})$ becomes

$$F_m(\boldsymbol{\theta}, \hat{\lambda}) = \sum_{n=t_m}^{t_{m+1}-1} \left[\nabla g_{i_n}(\boldsymbol{\theta}) + \hat{h}_{i_n}(\boldsymbol{\theta}, \hat{\lambda}) \frac{\nabla p_{i_{n-1} i_n}}{p_{i_{n-1} i_n}}\right], \tag{9}$$

which is related to likelihood ratio methods [17].

C. On-line Update for Markov Reward Processes

To develop the on-line update formula, we rewrite Equation (9), using $\hat{h}_{i_{t_m}}(\boldsymbol{\theta}, \hat{\lambda}) = 0$ and Eq. (8), as

$$\begin{aligned}
&F_m(\theta, \hat{\lambda}) \\
&= \textstyle\sum_{n=t_m}^{t_{m+1}-1} \nabla g_{i_n}(\theta) + \sum_{n=t_m+1}^{t_{m+1}-1} \hat{h}_{i_n}(\theta, \hat{\lambda}) \frac{\nabla p_{i_{n-1} i_n}(\theta)}{p_{i_{n-1} i_n}(\theta)} \\
&= \nabla g_{i^*}(\theta) \\
&+ \textstyle\sum_{n=t_m+1}^{t_{m+1}-1} \left[\nabla g_{i_n}(\theta) + \frac{\nabla p_{i_{n-1} i_n}(\theta)}{p_{i_{n-1} i_n}(\theta)} \sum_{k=n}^{t_{m+1}-1} \left\{g_{i_k}(\theta) - \hat{\lambda}\right\}\right] \\
&= \nabla g_{i^*}(\theta) \\
&+ \textstyle\sum_{k=t_m+1}^{t_{m+1}-1} \left[\nabla g_{i_k}(\theta) + \left\{g_{i_k}(\theta) - \hat{\lambda}\right\} \sum_{n=t_m+1}^{k} \frac{\nabla p_{i_{n-1} i_n}(\theta)}{p_{i_{n-1} i_n}(\theta)}\right] \\
&= \nabla g_{i^*}(\theta) + \textstyle\sum_{k=t_m+1}^{t_{m+1}-1} \left[\nabla g_{i_k}(\theta) + \left\{g_{i_k}(\theta) - \hat{\lambda}\right\} \mathbf{z}_k\right],
\end{aligned} \tag{10}$$

where $\mathbf{z}_k$ is a vector of the same dimension as $\boldsymbol{\theta}$:

$$\mathbf{z}_k = \sum_{n=t_m+1}^{k} \frac{\nabla p_{i_{n-1} i_n}(\boldsymbol{\theta})}{p_{i_{n-1} i_n}(\boldsymbol{\theta})}, \quad k = t_m + 1, \ldots, t_{m+1} - 1.$$

This vector $\mathbf{z}$ plays a role as **eligibility trace** in TD learning [3].

The following on-line update formulae that have a form of Equation (5) offer a way to tune policy incrementally:

$$\begin{aligned}
\boldsymbol{\theta}_{k+1} &= \boldsymbol{\theta}_k - \eta_k \left[\nabla g_{i_k}(\boldsymbol{\theta}_k) + \left\{g_{i_k}(\boldsymbol{\theta}_k) - \hat{\lambda}_k\right\} \mathbf{z}_k\right], \\
\hat{\lambda}_{k+1} &= \hat{\lambda}_k + \eta_k \left[g_{i_k}(\boldsymbol{\theta}_k) - \hat{\lambda}_k\right].
\end{aligned} \tag{11}$$

The latter $\hat{\lambda}$ updating formula [cf. Equation (8)] is related to the concept of **reinforcement comparison** [18]. After updating $\boldsymbol{\theta}$ and $\hat{\lambda}$, we simulate a transition to the next state i_{k+1} according to $p_{i_k i_{k+1}}(\boldsymbol{\theta}_{k+1})$, and then update $\mathbf{z}$ by the following:

$$\mathbf{z}_{k+1} = \begin{cases} 0 & \text{if } i_{k+1} = i^*, \\ \beta \mathbf{z}_k + \frac{\nabla p_{i_k i_{k+1}}(\boldsymbol{\theta}_{k+1})}{p_{i_k i_{k+1}}(\boldsymbol{\theta}_{k+1})} & \text{otherwise}, \end{cases} \tag{12}$$

where β is a recency parameter ranging from 0 to 1. Here, if $p(\boldsymbol{\theta})$ is constructed by an *actor neural network*, as illustrated in Figure 1, then $\nabla p(\boldsymbol{\theta})$ can by obtained in conjunction with the well-known **backpropagation**.

