AN RBF BASED NEURO-DYNAMIC APPROACH FOR THE CONTROL OF STOCHASTIC DYNAMIC SYSTEMS

Panagiotis K. Patrinos, Haralambos Sarimveis

School of Chemical Engineering, National Technical University of Athens, 9, Heroon Polytechniou str., Zografou Campus, Athens 15780, Greece
patrinos@central.ntua.gr, hsarimv@central.ntua.gr

Abstract: This paper presents a neuro-dynamic programming methodology for the control of markov decision processes. The proposed method can be considered as a variant of the optimistic policy iteration, where radial basis function (RBF) networks are employed as a compact representation of the cost-to-go function and the λ-LSPE is used for policy evaluation. We also emphasize the reformulation of the Bellman equation around the post-decision state in order to circumvent the calculation of the expectation. The proposed algorithm is applied to a retailer-inventory management problem. *Copyright © 2005 IFAC*

Keywords: Markov decision processes, Radial base function networks, uncertain dynamic systems, optimal control

1. INTRODUCTION

1.1 Overview

In this paper we deal with the control of discrete-time stochastic dynamic systems using neuro-dynamic programming methods. We consider dynamic systems which evolve through time according to:

$$x_{t+1} = f(x_t, u_t, \omega_t) \tag{1}$$

where $x_t \in \mathbb{X}$ is the state of the system at time *t*, $u_t \in \mathbb{U}$ is the control action and ω_t takes values in a finite set $\mathbb{W}$ with given probability distribution $p_t\left(\cdot \middle| x_t, u_t\right)$ depending on x_t and u_t. The state and control sets, $\mathbb{X}$, $\mathbb{U}$ respectively are assumed to be finite as well. With each state-control pair and disturbance we associate a bounded cost function $g : \mathbb{X} \times \mathbb{U} \times \mathbb{W} \mapsto \mathbb{R}$. We define an admissible control policy π as a sequence of functions μ_t that map states on control actions that are taken at each time instance *t*. The objective of the control problem is to find an optimal policy, that is, an admissible policy $\pi = \{\mu_0, \mu_1, ...\}$ which minimizes the expected infinite horizon discounted cost:

$$J^{\pi}(x_0) = \mathbb{E}\left[\sum_{t=0}^{\infty} a^t g\left(x_t, \mu_t(x_t), \omega_t\right)\right] \tag{2}$$

where $J^{\pi}(x_0)$ is the cost-to-go of policy π starting from state x_0 and $a \in (0,1)$ is the discount factor.

We define the optimal cost-to-go function J^* as:

$$J^*(x) = \min_{\pi} J^{\pi}(x) \;\; \forall x \in \mathbb{X} \tag{3}$$

The above formulation is known as the Markov decision problem and has been studied thoroughly by researchers in the control and operations research community. The problem originated from the seminal work of Bellman (1957), where a new theoretic and algorithmic framework named *dynamic programming*

was proposed for addressing optimal control problems

1.2 Dynamic Programming

The optimal cost-to-go function is given by the solution of the *Bellman equation*:

$$J^*(x) = \min_{u\in\mathbb{U}} \mathbb{E}\left[g(x,u,\omega) + aJ^*(f(x,u,\omega)) \middle| x,u \right], \forall x \in \mathbb{X} \tag{4}$$

where $\mathbb{E}\left[\cdot \middle| x,u\right]$ denotes the expected value with respect to the probability distribution of ω. Once the optimal cost-to-go value is determined for each state, the optimal control policy is defined as follows:

$$\mu^*(x) = \arg\min_{u\in\mathbb{U}} \mathbb{E}\left[g(x,u,\omega) + aJ^*(f(x,u,\omega)) \middle| x,u \right] \tag{5}$$

There exist two central algorithmic techniques for calculating the optimal cost-to-go function, that is, value iteration and policy iteration.

