
Annals of Operations Research 134, 215–238, 2005

Basis Function Adaptation in Temporal Difference Reinforcement Learning

ISHAI MENACHE imenache@tx.technion.ac.il
Department of Electrical Engineering, Technion, Israel Institute of Technology, Haifa 32000, Israel

SHIE MANNOR* shie@ece.mcgill.ca
Department of Electrical and Computer Engineering, McGill University, 3480 University Street, Montreal, Quebec, Canada

NAHUM SHIMKIN † shimkin@ee.technion.ac.il
Department of Electrical Engineering, Technion, Israel Institute of Technology, Haifa 32000, Israel

Received July 2003; Revised October 2003; Accepted January 2004

Abstract. Reinforcement Learning (RL) is an approach for solving complex multi-stage decision problems that fall under the general framework of Markov Decision Problems (MDPs), with possibly unknown parameters. Function approximation is essential for problems with a large state space, as it facilitates compact representation and enables generalization. Linear approximation architectures (where the adjustable parameters are the weights of pre-fixed basis functions) have recently gained prominence due to efficient algorithms and convergence guarantees. Nonetheless, an appropriate choice of basis function is important for the success of the algorithm. In the present paper we examine methods for adapting the basis function during the learning process in the context of evaluating the value function under a fixed control policy. Using the Bellman approximation error as an optimization criterion, we optimize the weights of the basis function while simultaneously adapting the (non-linear) basis function parameters. We present two algorithms for this problem. The first uses a gradient-based approach and the second applies the Cross Entropy method. The performance of the proposed algorithms is evaluated and compared in simulations.

Keywords: reinforcement learning, temporal difference algorithms, cross entropy method, radial basis functions

Introduction

Reinforcement Learning (RL) has evolved in the last decade into a major approach for solving hard Markov Decision Problems (MDPs). This framework addresses in a unified manner the problems posed by an unknown environment and a large state space (Bertsekas and Tsitsiklis, 1996; Sutton and Barto, 1998). The underlying methods are based on

* Corresponding author.
† This research was partially supported by the Fund for Promotion of Research at the Technion. The work of S.M. was partially supported by the National Science Foundation under grant ECS-0312921.

Dynamic Programming (DP), and include adaptive schemes that mimic either value iteration (such as Q-learning) or policy iteration (actor-critic methods). While the former attempt to directly learn the optimal value function, the latter are based on quickly learning the value of the currently used policy, followed by a slower policy improvement step.

A large state space presents two major challenges. The most obvious one is the storage problem, as it becomes impractical to store the value function (or optimal action) explicitly for each state. The other is the generalization problem, assuming that limited experience does not provide sufficient data for each and every state. Both these issues are addressed by the *Function Approximation* approach (Sutton, 1988), which involves approximating the value function by functional approximators with given architectures and a manageable number of adjustable parameters. Obviously, the success of this approach rests on some regularity properties of the state space, possibly induced by appropriate feature selection, and on the proper choice of an approximation architecture and size.

In a *linear* approximation architecture, the value of a state is computed by first mapping it to a low dimensional feature vector, and then linearly weighting these features using adjustable weights. The functions used to compute each entry in the feature vector are called the "basis functions". A notable class of linear function approximators is that of Radial Basis Function (RBF) networks (Haykin, 1998). In the RL context, linear architectures uniquely enjoy convergence results and performance guarantees, particularly for the problem of approximating the value of a fixed stationary policy (see Tsitsiklis and Van Roy, 1997; Nedic and Bertsekas, 2001). Yet, the approximation quality hinges on the proper choice of the basis functions.

In this paper we consider the possibility of on-line tuning of the basis functions, in the context of estimating the value function of a fixed policy. Following the common practice in RBF network training, we separate the problem of estimating the linear weights of the network, from the (harder) problem of adjusting the parameters of the basis functions themselves. The former is handled on a faster time scale, using standard $TD(\lambda)$ algorithms that optimize the linear weights for fixed basis functions. Basis function parameters are tuned in a batch manner, using an explicit score function. This approach allows the use of global optimization methods. Furthermore, it provides safeguards against possible divergence and performance degradation, which are well known to occur in on-line algorithms for non-linear architectures. We consider here two algorithms for adjusting the basis function parameters: The first is a local, gradient-based method, while the second is based on the Cross-Entropy (CE) method for global optimization (Rubinstein, 1999; Rubinstein and Kroese, 2004).

To evaluate the performance of these algorithms, we use the common "grid world" shortest path problem as a convenient test case. Our first simulations show that the gradient-based algorithm quickly converges to a steady-state value, but tends to get trapped in local minima that may be quite far from the optimum. This could well be expected, due to the inherent non-linearity in the basis-function parameters, which leads to a highly non-convex optimization problem. The CE method offers a significant improvement in this respect, at the expense of greater computational effort. Focusing on the CE method, we proceed to examine its performance for a larger state space, and to

examine the benefit of basis function adaptation as compared to increasing their number without adapting their parameters. Our results indicate the feasibility of the proposed algorithms for on-line tuning of the basis function in an unsupervised, reward-driven environment, and demonstrate the usefulness of efficient global optimization methods for this purpose.