D. On-line Update for Markov Decision Processes

For solving the stochastic Markov decision problems, we employ a randomized policy associated with the *probability of choosing action a at state i,* denoted by $\mu_a(i, \boldsymbol{\theta})$, which satisfies $\sum_a \mu_a(i, \boldsymbol{\theta}) = 1$. The resulting transitional probabilities and the expected immediate costs can be expressed as:

$$\begin{aligned}
&p_{ij}(\boldsymbol{\theta}) = \textstyle\sum_a \mu_a(i, \boldsymbol{\theta}) p_{ij}(a), \\
&g_i(\boldsymbol{\theta}) = \textstyle\sum_a \mu_a(i, \boldsymbol{\theta}) g_i(a) = \sum_a \mu_a(i, \boldsymbol{\theta}) \sum_j p_{ij}(a) c_{ij}(a),
\end{aligned}$$

where $c_{ij}(a)$ is a transitional cost. Taking derivatives and plugging them into Equation (7) yields:

$$\nabla \lambda(\boldsymbol{\theta}) = \sum_i \sum_a \pi_i(\boldsymbol{\theta}) \mu_a(i, \boldsymbol{\theta}) Q_{i,a}(\boldsymbol{\theta}) \frac{\nabla \mu_a(i, \boldsymbol{\theta})}{\mu_a(i, \boldsymbol{\theta})},$$

where $Q_{i,a}(\boldsymbol{\theta})$ is a **relative Q-factor** [cf. Equations (3) and (4)] given by:

$$\begin{aligned}
&Q_{i,a}(\boldsymbol{\theta}) = g_i(a) - \lambda(\boldsymbol{\theta}) + \textstyle\sum_j p_{ij}(a) h_j(\boldsymbol{\theta}), \\
&\quad = E_{\boldsymbol{\theta}} \left[\textstyle\sum_{k=0}^{T-1} \left\{g_{i_k}(a_k) - \lambda(\boldsymbol{\theta})\right\} \mid i_0 = i, a_0 = a\right].
\end{aligned}$$

The resultant on-line update formulae for the stochastic action problems are given by

$$\begin{aligned}
\boldsymbol{\theta}_{k+1} &= \boldsymbol{\theta}_k - \eta_k \left[\nabla g_{i_k}(\boldsymbol{\theta}_k) + \left\{g_{i_k}(\boldsymbol{\theta}_k) - \hat{\lambda}_k\right\} \mathbf{w}_k\right], \\
\hat{\lambda}_{k+1} &= \hat{\lambda}_k + \eta_k \left[g_{i_k}(\boldsymbol{\theta}_k) - \hat{\lambda}_k\right],
\end{aligned} \tag{13}$$

where ∇g is, due to $\sum_a \nabla \mu_a(i, \boldsymbol{\theta}) = 0$, given by

$$\begin{aligned}
\nabla g_{i_k}(\boldsymbol{\theta}_k) &= \textstyle\sum_a \nabla \mu_a(i_k, \boldsymbol{\theta}) \left[g_{i_k}(a) - \hat{\lambda}\right], \\
&= \textstyle\sum_a \nabla \mu_a(i_k, \boldsymbol{\theta}) \left[\sum_j p_{i_k j}(a) c_{i_k j}(a) - \hat{\lambda}\right].
\end{aligned}$$

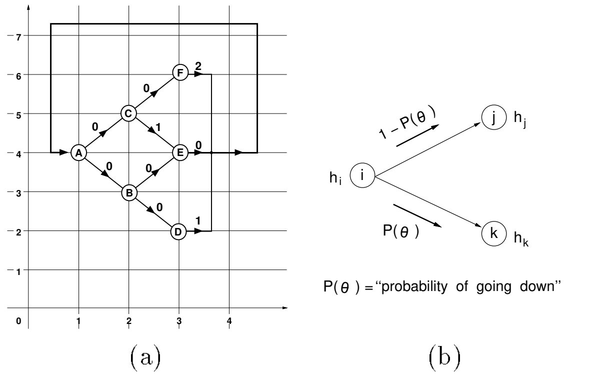


Fig. 3. *(a) State transition diagram in the x-y coordinates system. (b) The agent's "action selection" based on the output of the actor neural network that produces the probability of action "going down."*