Value iteration starts with an *n*-dimensional vector *J*, where *n* is the cardinality of set X and successively computes approximations of the optimal cost-to-go value for each $x \in \mathbb{X}$ by using the Bellman equation. Thus, at iteration *k* the method performs updates of the optimal cost-to-go value for each $x \in \mathbb{X}$ as follows:

$$J^{k+1}(x) = \min_{u\in\mathbb{U}} \mathbb{E}\left[g(x,u,\omega) + aJ^k(f(x,u,\omega)) \middle| x,u \right], \forall x \in \mathbb{X} \tag{6}$$

The method terminates when the values $J^k(x)$ converge for every $x \in \mathbb{X}$. It is proven that the values at which the successive approximations converge are the optimal cost-to-go values $J^*(x)$ for each $x \in \mathbb{X}$. The method generally requires an infinite number of iterations.

Policy iteration involves a two-step procedure comprised of policy evaluation and policy improvement. The method starts with a specific stationary policy π^0, that is, a policy of the form $\{\mu^0, \mu^0, ...\}$. At each iteration *k* the method retains a stationary policy π^k and the corresponding cost-to-go values denoted by J^{μ^k} are calculated for each $x \in \mathbb{X}$ from the solution of the system of equations:

$$J^{\mu^k}(x) = \mathbb{E}\left[g(x,u,\omega) + aJ^{\mu^k}(f(x,u,\omega)) \middle| x,u \right], \quad \forall x \in \mathbb{X} \tag{7}$$

Once $J^{\mu_k}(x)$ has been calculated for every state, the policy improvement step takes place:

$$\mu^{k+1}(x) = \arg\min_{u\in\mathbb{U}} \mathbb{E}\left[g(x,u,\omega) + aJ^{\mu_k}(f(x,u,\omega)) \middle| x,u \right] \tag{8}$$

The above procedure is repeated with μ^{k+1} in place of μ^k unless $J^{\mu^{k+1}}(x) = J^{\mu^k}(x), \quad \forall x \in \mathbb{X}$. The main characteristic of policy iteration is that the sequence of stationary policies generated by the method is improving.

There exist numerous other algorithms that involve variants or combinations of the above two methods, such as the *modified policy iteration* (Puterman, 1994) method.

1.3 Neuro-Dynamic Programming

Dynamic programming techniques are inapplicable in practice because they suffer from the well known *curse of dimensionality* of the state space, i.e. the exponential growth of the cardinality of set X with respect to the number of state variables. For this reason the computation and storage of the cost-to-go value for each state that is required by value and policy iteration, is prohibitive for even medium – scale systems. One more difficulty that arises in classical dynamic programming is the requirement of the explicit dynamics of the system, that is, the transition probabilities of the underlying Markov model. To overcome these difficulties, a new class of simulation-based approximate dynamic programming methods has been developed. This class of suboptimal methods originated from the artificial intelligence research community under the name *reinforcement learning* (Sutton, 1988, Sutton & Barto, 1998). The methods were adopted and mathematically substantiated by the engineering community which used the name *neuro-dynamic programming* (Bertsekas, Tsitsiklis, 1996). The key points of these methods are the utilization of compact representations of the cost-to-go vectors by function approximators such as neural networks and the use of system simulators in order to obtain sample trajectories and train the approximators. Most of these methods lead to an approximate cost-to-go function $\tilde{J}(x,w)$ of the optimal cost-to-go function $J^*(x)$. The main purpose of neuro-dynamic algorithms is to compute the appropriate parameter values $w \in \mathbb{R}^k$, such that the cost-to-go approximator is as close as possible to $J^*(x)$. Once the approximate cost-to-go is determined off-line, the corresponding *greedy* policy can be implemented on-line for controlling the system in a suboptimal fashion, according to:

$$\mu(x) = \arg\min_{u\in\mathbb{U}} \mathbb{E}\left[g(x,u,\omega) + a\tilde{J}(f(x,u,\omega),w) \middle| x,u \right] \tag{9}$$

In this paper we will focus on a variant of the neuro-dynamic programming method named approximate policy iteration, which we briefly describe next:

We start with some stationary policy μ^0 and at each iteration *k* we approximately evaluate the cost of the corresponding policy in eq. (2) by linear regression, fitting the cost-to-go approximator to the results (state/cost-to-go pairs) of a theoretically infinitely

long trajectory. However, in practice we cannot carry out an infinitely long simulation. Thus, we constraint ourselves to trajectories consisting of only a finite number of N steps, and we accumulate the discounted costs of only the first $N - M$ states, considering for each one of them an M-step horizon. In this way we collect N-M pairs of state - cost-to-go values. These sampled cost-to-go values $c(x)$ are within $Ga^M/(1- a)$ of J^μ, where G is an upper bound on $|g(x,u,\omega)|$ (Bertsekas & Tsitsiklis, 1996).