Our approach is related to the work of Singh, Jaakkola, and Jordan (1995) on soft state aggregation, where a gradient-based algorithm was proposed in order to improve the weights relating each state to the possible clusters. We note that the number of adjustable parameters required in Singh, Jaakkola, and Jordan (1995) is a multiple of the state cardinality, hence this scheme is not suitable as an approximation architecture for large problems. When using RBFs as the basis functions, the adaptation procedure developed in this paper may be considered as a soft state aggregation procedure. Previous applications of the CE method to MDP optimization include Dubin (2002), where a direct search in policy space was considered. This approach was later extended by Mannor, Rubinstein, and Gat (2003). We also note that the CE method has been applied to several problems that fall within the MDP framework. Specifically, the buffer allocation problem (Alon et al., 2005) and robot path planning (Helvik and Wittner, 2001) were recently studied; see de-Boer et al. (2005) for additional references and a discussion.

The paper is organized as follows. We start in Section 1 with a short summary of necessary ideas concerning RL in large MDPs. We then present the basis function optimization problem in Section 2, and propose appropriate score functions. The CE and gradient-based adaptation methods are presented in Section 3. Our experiments are described in Section 4. We conclude and mention directions for future research in Section 5.

1. MDPs and reinforcement learning

Consider a learning agent in a dynamic environment, which is modelled as an MDP. The model evolves in discrete time, and is specified by the finite state space S, finite action sets $(A(s),\ s \in S)$, transition probability $p = (p(s' \mid s, a))$, and reward function $R = (R(s, a))$. At each time step $t = 0, 1, 2, \ldots$, the agent observes the state $s_t \in S$, and generates an action $a \in A(s_t)$. As a result, the agent obtains a reward $R_t = R(s_t, a_t)$, and the process moves to a new state $s_{t+1} = s'$ with probability $p(s' \mid s_t, a_t)$. The agent's goal is to find a (stationary) policy π, mapping states to actions, that maximizes some reward functional. In this work we consider the *discounted* reward criterion, specified by

$$V^{\pi}(s) = \mathbb{E}^{\pi}\left(\sum_{t=0}^{\infty} \gamma^t R_t \mid s_0 = s\right). \tag{1}$$

Here $0 < \gamma < 1$ is the discount factor, π is the agent's policy (to be optimized), $\mathbb{E}^{\pi}$ is the expectation operator induced by π, and s denotes the initial state. As it is well known (Puterman, 1994; Bertsekas, 1995), an optimal policy exists within the class of

(deterministic) stationary policies, that is, a policy which is prescribed as a map between states into actions. A randomized stationary policy can be identified with conditional probabilities $\pi(a \mid s)$, which specify the probability of choosing action a at state s. The value function V^π for such a policy is the unique solution of Bellman's equation, namely the following set of linear equations

$$V^\pi(s) = \sum_{a \in A(s)} \pi(a \mid s) \left(R(s,a) + \gamma \sum_{s' \in S} p(s' \mid s, a) V^\pi(s') \right), \quad s \in S. \tag{2}$$

The computation of V^π is a basic step in the policy improvement algorithm, which is one of the central DP algorithms for computing the optimal policy.

A direct solution of the above equation (or the corresponding optimality equation) may not be feasible, either due to unknown environment parameters (p and R) or due to the cardinality of the state space. In either case, temporal difference methods may be employed to estimate V^π. The resulting algorithm, where on-line estimates of V^π are used for policy improvement steps, is often referred to as an *actor-critic* architecture (e.g., Witten, 1977, Barto, Sutton, and Anderson, 1983). In what follows we focus on the task of learning V^π for a fixed policy π. Suppose we wish to approximate V^π using a functional approximator of the form $\tilde{V}_r : S \to \mathbb{R}$, where $r \in \mathbb{R}^K$ is a tunable parameter vector. Suppose further that the stationary policy π is in effect, and the agent observes the sequence of states and rewards $\{(s_t, R_t)\}_{t=0}^T$. The $TD(\lambda)$ algorithm (e.g., Sutton and Barto, 1998) for $\lambda \in [0,1]$ updates the parameter vector r at time t according to the iteration

$$r := r + \alpha_t d_t \sum_{i=0}^{t} (\gamma\lambda)^{t-i} \nabla \tilde{V}_r(s_i).$$

Here ∇ denotes the gradient of $\tilde{V}$ with respect to r, α_t is the learning gain, and $d_t \stackrel{\triangle}{=} R_t + \gamma V_r(s_{t+1}) - V_r(s_t)$ is the *temporal difference* corresponding to transition at stage t.