Table 1. *Six experimental conditions (a) through (f) for solving a deterministic minimum-cost path problem, as illustrated in Figure 3(a). The column "Para #" implies "# of parameters in $p(\theta)$," and the column "η" signifies the "step size for θ-update" in Equation (11). The results are shown in Figure 4.*

Exp.	Architecture of $p(\theta)$	Para #	η
(a)	$p(\theta) = \theta$	1	0.0004
(b)	$p(\theta) = \frac{1}{1+exp(\theta)}$	1	0.01
(c)	actor neural network $(2 \times 1 \times 1)$ with only vertex A $(1,4)$ as input	5	0.01
(d)	$p_i(\theta_i) = \theta_i \ (i = 1, 2, 3)$	3	0.0004
(e)	$p_i(\theta_i) = \frac{1}{1+exp(\theta_i)} \ (i = 1, 2, 3)$	3	0.01
(f)	actor neural network $(2 \times 1 \times 1)$ with different coordinates as input	5	0.01

The eligibility trace vector $\mathbf{w}$ can be updated by

$$\mathbf{w}_{k+1} = \begin{cases} 0 & \text{if } i_{k+1} = i^*, \\ \beta \mathbf{w}_k + \frac{\nabla \mu_a(i_k, \boldsymbol{\theta}_{k+1})}{\mu_a(i_k, \boldsymbol{\theta}_{k+1})} & \text{otherwise.} \end{cases} \tag{14}$$

In this case, $\mu_a(i, \boldsymbol{\theta})$ can be constructed by an *actor neural network*, and $\nabla \mu_a(., \boldsymbol{\theta})$ is obtainable from backpropagation.

As far as the policy changes in an on-line fashion, $\hat{\lambda}$ becomes a *biased* (not unbiased) estimate of the true average cost; hence, the mathematical analysis is complicated but the convergence of the algorithm has been proved [1; 2].

III. Experimentation and Discussions

We consider small-scale two-action problems, in which the agent needs to make an action of either going down or up; we thus employ a single-output actor neural network; its output can be interpreted as P_{down}: i.e., *probability of action d (going down)* [see Figure 3(b)]. In this implementation, the input state vector may not necessarily require the next state j but the current state i alone. For the input representation, the two-dimensional coordinate state vectors are used; for instance, vertex A is represented as $(1, 4)$, as shown in Figure 3(a).

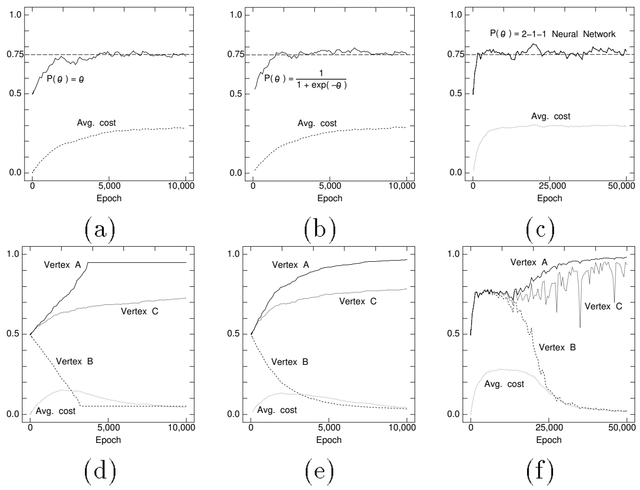


Fig. 4. *The results for the minimum cost path problem depicted in Figure 3(a). The six experiments were conducted: (a) a single parameter using a linear identity function; (b) a single parameter using a sigmoidal logistic function; (c) a neural network $(2 \times 1 \times 1)$ with five parameters; (d) three parameters using linear identity functions; (e) three parameters using sigmoidal logistic functions; (f) a neural network $(2 \times 1 \times 1)$ with five parameters. (See Table 1.) Here, "epoch" means "transition."*