Assuming that the state - cost-to-go pairs have been collected, one can solve the least-squares optimization problem in batch mode in order to determine the parameter vector w:

$$\arg\min_{w} \sum_{x\in\bar{\mathbb{X}}} \left(\tilde{J}(x,w) - c(x)\right) \quad (10)$$

Another option is to solve the least squares problem 'on-line', as the simulation progresses and new data are becoming available. The most famous of these methods is the so called *temporal difference learning* (Sutton, 1988) which is considered as a major breakthrough in neuro-dynamic programming. The method can be described as an incremental gradient-like least squares method, where the parameter values are updated upon each observation of a state transition and the associated cost. The method is based on the notion of temporal difference d_t corresponding to the transition from x_t to x_{t+1}:

$$d_t = g(x_t, u_t, \omega_t) + a\tilde{J}(x_{t+1}, w_t) - \tilde{J}(x_t, w_t) \quad (11)$$

Favorable convergence results for the method are presented in (Tsitsiklis and Van Roy, 1997) in the case of linear function approximators. Other methods proposed in the literature for approximate policy evaluation are *least-squares temporal difference learning* (LSTD) and *λ-least squares policy evaluation* (λ-LSPE) (Nedic, Bertsekas, 2003), which will be described in detail in 2.3.

After the approximate policy evaluation step has been performed, the policy improvement step follows:

$$\mu^{k+1}(x) = \arg\min_{u\in\mathbb{U}} \mathbb{E}\left[g(x,u,\omega) + a\tilde{J}^{\mu_k}\left(f(x,u,\omega)\right) \middle| x,u \right], \forall x \in \mathbb{X} \quad (12)$$

The steps of approximate policy evaluation and policy improvement are repeated in an interleaved fashion. It is worth mentioning that contrary to policy iteration, approximate policy iteration is not producing improving policies in each iteration.

There are also optimistic variants of the approximate policy iteration method, where the policy improvement step is performed after few transitions of the simulated dynamic system, or even after each transition. Although optimistic policy iteration has been proven convergent only for look-up table representation (Tsitsiklis, 2002) there is no similar result in the case of function approximators. However, this method has produced the most encouraging results in the literature of approximate dynamic programming. The most noticeable application of the method is its incorporation in the Tesauro's backgammon player (Tesauro, 1994).

2. PROPOSED METHODOLOGY

We propose the use of radial basis functions as function approximators of the cost-to-go functions within an optimistic policy iteration scheme for the control of stochastic discrete-time dynamic systems. We employ λ-LSPE for the policy evaluation step.

2.1 Radial Basis Functions

The true cost-to-go function is inherently nonlinear with respect to the state variables. So in order to achieve a good approximation performance, which is the key for success in approximate dynamic programming, nonlinear function approximators, such as neural networks can be employed. However, classical feedforward neural networks are nonlinear with respect to their weight parameters. This fact complicates the development of a method that can guarantee convergence of the existing policy evaluation methods, since no policy evaluation method has proven to converge for nonlinear (in the parameters) approximators. Instead, we employed the RBF neural network architecture for approximating the cost-to-go function. In this way, we manage to capture the nonlinearities of the true cost-to-go function, while assuring that the policy evaluation method will not diverge.

The approximation architecture we employ has the form:

$$\tilde{J}(x,w) = \phi(x)'w, \quad \forall x \in \mathbb{X} \quad (13)$$

where $\phi(x)$ is an L-dimensional vector associated with the state x that has components $\phi_1(x)$, $\phi_2(x)$, $...,\phi_L(x)$ while w is a weight vector with components $w(1), w(2), .., w(L)$. We use Gaussian functions with fixed centers as basis functions. The centers of the basis functions are selected in batch mode before the method starts operating. Special care needs to be taken for the center selection due to the localized character of the radial basis functions. The centers must span the entire state space in order to achieve a global behavior of the network. Once this is accomplished, the network can combine good approximation properties through the entire state space while retaining its simple linear structure with respect to the weight parameters.