In a *linear* approximation architecture, the approximator $\tilde{V}_r$ is given by

$$\tilde{V}_r(s) = \sum_{k=1}^{K} r_k \varphi_k(s) \stackrel{\triangle}{=} r \cdot \varphi(s), \tag{3}$$

where $r = (r_1, \ldots, r_K)$ is the vector of tunable weights, $\varphi_k : S \to \mathbb{R}$ are the *basis functions*, also referred to as the state features, $\varphi(s) = (\varphi_1(s), \ldots, \varphi_K(s))$, and $\cdot$ is the standard inner product in $\mathbb{R}^K$. With this linear approximator, the $TD(\lambda)$ algorithm takes the form (Sutton (1988), Tsitsiklis and Van Roy (1997))

$$r := r + z_t (R_t + (\gamma \varphi(s_{t+1}) - \varphi(s_t)) \cdot r), \tag{4}$$

where $z_t = \sum_{i=0}^{t} (\gamma\lambda)^{t-i} \varphi(s_i) \in \mathbb{R}^K$. Roughly, $z_t(k)$ keeps track of the relevance of the present transition to the kth feature, and thus z_t is referred to as the eligibility vector.

Obviously, it may be computed iteratively as

$$z_{t+1} = \gamma\lambda z_t + \varphi(s_{t+1}), \tag{5}$$

with $z_{-1} = 0$. It has been shown in Tsitsiklis and Van Roy (1997) that for appropriate gains α_t, and assuming that the Markov chain induced by π is irreducible, the above algorithm converges to a unique parameter vector r^*. Furthermore, the approximation error is bounded by a fixed multiple of the optimal one (see Munos, 2003 for tighter bounds).

Finally, the $LSTD(\lambda)$ algorithm (Boyan, 2002) is a batch variant of $TD(\lambda)$, which converges to the same weight vector as the iterative algorithm above. This algorithm computes the following K dimensional vector and a $K \times K$ matrix:

$$b_t = \sum_{i=0}^{t} z_i R_i, \quad A_t = \sum_{i=0}^{t} z_i (\varphi(s_i) - \gamma\varphi(s_{i+1}))^\top. \tag{6}$$

The matrix A_t and the vector b_t may be incrementally updated after each transition. The eligibility vector z_t is updated as in (5). The approximating weight vector is calculated, when required, via $r = A^{-1}b$. This algorithm has been shown to give favorable convergence rates (e.g., Lagoudakis and Parr, 2001).

2. Evaluation criteria

Assume now that each of the basis functions φ_k in (3) has some pre-determined parametric form φ_{θ_k}. For example, for a Gaussian radial basis function of the form

$$\varphi_{\theta_k}(s) = \exp\left(-\frac{1}{2}(s - c_k)^\top W_k^{-1}(s - c_k) \right), \tag{7}$$

where the parameter θ_k corresponds to the center c_k and the width W_k. The value function in (3) may now be written as

$$\tilde{V}_{\theta,r}(s) = \sum_{k=1}^{K} r_k \varphi_{\theta_k}(s). \tag{8}$$

We refer to $\theta = (\theta_1, \ldots, \theta_K)$ as the *basis function parameters*, while $r = (r_1, \ldots, r_K)$ are the *weights*. Our problem is to determine the set (θ, r) so that $\tilde{V}_{\theta,r}$ best approximates V^π. Similar problems have been studied in the context of *supervised* learning (e.g., in radial basis networks), where it is common to handle separately the weight adjustment from the (harder) problem of parameter selection (Haykin, 1998). In this spirit, we use the $LSTD(\lambda)$ algorithm, as described above, to determine the weights while the basis functions are held fixed. Assuming that r is determined as a function of θ by an appropriate algorithm, we shall henceforth omit the explicit dependence of $\tilde{V}$ on r in our notation.

We are thus left with the problem of learning the "non-linear" parameters θ, namely tuning the basis functions, in order to allow a better approximation of the value function.

We shall require explicit evaluation criteria for the quality of a given approximation $\tilde{V}$. Since the true value function V^π is not available for comparison, we resort to the *Bellman error*, which is defined at each state as

$$J_\theta(s) \stackrel{\triangle}{=} \tilde{V}_\theta(s) - \left[R^\pi(s) + \gamma \sum_{s' \in S} P^\pi(s, s') \tilde{V}_\theta(s') \right], \tag{9}$$

where $P^\pi(s, s')$ is the probability that the next state is s' given that the current state is s and policy π is used.