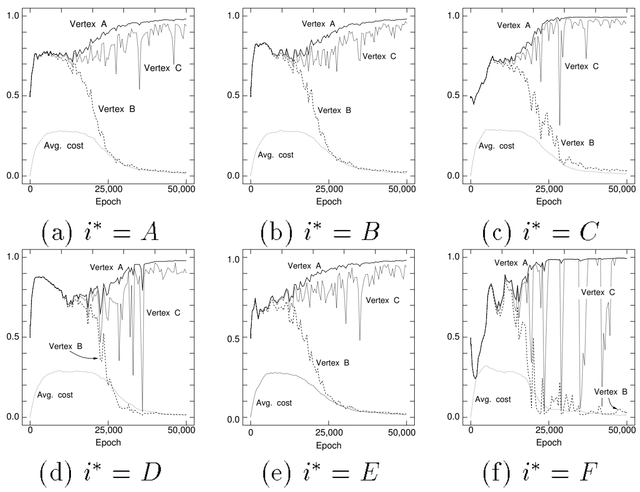


Fig. 5. *Effects of choosing different states as a recurrent state i^* for solving the minimum-cost path problem in Figure 3(a).*

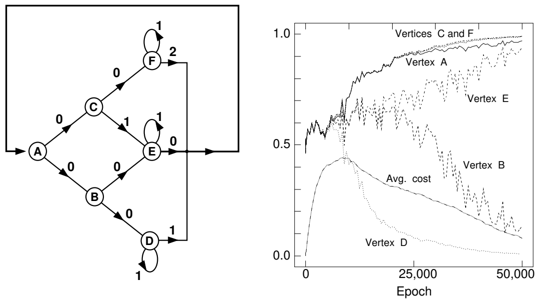
A. Deterministic Minimum-Cost Path Problems

In our first experiment, we used a three-stage shortest path problem depicted in Figure 3(a), in which the optimal trajectory is $A \rightarrow B \rightarrow E \rightarrow A$. To investigate average-cost algorithms, we purposely introduce a feed-back loop from vertices D, E, and F to vertex A, as illustrated in Figure 3(a), which can be viewed as an *irreducible periodic Markov chain*. Notice that at vertices D, E, and F, the agent proceeds back to vertex A with probability 1; hence, no action is made at those three vertices. The learning algorithm was implemented in six cases, summarized in Table 1, using the three different functions as $p(\boldsymbol{\theta})$: "linear identity function," "sigmoidal logistic function," "actor network ($2 \times 1 \times 1$)." Note that the $2 \times 1 \times 1$ actor network has totally five tunable parameters (including weights connected to bias units) with sigmoidal logistic neuron functions. Figure 4 illustrates the learning curves for the six cases; for neural network training, 50,000 transitions were simulated. In these experiments, $i^* =$ vertex A and $\beta = 1$ in Equation (12).

For Experiment (a), we can readily compute the optimal average cost analytically because $\lambda(\boldsymbol{\theta})$ has a simple quadratic form [cf. Equation (1)]: $\lambda(\theta) = \frac{1}{3}\left[\theta^2 + 2(1-\theta)^2 + (1-\theta)\theta\right]$. Hence, "$\nabla\lambda(\theta) = 0$" gives $\theta^* = 0.75$, and thus $\lambda^* = 0.2917$; the outputs were converging to these values, as clearly seen in Figures 4(a), (b), and (c). In Experiment (c), the input stimuli for the actor network was always the same as A $(1, 4)$ at any state, unlike Experiment (f); hence, its output was expected to be the same as the single parameter cases (a) and (b), whereas the agent in Experiment (f) did make different actions at each state in spite of the same neural network structure. In other words, the **input state representation** for neural networks is crucial.

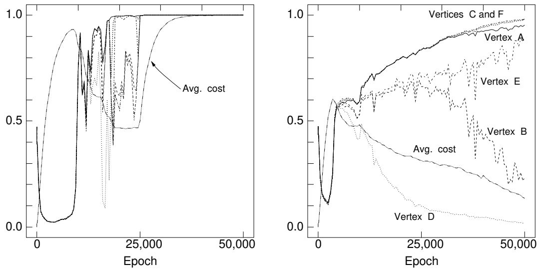
As shown in Figures 4(d) and (e), the three values (associated with vertices A,B,C) evolved in different routes from the scratch. In Figure 4(f), on the other hand, those three values evolved in the same route at the beginning of learning, presumably because this actor neural network was a multilayer perceptron. For Experiment (d), we set the upper (0.95) and lower (0.05) limits for $p(\boldsymbol{\theta})$ to keep minimal amount of **exploration**. (Similarly, in Experiments (c) and (f), when the linear identity function was employed as the neuron function at the output layer, the learning speed was faster; yet, we needed to set the upper and lower bounds.)