2.2 Importance of the Post Decision State

Notice that the policy improvement step in eq. (12) involves the calculation of the minimum of an expectation that implies knowledge of the transition probabilities of the markov chain. However, these probabilities are generally not known. To overcome this problem we reformulate the problem using the post decision state x_t^u, i.e. the state of the system that emerges after the control has acted on it. We denote by x_t the pre-decision state, that is the state before control u_t has been applied but after information ω_{t-1} has arrived. To make it clearer, we define the

information history of the markov decision process up to time t, using the pre-decision state:

$$\mathcal{F}_t = \left(x_0, u_0, \omega_0, x_1, u_1, \omega_1, ..., x_t, u_t, \omega_t\right) \quad (14)$$

The dynamics of the system are given by eq. (1). We notice that the transition from one state to another is stochastic. The control function is given by:

$$\mu(x_t) = \arg\min_{u_t \in \mathbb{U}} \mathbb{E}\left[\left(g(x_t, u_t, x_{t+1}(\omega_t)) + aJ^*(x_{t+1}(\omega_t))\right)\right] \quad (15)$$

If a single simulation is used (i.e. if we use a single sample ω_t) to approximate the expectation in (15) we have:

$$\mu(x_t) = \arg\min_{u_t \in \mathbb{U}} \left(g(x_t, u_t, x_{t+1}(\omega_t)) + aJ^*(x_{t+1}(\omega_t))\right) \quad (16)$$

Eq. (16) implies that in order to calculate u_t, the value of $x_{t+1}(\omega_t)$ is needed, which is not yet available.

We define the post decision state x_t^u as the state of the system after decision u_t has been applied, but before new information ω_t has arrived. Now the history of the process is given by:

$$\mathcal{F}_t = \left(x_0, u_0, x_0^u, \omega_0, x_1, u_1, x_1^u, \omega_1, ..., x_t, u_t, x_t^u, \omega_t\right) \quad (17)$$

The transition from the pre-decision state to the post-decision state is given by a function of the form:

$$x_t^u = f^u(x_t, u_t) \quad (18)$$

Notice that the transition from the pre-decision to the post-decision variable is deterministic.

We also define the transition from the post-decision state to the pre-decision state as a function f^ω of the form:

$$x_{t+1} = f^\omega(x_t^u, \omega_t) \quad (19)$$

The transition from the post-decision to the pre-decision state is stochastic. The relationship between the optimal pre-decision and the post-decision cost-to-go functions, J^* and J^u respectively is given by:

$$J^u\left(x_t^u\right) = \mathbb{E}\left[g\left(x_t^u, \omega_t\right) + aJ^*\left(x_{t+1}(\omega_t)\right) \middle| x_t^u\right] \quad (20)$$

$$J^*\left(x_{t+1}\right) = \min_{u_{t+1} \in \mathbb{U}} J^u\left(x_{t+1}^u\right) \quad (21)$$

The Bellman equation for the post-decision state becomes:

$$J^u\left(x_t^u\right) = \mathbb{E}\left[g\left(x_t^u, \omega_t\right) + a \min_{u_{t+1} \in \mathbb{U}} J^u\left(x_{t+1}^u\right) \middle| x_t^u\right] \quad (22)$$

Notice that the expectation has been drawn out of the *min* operator. The optimal policy μ^* is determined by:

$$\mu^*(x_t) = \arg\min_{u_t \in \mathbb{U}} J^u\left(x_t^u(u_t)\right) \quad (23)$$

Here, we calculate u_t given the value of ω_{t-1} which is available at the time we make the decision. So, we do not violate any informational constraints, i.e. we are not using information that is not available at the time we make the decision. Hence a single simulation is enough to generate an unbiased estimate of cost-to-go function. More generally, we can use a single simulation when we need to approximate terms of the form $E\left[\min\{\bullet\}\right]$, but for terms of the form $\min\left\{E[\bullet]\right\}$ we must use multiple simulations.