It is well known that the norm of the approximation error $(V^\pi - \tilde{V})$ is bounded by a constant multiple of the Bellman error norm (Bertsekas, 1995; Bertsekas and Tsitsiklis, 1996). For the purpose of evaluating the quality of a value function estimation, it will be convenient to consider the following score function (weighted 2-norm):

$$S(\theta) = \sum_{s \in S} \beta(s) J_\theta(s)^2. \tag{10}$$

The weights $\beta(s) \geq 0$ may be chosen equal for all states, or alternatively may be used to emphasize certain states at the expense of others. A reasonable choice is $\beta(s) = p^\pi(s)$, the stationary steady state frequencies under the policy π. This means that we are willing to put up with relatively high Bellman errors for states that are not visited often (and for which it is harder to have good value approximations), as long as we have good approximations for frequently visited states. We note that a similar score function has been used in Singh, Jaakkola, and Jordan (1995); see also the discussion in Bertsekas and Tsitsiklis (1996), Chapter 6.10.

Direct calculation of the score (10) requires the model of the environment (transition probabilities and rewards). In addition, depending on the choice of weights, the steady state probabilities under policy π may be required. When these quantities are not available, we may try to estimate or approximate them on-line. When the state space is large, summing over all states is not feasible, and we might be forced to sum over a representative subset of states (see Bertsekas and Tsitsiklis, 1996). The representative states (RS) should typically be separated relative to some state space metric, and possibly represent significant states, such as bottlenecks (McGovern and Barto, 2001; Menache, Mannor, and Shimkin, 2002) or uncommon states (see Ratitch and Precup, 2002 for novel criteria for "complex" states). The actual number of RS is determined according to available memory and computation resources. The score function now becomes

$$S(\theta) = \sum_{s \in RS} \beta(s) \left(\tilde{V}_\theta(s) - \left[\tilde{R}^\pi(s) + \gamma \sum_{s' \in S} \tilde{P}^\pi(s, s') \tilde{V}_\theta(s') \right] \right)^2, \tag{11}$$

where $\tilde{P}^\pi(\cdot, \cdot)$ and $\tilde{R}^\pi(\cdot)$ are estimates for the transition probabilities and rewards, respectively. The weights $\beta(s)$ may be chosen as above.

When the MDP is deterministic (or almost so), a reasonable score function can be expressed directly in terms of the temporal differences, for example

$$S(\theta) = \frac{1}{T}\sum_{t=0}^{T-1}(\tilde{V}_\theta(s_t) - [R_t + \gamma \tilde{V}_\theta(s_{t+1})])^2 . \tag{12}$$

This score is comparable to (11), with $RS = S$, $\beta(s) = \tilde{p}^\pi(s)$, $\tilde{R}^\pi(s)$ and $\tilde{P}^\pi(s, s')$ taken as their empirical means. For batch processing the latter score does not seem to present a particular advantage, while equation (11) allows greater flexibility, for example in evaluating the approximation error at infrequently visited states. We note that if the reward or transition probabilities are stochastic then equation (12) considers the inherent variance in the value as well. If this evaluation criterion is used within a policy improvement scheme, this might lead to favoring policies with a small variance.

An alternative approach for defining a score for an unknown model is based on Monte-Carlo simulation. It might be possible to obtain a high-precision estimate $\hat{V}(s)$ for the value function $V^\pi(s)$ over the restricted set of representative states, using direct simulation. The score of the approximation $\tilde{V}_\theta$ may then be defined as

$$S(\theta) = \sum_{s\in RS} \beta(s)(\hat{V}(s) - \tilde{V}_\theta(s))^2 . \tag{13}$$

The apparent advantage of (13) over (11) is that there is no need to estimate the transition probabilities and rewards for the representative states. Yet, using equation (13) requires exhaustive sampling of the representative states and their successor states. The advantage of score (13) over (12) is that it only considers the value and the randomness in the reward or transitions is averaged out.

3. Basis function optimization

Recall that we consider the basis function approximation (8) to the value function $V^\pi(s)$. We present here two methods for optimizing the basis functions parameters θ with respect to the selected score function. The first is gradient-based, and the second employs the CE method. In either case, the state-reward history needs to be observed only once, and then stored and re-used at subsequent stages. It is assumed that this history is sampled under a fixed policy π, so that the state process is a stationary Markov chain.

3.1. Gradient based adaptation

Adaptation of the weights of basis functions using gradient descent and temporal difference learning dates back to Sutton (1988), where the case where only r is modified was considered; see Tsitsiklis and Van-Roy (1996) for a review. The algorithm presented in this section proceeds by interleaving optimization steps for either r or θ, while keeping

the other fixed. We shall concentrate on the score function provided in equation (11) and show that we are able to analytically calculate the gradient. The algorithm presented below may be easily adjusted for the score function of equation (13). This algorithm has similarities to the algorithms suggested by Bradtke (1993), Bradtke and Barto (1996) who considered using linear least squares for temporal difference learning, and to the algorithm used by Werbos (1990). The LSTD algorithm is used here to find gradients of the score with respect to both the linear (r) and non-linear (θ) parameters.