We next investigate the effects of choosing different states as a recurrent state i^*. There are six choices for i^* in this problem setting. Figure 5 shows a resultant comparison among those six choices. In this deterministic action problem, vertices C, D, and F were rarely visited as decisions became more optimal. Hence, the accumulated values in eligibility trace vectors $\mathbf{z}$ (or $\mathbf{w}$) in Equation (11) [or (13)] may change learning curves dramatically, even for this small problem. In fact, it appears that three values (associated with vertices A,B,C) in Figures 5 (c), (d), and (f), evolved in the same route for a longer time than those in Figures 5(a), (b), and (e). Larger spikes of the value associated with vertex C are also another evidence of the influence of eligibility trace. Furthermore, in Figures 5(c) and (f), all the three values



(a) (b)

Fig. 6. *A minimum-cost path with self-loops problem: (a) the transition diagram, where "up" now means "choose left-hand path when looking forward from vertex," and (b) a sample learning curve obtained by a $2 \times 1 \times 1$ actor network that has totally five adjustable parameters, when $i^* = A$. The recency parameter β was set equal to 1. "Epoch" means "transition."*



(a) $\beta = 1.0$ ($i^* = F$) (b) $\beta = 0.85$ ($i^* = F$)

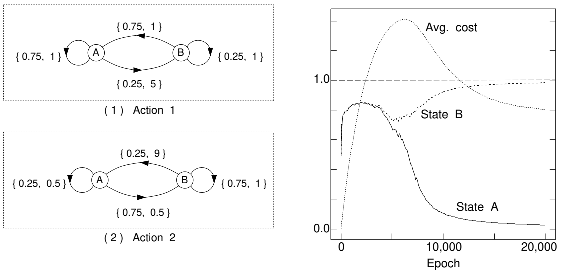
Fig. 7. *Effects of recency parameter β in the eligibility trace update, when i^*=F: (a) $\beta = 1.0$, and (b) $\beta = 0.85$.*

associated with vertices A, B, and C, went down initially and then went up to about 0.8, whereas in Figures 5(a),(b), (d), and (e), the learning curves started rising up initially. This may be because the optimal action at vertex A is "going down" but vertices C and F are not located in that direction. In particular, this tendency was intensified when $i^* = F$; see also Figure 7.

B. Deterministic Self-Looped Minimum-Cost Path

We next introduce "unit-cost self-loops" at vertices D, E, and F, as illustrated in Figure 6(a), which can be viewed as an irreducible *aperiodic* Markov chain. Notice that the optimal decision at vertex D is "going up," whereas the optimal decision at vertices E and F is "going down" to avoid the unit-cost self-loops. The optimal path ($A \to B \to E \to A$) in Figure 6(a) is the same as before in Figure 3(a). The $2 \times 1 \times 1$ actor-network agent did learn the optimal trajectory, as shown in Figure 6(b), where $i^* = A$.

In order to see the effects of the **recency parameter** β in the eligibility trace update formulae (12) and (14), we purposely chose F as i^* since we observed a strange behavior when $i^* = F$ in Figure 5(f). Figure 7 shows that a decreased β (less than 1.0) has a positive impact on reducing oscillatory learning be-



(a) Transition diagrams. (b) A learning curve.

Fig. 8. *A simple two-state two-action Markov decision problem; (a) Transition diagrams for each of two actions 1 and 2. A pair on each arc denotes {probability, cost}. (b) A learning curve obtained by an actor network ($2 \times 1 \times 1$).*

haviors; when $\beta = 1.0$, the agent failed to learn.

In this problem, the agent needs to make a decision at all six vertices. If the lookup-table representation is used to construct a policy, then *six* parameters are necessary. Here, an actor network has *only five* parameters, although six more parameters for $\mathbf{z}$ and $\hat{\lambda}$ are necessary to update $\boldsymbol{\theta}$. The approximation capacity of neural network function approximators will become much more important in practical large state space problems.

C. A Stochastic Decision Problem

In the foregoing, we have solved the *deterministic* action problems based on Equations (11) and (12). We now consider an application of Equations (13) and (14) to a simple *stochastic* action problem. Its Markov chains are illustrated in Figure 8(a); the associated transition probabilities and costs are shown on each associated arc in the form {probability, cost}.