2.3 λ- Least Squares Policy Evaluation

Perhaps the most crucial part for the success of a neuro-dynamic programming method is the policy evaluation step. The method that is utilized must be reliable and must converge as fast as possible because this is the most time consuming part of the method. These methods originate from stochastic approximation techniques (Kushner and Yin, 1997), so they usually involve a positive diminishing stepsize parameter that needs much effort and experience to be tuned, in order to ensure convergence. Various heuristic stepsize parameters have been proposed in the literature, however the choice of the stepsize parameter seems to be problem dependent. The only temporal difference method until now that has been proven convergent without requiring a diminishing stepsize is λ-Least Squares Policy Evaluation (λ-LSPE) (Nedic and Bertsekas, 2003). The convergence proof for the method can be found in (Bertsekas *et al.*, 2003). The proof is generic for all models employing linear combinations of basis functions that depend on the state of the system and thus can be used to prove convergence of the proposed method. Due to space limitations, no further discussion on the convergence can be provided and the reader is referred to the above paper.

λ-LSPE is a gradient-like method based on temporal differences and λ-policy iteration, which involves the solution of a least-squares problem. The essence of the method is the progressive reduction of the discount factor in order to accelerate the policy evaluation step. The method involves incremental solutions of the following least-squares problems:

$$\bar{r}_t = \arg\min_r \sum_{m=0}^{t}\left(\phi(x_m)'r - \phi(x_m)'r_t - \sum_{k=m}^{t}(a\lambda)^{k-m} d_t(x_k, x_{k+1})\right)^2 \quad (24)$$

where $d_t(x_k, x_{k+1})$ is the temporal difference defined in (11). The new weight vector is given by:

$$r_{t+1} = r_t + \gamma(\bar{r}_t - r_t) \quad (25)$$

2.4 Description of λ-LSPE based RBF Optimistic Policy Iteration

Below describes in detail the proposed methodology. It is an optimistic policy iteration algorithm where the policy improvement step follows after each transition. We use an efficient implementation scheme for λ-LSPE by applying the Sherman-Morrison-Woodbury formula (Golub & Van Loan, 1989) to compute the rank one update of the inverse of the matrix B_t.

Detailed description of the λ-LSPE based RBF Optimistic Policy Iteration

---

Set $B_0^{-1} = \delta I$, $A_0 = \mathbf{0}$ and $b_0 = \mathbf{0}$, where B_0^{-1} is a $L \times L$ matrix, I is the $L \times L$ identity matrix, A_0 is a $L \times L$ matrix and b_0 is a $L \times 1$ vector.

1. Given the initial pre-decision state x_0, set t = 0 and generate a control u_0 according to the steps a and b:

a. Simulate the system from x_0 to x_0^u deterministically according to eq. (18) for each $u_0 \in \mathbb{U}$

b. Select $\bar{u}_0 = \arg\min_{u_0 \in \mathbb{U}} \tilde{J}^u\left(x_0^u(u_0), w_0\right)$

2. Simulate the system from x_0 according to eq. (18), applying control $\bar{u}_0$ to get the post-decision state x_0^u.

3. Set $t \leftarrow t+1$. Simulate the system from x_t^u generating a random sample ω_t and applying eq. (19) to obtain the next pre-decision state x_{t+1} and the one-stage cost $g\left(x_t^u, \omega_t\right)$.

4. Generate a control $\bar{u}_{t+1}$ according to $\bar{u}_{t+1} = \arg\min_{u_{t+1} \in \mathbb{U}} \tilde{J}^u\left(x_{t+1}^u(u_{t+1}), w_{t-1}\right)$

5. Simulate the system from x_{t+1} applying control $\bar{u}_{t+1}$ according to eq. (18), to get the post-decision state x_{t+1}^u.

6. Update the weight parameters of the RBF network using the recursive form of λ-LSPE:

$$w_t = w_{t-1} + \gamma_{t-1} B_{t-1}^{-1}\left(A_{t-1} w_{t-1} + b_{t-1}\right)$$

$$B_t^{-1} = B_{t-1}^{-1} - \frac{B_{t-1}^{-1}\phi\left(x_t^u\right)\phi\left(x_t^u\right)' B_{t-1}^{-1}}{1+\phi\left(x_t^u\right)' B_{t-1}^{-1}\phi\left(x_t^u\right)}$$

$$z_t = a\lambda z_{t-1} + \phi\left(x_t^u\right)$$

$$A_t = A_{t-1} + z_t\left(a\phi(x_{t+1}^u)' - \phi(x_t^u)'\right)$$

$$b_t = b_{t-1} + z_t g\left(x_t^u, x_{t+1}^u\right)$$

and return to step 3.