The basis functions are initialized with some choice of θ. The following steps are then repeated: (i) The $LSTD(\lambda)$ algorithm is applied to estimate the optimal weights r (with θ fixed). (ii) The basis-function parameters θ are updated using gradient descent steps with respect to the score $S(\theta)$, while r is held fixed. The gradient of $S(\theta)$ is easily obtained from (8) and (11), as follows:

$$\frac{\partial}{\partial\theta}S(\theta) = 2\sum_{s\in RS}\beta(s)J(s,\theta,r)\frac{\partial J(s,\theta,r)}{\partial\theta}\,, \tag{14}$$

with

$$J(s,\theta,r) = r\cdot\varphi_\theta(s) - \left[\tilde{R}^\pi(s) + \gamma\sum_{s'\in S}\tilde{P}^\pi(s,s')r\cdot\varphi_\theta(s')\right]. \tag{15}$$

The gradient descent step may be improved by letting r change optimally with θ, and computing the gradient in θ accordingly. Let $r(\theta)$ denote the optimal value of r given θ. Then we replace the partial derivative $\frac{\partial J(s,\theta)}{\partial\theta}$ with the composite derivative $\frac{\partial J(s,\theta,r(\theta))}{\partial\theta}$. Letting θ_q denote the q-th element of θ, the corresponding derivative is given by

$$\begin{aligned}\frac{\partial}{\partial\theta_q}J(s,\theta,r(\theta)) = {}& \frac{\partial r(\theta)}{\partial\theta_q}\cdot\varphi_\theta(s) + r(\theta)\cdot\frac{\partial\varphi_\theta(s)}{\partial\theta_q} \\ & - \gamma\sum_{s'\in S}\tilde{P}^\pi(s,s')\left[\frac{\partial r(\theta)}{\partial\theta_q}\cdot\varphi_\theta(s') + r(\theta)\cdot\frac{\partial\varphi_\theta(s')}{\partial\theta_q}\right].\end{aligned} \tag{16}$$

We note that $\frac{\partial\varphi_\theta(s)}{\partial\theta_q}$ is a vector of zeros except for one element, which corresponds to the derivative of the basis function to which θ_q belongs. The main issue remains calculation of the derivatives of the linear weights with respect to the basis function parameters, namely $\frac{\partial r(\theta)}{\partial\theta_q}$. We next describe how to estimate $\frac{\partial r(\theta)}{\partial\theta_q}$ by extending the $LSTD(\lambda)$ algorithm.

Recall that the $LSTD(\lambda)$ algorithm calculates the optimal weights via $r(\theta) = A^{-1}b$, with A and b defined in (6). We thus have the estimate

$$\frac{\partial r(\theta)}{\partial\theta_q} = -A^{-1}\frac{\partial A}{\partial\theta_q}A^{-1}b + A^{-1}\frac{\partial b}{\partial\theta_q}\,. \tag{17}$$

Using the expressions for A, b and z_t (equations (5) and (6), with φ replaced by φ_θ), we can form an estimate for their partial derivatives which is computed incrementally within

Table 1
Gradient-based adaptation algorithm.

Input: state-reward trace $(s_t, R_t)_{t=0}^{T}$
Parameters:
- Representative States (RS)
- Basis functions $(\varphi_{\theta_k})_{k=1}^{K}$
- Initial basis function parameters θ^0
- Initial learning rate vector, η_0.

Set $m := 0$
Repeat:
1. Simulation step:
   - Estimate $\tilde{V}_{\theta^m}$ using $LSTD(\lambda)$ (eq. (5), (6)) and $\frac{\partial r(\theta^m)}{\partial \theta}$ (equation (18))
   - Calculate $S(\theta^j)$, according to (11)
   - Calculate the gradient of $S(\theta^m)$ using (14)–(16).
2. Optimization step:
   - Set $\theta^{m+1} := \theta^m - \eta_m \cdot \nabla S(\theta^m)$
   - If $\|\theta^{m+1} - \theta^m\|_\infty < \epsilon$ then **stop**; otherwise
     - Set $m := m + 1$
     - Update η_m
     - Reiterate from step 1.

the $LSTD(\lambda)$ algorithm, in parallel with the usual updates of A and b:

$$\begin{aligned}
\frac{\partial A_t}{\partial \theta_q} &= \frac{\partial A_{t-1}}{\partial \theta_q} + \frac{\partial z_t}{\partial \theta_q}(\varphi_\theta(s_t) - \varphi_\theta(s_{t+1}))^\top + z_t\left(\frac{\partial \varphi_\theta(s_t)}{\partial \theta_q} - \frac{\partial \varphi_\theta(s_{t+1})}{\partial \theta_q}\right)^\top;\\
\frac{\partial b_t}{\partial \theta_q} &= \frac{\partial b_{t-1}}{\partial \theta_q} + \frac{\partial z_t}{\partial \theta_q} R_t;\\
\frac{\partial z_{t+1}}{\partial \theta_q} &= \lambda \frac{\partial z_t}{\partial \theta_q} + \frac{\partial \varphi_\theta(s_{t+1})}{\partial \theta_q}.
\end{aligned} \tag{18}$$

The partial derivative $\frac{\partial r(\theta)}{\partial \theta_q}$ may then be evaluated from equation (17) whenever required. The whole algorithm is summarized in Table 1.