We can solve this small-scale problem analytically by application of the well-known Howard's exact policy iteration technique for the average cost criteria [4]. The obtained results gave the optimal policy "$\mu^*(A) =$ Action 2 and $\mu^*(B) =$ Action 1," and the associated steady-state probabilities: $q_A^* = q_B^* = 0.5$. Thus, the optimal average cost can be computed as $\lambda^* = q_A^* g(A, \mu^*(A)) + q_B^* g(B, \mu^*(B)) = 0.5 \cdot 0.5 + 0.5 \cdot 1 = 0.75$. The simulation result obtained by the $2 \times 1 \times 1$ actor neural network matches those analytical calculations, as presented in Figure 8(b). Here, the input representations for states A and B were the same as coordinates in Figure 3(a).

IV. Concluding Remarks

Since the policy-only algorithm does not need to approximate optimal cost-to-go functions *explicitly*, this learning scheme with actor neural networks may work significantly better for some applications than *value-iteration*-based approaches (e.g., Q-learning). In this sense, this algorithm has a great potential.

Equations (11) and (13) require models to compute g and ∇g unlike Q-learning; for comparison purposes, this algorithm should be implemented in a **totally model-free** fashion by replacing the expected values with a single sample so as to see how much profound influence noise terms can have on the learning process.

Moreover, it is of our great interest to apply the policy-only algorithm to solving more practical problems; such as control of queueing systems. Marbach and Tsitsiklis's algorithm is still quite conceptual, but further computational experiments may clarify practical advantages and suitable applications.

References

[1] P. Marbach and J. N. Tsitsiklis. Simulation-based optimization of Markov reward processes. *IEEE Transactions on Automatic Control*, February 1998. (submitted).

[2] Peter Marbach. *Simulation-based optimization of Markov decision processes.* PhD thesis, EECS Department, Massachusetts Institute of Technology, Cambridge, MA 02139, September 1998.

[3] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction.* MIT Press, Cambridge, MA., 1998.

[4] D. P. Bertsekas. *Dynamic Programming and Optimal Control (vol.1 and 2).* Athena Scientific, Belmont, Massachusetts, 1995.

[5] D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-Dynamic Programming.* Athena Scientific, Belmont, MA, 1996.

[6] R.E Bellman and S.E. Dreyfus. *Applied dynamic programming.* Princeton University Press, 1962.

[7] D. Michie. Trial and error. *Penguin Science Survey*, 2, 1961.

[8] E. 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.

[9] Y.C. Ho and X.R. Cao. Perturbation analysis and optimization of queueing networks. *Journal of Optimization: Theory and Applications*, 40:559–582, 1983.

[10] P. Glasserman. *Gradient Estimation via Perturbation Analysis.* Kluwer Academic Publisher, Boston,MA, 1990.

[11] Y.C. Ho and X.R. Cao. *Perturbation Analysis of Discrete Event Systems.* Kluwer Academic, Boston,MA, 1991.

[12] Henk C. Tijms. *Stochastic models : an algorithmic approach.* John Wiley & Sons Inc., New York, 1994.

[13] Sheldon M. Ross. *Stochastic processes.* Wiley, New York, 1996. (2nd Ed.).

[14] M. C. Fu and J. Q. Hu. Smoothed perturbation analysis derivative estimation for Markov chains. *Operations Research Letters*, 15:241–251, 1994.

[15] X.R. Cao and H.F. Chen. Perturbation realization, potentials, and sensitivity analysis of Markov processes. *IEEE Transactions on Automatic Control*, AC-42(10):1382–1393, October 1997.

[16] T. Jaakkola, S. P. Singh, and M. I. Jordan. Reinforcement learning algorithms for partially observable Markov decision. In *Advances in Neural Information Processing Systems 7*, pages 345–352, San Mateo, CA, 1995. Morgan Kaufmann.

[17] Peter W. Glynn. Likelihood ratio gradient estimation for stochastic systems. *Communications of the ACM*, 33(10):75–84, 1990.

[18] P. Dayan. Reinforcement comparison. In D. S. Touretzky, J. L. Elman, T. J. Sejnowski, and G. E. Hinton, editors, *Proceedings of the 1990 Connectionist Models Summer School*, pages 45–51, San Mateo, CA., 1990. Morgan Kaufmann.