3. SIMULATION RESULTS

We applied the proposed λ-LSPE-based Optimistic Policy Iteration with RBF networks (OPI-λ-LSPE-RBF) as the cost-to-go approximator to a retailer-inventory problem presented in Van Roy *et al.* (1997). The system under investigation consists of one production facility, one warehouse and ten stores. The goal is to efficiently manage inventory levels at warehouses and stores in order to meet customer demands and minimize storage and transportation costs. For detailed information about the model we refer the interested reader to the aforementioned technical report. In table 1 we list the problem parameters that were used in this work. The proposed method was compared with an *s*-type policy that is described in (Van Roy *et al.*, 1997). The method was further compared to a neuro-dynamic programming algorithm using linear- with respect to the state variables- function approximators (OPI-λ-LSPE-LINEAR). We finally tested the method using an ε-greedy soft policy scheme (OPI-λ-LSPE-RBF-ε) as the cost-to-go approximator, that will be described later.

Table 1. Values for the parameters of the retailer-inventory management problem

Parameters of the model	PValues
number of stores	10
delay to stores	2
Delay to warehouse	2
production capacity	100
warehouse capacity	1000
store capacity	100
Probability of customer waiting	0.8
Cost of special delivery	10
Warehouse storage cost	3
store storage cost	3
shortage cost	60
mean demand	5
standard deviation of demand	14

The heuristic *s*-type policy is parameterized by two decision variables: a warehouse-order-up-to level and a store-order-up-to level. These two variables also constitute the vector of manipulated variables in the control model we propose using the neuro-dynamic approach. Hence, control u_t consists of these two variables. The number of state variables is 33, which produces an enormous cardinality, thus making classical dynamic programming techniques impractical. The optimal order-up-to levels for the *s*-type policy were optimized by evaluating all possible combinations using lengthy simulations. The optimal values resulting from the exhaustive enumeration were 250 for the warehouse and 12 for the stores. All the states visited and their corresponding cost-to-go values during the above simulations were used to locate the centers of the RBF weights and calculate an initial estimate of the weight parameters. Specifically, the centers of the RBF network were selected from a normal distribution with mean and standard deviation equal to the ones of the collected states. The weight parameters were calculated using standard least squares on the above data. After the preliminary training phase, the λ-LSPE-based Optimistic Policy Iteration was applied as described in the previous section. The step size parameter was set equal to 1 and remained constant during the execution of the method while λ was set equal to 0.8. As we can see in figure 1 the weights of the RBF network converge after approximately 1,500,000 state transitions.

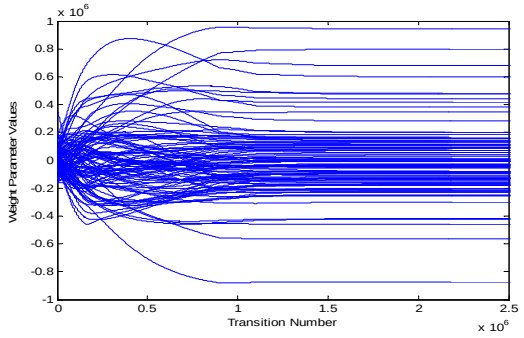

Fig.1. Evolution of weight parameters of the RBF network during the course of λ-LSPE-based Optimistic Policy Iteration

As mentioned previously, we experimented with a linear combination of the state variables as a cost-to-go approximator. The method failed badly, since the linear architecture was not able to capture the nonlinearities of the true cost-to-go function, thus resulting to a very large value bias value, while the values of the other weights were very close to zero.

We also tested a variant of the λ-LSPE-based Optimistic Policy Iteration with RBF networks that employed an ε-greedy policy in order to achieve exploration of the state space during the training phase. In de Farias & Van Roy (2000) it was shown that the trajectory of the weight vector can be considered as an approximation of the trajectory of an ordinary differential equation, where the limits of convergence correspond to the stationary points of the ODE. However, they pointed out that if no sufficient exploration is achieved, the respective ODE need not possess any stationary points. ε-greedy policy is a soft stochastic policy where the control action is selected randomly with probability ε, while the greedy control action with respect to the current estimate of the cost-to-go values is selected with probability 1-ε. We selected the initial value of ε close to one and as the algorithm progressed we gradually reduced its value asymptotically to zero. The weight parameters converged, but to completely different values from the ones shown in Fig. 1.