The stopping condition requires having two consecutive parameter vectors, that are ϵ-close to each other (component-wise), where $\epsilon > 0$ is a small real value. The learning rate vector η_m may be separately chosen for each entry of θ. There are many heuristic rules for controlling the step size for gradient methods, see Bertsekas (1999) for discussion of such methods.

3.2. *Cross-entropy based adaptation*

Recall that the score function of either (11) or (13) may be calculated for a given vector of parameters θ. This score can naturally serve as the score function for the CE method for optimization. We now describe the CE-based adaptation algorithm and refer the reader

to de-Boer et al. (2005) for further discussion of the CE method. We assume that the parameter vector θ is drawn from a probability density function (pdf) $f(\cdot; v)$, which has some parametric form (e.g., Gaussian), with a meta-parameter v. A CE iteration starts by generating a sample of candidate parameter vectors $\theta^1 \ldots \theta^N$ drawn independently from $f(\cdot; v)$. The scores $S(\theta^j)$, $j = 1, 2, \ldots, N$ are then computed for these parameter vectors; in our case, the $LSTD(\lambda)$ algorithm (described in Section 1) is used for estimating the approximated value function $\tilde{V}$, which is then used to calculate the score. Next the parameters v of the random mechanism are updated using these scores, as follows. First, the "best" ρN parameter vectors θ^j are selected (both N and $0 < \rho < 1$ are predetermined parameters of the algorithm). Specifically, in the m-th CE iteration we order the $S(\theta^j)$'s in decreasing order, $S_{(1)} \geq \cdots \geq S_{(N)}$ and evaluate the $(1-\rho)$ sample quantile,

$$\hat{\gamma}_m = S_{(\lceil (1-\rho)N \rceil)}. \tag{19}$$

We retain those θ^j parameter vectors with $S(\theta^j) \leq \hat{\gamma}_m$, and discard all others. Next, a new meta parameter v_m is calculated by solving

$$v_m = \arg\max_v \sum_{j=1}^{N} I_{\{S(\theta^j) \leq \hat{\gamma}_m\}} \log f(\theta^j; v). \tag{20}$$

If f belongs to the natural exponential family (e.g., Gaussians) then (20) has a closed form solution (see Rubinstein and Kroese, 2004). For example, let us assume that each θ_q element of θ is drawn independently of the others according to a Gaussian pdf with mean μ_{θ_q} and variance $\sigma^2_{\theta_q}$. Let $m+1$ be the current CE iteration and $\hat{\gamma}_m$ the current threshold. The corresponding pdf is updated according to the following equations, which are the solution of (20) (see Rubinstein and Kroese, 2004)

$$\begin{aligned}
\mu^{(m+1)}_{\theta_q} &= \frac{\sum_{j=1}^{N} I_{\{S(\theta^j) \leq \hat{\gamma}_m\}} \theta^j_q}{\sum_{j=1}^{N} I_{\{S(\theta^j) \leq \hat{\gamma}_m\}}}, \\
\sigma^{2\,(m+1)}_{\theta_q} &= \frac{\sum_{j=1}^{N} I_{\{S(\theta^j) \leq \hat{\gamma}_m\}} \left(\theta^j_q - \mu^{(m+1)}_{\theta_q}\right)^\top \left(\theta^j_q - \mu^{(m+1)}_{\theta_q}\right)}{\sum_{j=1}^{N} I_{\{S(\theta^j) \leq \hat{\gamma}_m\}}}.
\end{aligned} \tag{21}$$

Here θ^j_q denotes the qth element of the parameter vector θ^j. While there are several possible stopping conditions, we chose to terminate the algorithm when the improvement in the score is small for d consecutive iterations, see equation (22). The stopping rule is slightly different than the standard stopping rule where $\hat{\gamma}_m$ is required not to change for a few iterations (e.g. de-Boer et al., 2005). The reason for this deviation from the standard stopping rule is that the score function is stochastic, so the score of the elite samples might fluctuate randomly even after effective convergence was attained. Using the score function of equation (22) leads to a faster convergence. The pseudo-code of the algorithm is given in Table 2.

Table 2
Cross-entropy adaptation algorithm.

Input: state-reward trace $(s_t, R_t)_{t=0}^{T}$
Parameters:
- Representative States (RS)
- Basis functions $(\varphi_{\theta_k})_{k=1}^{K}$
- The CE constants, ρ and N
- Initial values for the meta-parameter vector, v_0.

Set $m := 1$ (iteration counter)
Repeat:
1. Simulation step:
   - Generate a sample $(\theta^1, \theta^2, \ldots, \theta^N)$ from the density $f(\cdot; v_{m-1})$
   - For every θ^j, $j = 1, 2, \ldots, N$
     - Estimate $\tilde{V}_{\theta^j}$ using $LSTD(\lambda)$ (equation (5) and (6))
     - Calculate $S(\theta^j)$ (either (11) or (13)).
2. Optimization step:
   - Compute the sample $(1-\rho)$-quantile $\hat{\gamma}_m$ according to (19)
   - For the same sample $(\theta^1, \theta^2, \ldots, \theta^N)$, obtain the solution v_m of (20)
   - If for some $m \geq d$, say $d = 5$,

$$\hat{\gamma}_m \geq \hat{\gamma}_{m-1} - \epsilon, \ldots, \hat{\gamma}_{m-d+1} \geq \hat{\gamma}_{m-d} - \epsilon, \tag{22}$$

   then **stop**; otherwise set $m := m + 1$ and reiterate from step 1.

Note that the algorithm as described here is a batch algorithm in the sense that it uses a single experience trace, which is obtained initially. It is straightforward to modify the algorithm so that a different (updated or extended) experience trace is used at each iteration. Unlike in the gradient-based solution, $LSTD(\lambda)$ is invoked N times before an optimization step takes place. This could be a computationally expensive procedure. Yet, we observe that the $LSTD(\lambda)$ evaluations of different parameter vectors are computationally independent of each other. Thus, if parallel computing resources are at hand, $LSTD(\lambda)$ may be executed simultaneously for all parameter vector candidates.

4. Experiments

We describe in this section several experiments that were conducted on a maze-world, in order to evaluate the efficiency of the proposed algorithms. We start with a description of the maze world setup (Section 4.1). Our first experiment evaluates and compares the performance of both the gradient and the CE-based adaptation algorithms (Section 4.2). Subsequently, we restrict the discussion to the CE method. In Section 4.3 we examine a similar environment with a much higher density of states, thereby demonstrating the capability of the CE-based basis function adaptation algorithm to scale up in the state space cardinality. The final experiment (Section 4.4) compares the approximation error obtained by our CE-based adaptation process with those obtained with two alternatives;

the first is using a larger number of pre-determined basis functions, and the second is an unsupervised placement of basis functions, based on steady-state frequencies.

4.1. General setup

The domain which has been chosen for the experiments is a discrete two dimensional maze-world (e.g., Kaelbling, Littman, and Moore, 1996). In this domain an agent roams around the maze, trying to reach goal states as quickly as possible. The agent receives a small negative reward for each step, and a positive reward for reaching goal states. A goal state is also an absorbing state, after which the agent starts a new episode at a state which is chosen uniformly at random. The agent may choose to move to one of four neighboring tiles (unless there is an obstacle), and in our experiments its movement is perturbed with probability 0.1, meaning that it may fail to move in the chosen direction with this probability, in which case it moves in another (random) direction. The policy π that is selected for evaluation is the "shortest path" policy, which always follows the direction of the shortest Manhattan distance to the closest goal (if there is more than one such direction, the agent chooses between them with equal probabilities). The maze and the policy π are presented in figure 1. There are two goals in the maze (marked with "*G*"), one in the lower right corner and one in the middle, surrounded by a grey obstacle.

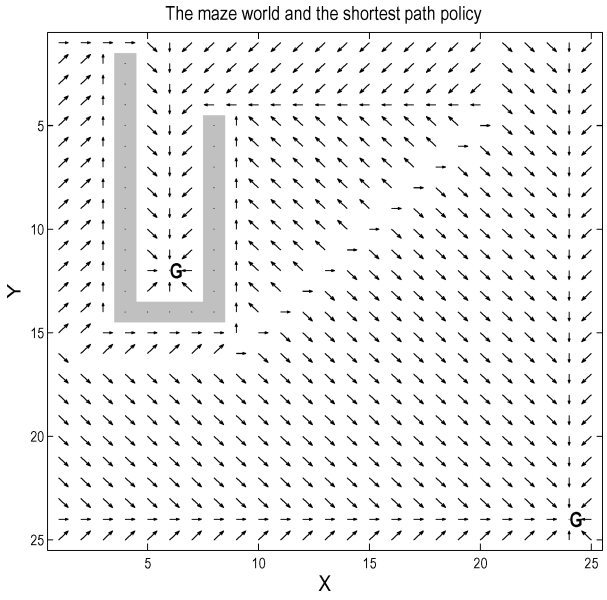

Figure 1. The maze-world. The obstacle is grey and the goals are marked with "G". The shortest path policy is shown by arrows. When two actions are optimal we plot a diagonal arrow in the direction of the mean of the two optimal directions.