Finally, we tested the approximation architectures using a validation set which was common to all four methods. By the term validation set we mean a new initial state and a new independent realization of the random variable, which for the specific problem is customer demand. The results are summarized in table 2, where we observe that OPI-λ-LSPE-RBF exhibits the best performance in term of average cost, while OPI-λ-LSPE-LINEAR performs poorly.

Table 2. Performance comparison for the various methods

Method	Average Cost	Average Cost-to-go
s-type policy	1106.4	110790
OPI-λ-LSPE-RBF	1090.4	108115
OPI-λ-LSPE-RBF-ε	1199.3	119124
OPI-λ-LSPE-LINEAR	1742.7	173990

4. CONCLUSIONS

The concept of dynamic programming for the control of stochastic discrete-time dynamic systems has revived due to the emergence of neuro-dynamic programming techniques which circumvent the classical curse of dimensionality by employing simulation and function approximation. In this paper we presented an optimistic policy iteration algorithm based on RBF networks and the λ-LSPE method. Utilization of the RBF network architecture provides excellent approximations of the cost-to-go function, while retaining a linear, with respect to the parameters, model structure. The λ-LSPE method is a robust policy evaluation technique which does not require a diminishing stepsize parameter. We applied the method to a retailer-inventory management problem found in literature and obtained very encouraging results. We also discovered that employing a linear approximation of the cost-to-go function does not produce similar results, a fact that emphasizes the need for nonlinear approximation techniques such as RBF networks.

REFERENCES

Bellman, R. E. (1957). *Dynamic Programming*, Princeton University Press, Princeton, NJ

Bertsekas, D. P. (2001). *Dynamic Programming and Optimal Control*, 2nd edition, Athena Scientific, Belmont, MA.

Bertsekas, D.P., Borkar, V.S. and Nedic, A. (2003). Improved Temporal Difference Methods with Linear Function Approximation. Lab. for Info. and Decision Systems Report LIDS-P-2573, MIT, Cambridge, MA.

Bertsekas, D. P., and Tsitsiklis, J. N. (1996). *Neuro-Dynamic Programming*, Athena Scientific, Belmont, MA.

Boyan, J. A. (2002). Technical Update: Least-Squares Temporal Difference Learning. *Machine Learning*, Vol. 49, pp. 1-15.

de Farias D.P. and Van Roy, B.. On the Existence of Fixed Points for Approximate Value Iteration and Temporal Difference Learning. *Journal of Optimization Theory and Applications, vol. 105, no. 3, June 2000.*

Kushner, H. J. & Yin, G. G. (1997). *Stochastic Approximation Algorithms and Applications*. Springer, New York.

Nedic, A., and Bertsekas, D. P. (2003). Least Squares Policy Evaluation Algorithms with Linear Function Approximation. *Discrete Event Dynamic Systems: Theory and Applications,*Vol. 13, pp. 79-110.

Puterman, M. L., *Markov Decision Processes*, John Wiley Inc., New York, 1994.

Tsitsiklis, J. N., and Van Roy, B. (1997). An Analysis of Temporal-Difference Learning with Function Approximation. *IEEE Transactions on Automatic Control*, 42, pp. 674–690.

Sutton, R. S. (1988). Learning to Predict by the Methods of Temporal Differences. *Machine Learning*, vol. 3, pp. 9–44.

Sutton, R. S., and Barto, A. G. (1998). *Reinforcement Learning*, MIT Press, Cambridge, MA.

Tesauro, G.J. (1994). TD-Gammon, a Self-Teaching Backgammon Program, Achieves Master-Level Play. *Neural Computation*, Vol. 6(2), 215-219.

Van Roy, B., Bertsekas, D. P., Lee, Y. and Tsitsiklis, J. N. (1997). A Neuro-dynamic Programming Approach to Retailer Inventory Management, Technical report, Laboratory for Information and Decision Systems, MIT, Cambridge, MA.

Golub, G. H., and Van Loan, C. F. 1989. *Matrix Computations*, Johns Hopkins University Press, Baltimore, second edition.