The basis functions, which have been chosen in order to approximate the value function are the standard radial basis functions (equation (7)). The parameters to be tuned are the two dimensional center position c_k, and the 2×2 width matrix W_k of each basis function φ_{θ_k}. We further assume that the two dimensions of each Gaussian basis function are independent, reducing the width matrix to a diagonal matrix,

$$W = \begin{pmatrix} W^x & 0 \\ 0 & W^y \end{pmatrix}.$$

This gives 4 tunable parameters per basis function. Radial basis functions satisfy the differentiability requirement of the gradient-based algorithm, and are a natural choice for our domain, where the value function has a local nature (i.e., nearby states have similar values) over most of the state-space. We note that there might be better choices of basis functions, even in our domain; our main purpose here is to examine the tuning capability of the adaptation algorithms for a given family.

4.2. Gradient and cross entropy based adaptations

Our initial experiments concern the maze of figure 1, with a 25×25 state space, on which both the gradient-based and CE-based adaptation algorithms are tested and compared. The parameters of the runs, which are common for both methods are:

- The default negative reward is set to -0.5 and the reward for reaching a goal state to 8.
- A state-reward trace of $T = 10000$ steps is recorded and serves as the data for adaptation. As described, the initial state is chosen a random, and whenever the agent reaches a goal state it starts a new episode at a randomly chosen state (see previous page).
- The number of Gaussian basis functions is set to 11. Note that this gives 44 tunable parameters, as compared to a state space of $|S| = 625$ states.
- The $LSTD$ algorithm (see equation (6)) is used for the evaluation step, with $\lambda = 0.9$.
- We use the score criterion (11), where due to the small environment, all visited states serve as representative states. Equal weights ($\beta(s) = \frac{1}{|S|}$) are used for all states.

We implemented the gradient-based adaptation described in Section 3.1. The parameters of the runs, specific to this algorithm were: The learning rate of each of the 44 parameters (11 radial basis functions, four parameters each) was set to 5×10^{-4}; each run terminated when the score did not improve anymore; the initial basis functions placement was uniform; the initial width of each basis function was determined in a way that all the state space is "covered", with some overlap between neighboring basis functions (the initial setup of the basis functions is illustrated in the left part of figure 6). In all the experiments performed (differing in their state-reward trace), the convergence of the

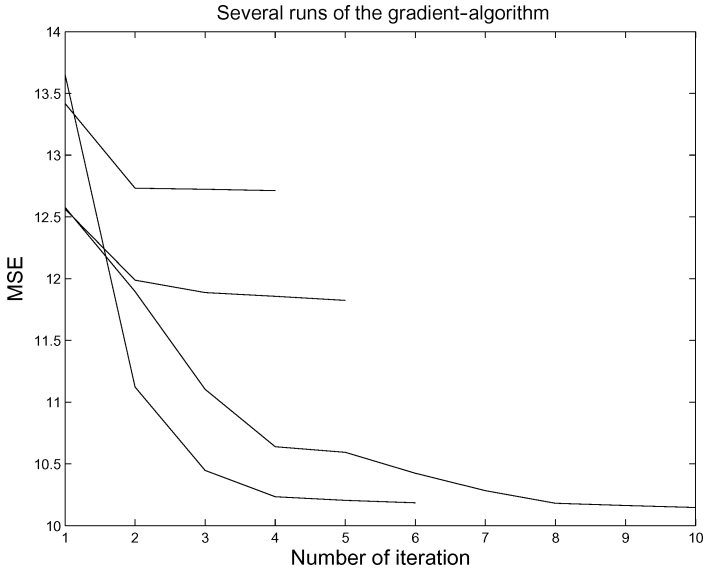

Figure 2. The mean square error (MSE) for a few sample runs of the gradient-based algorithm.

gradient-based adaptation was very fast and usually monotone (except for the vicinity of the minima, where it fluctuated). Some sample runs are presented in figure 2, where a monotone decrease of the mean square error may be observed. However, convergence is typically to a non-optimal value which significantly differs between different trials. This clearly indicates the existence of multiple local minima, as could be expected by the non-linear dependence of the basis functions on their parameters (see, e.g., Auer, Herbster, and Warmuth, 1996). To illustrate the extent of this local minima problem, we plot in figure 3 the mean square error of the estimated value (to which the algorithm converged) with respect to the true value. Note that each point in figure 3 represents a run of the gradient-based adaptation algorithm. Observe that some values are repeated, indicating convergence to identical parameter vectors; however, the many different values of the eventual error indicate a plethora of local minima, with widely varying error performance.

Applying the CE method to the maze-world requires choosing the distribution from which the basis function parameters are drawn in every CE iteration. For simplicity we assume that each parameter is drawn from an independent Gaussian distribution. This is performed for each of the four parameters that define each basis function. For example, the x coordinate of the center of the kth basis function, c_k^x, is assumed to be distributed according to a probability distribution function (pdf) $N(\mu_{c_k^x}, \sigma^2_{c_k^x})$. A similar assumption holds for the other three parameters of each radial basis function: the y coordinate of the center c_k^y, and the widths W_k^x, W_k^y along the two axes. We comment that when a parameter that corresponds to the width of a basis function (W_k^x or W_k^y) was negative, we simply took its absolute value. Since there are two parameters for the pdf of each of

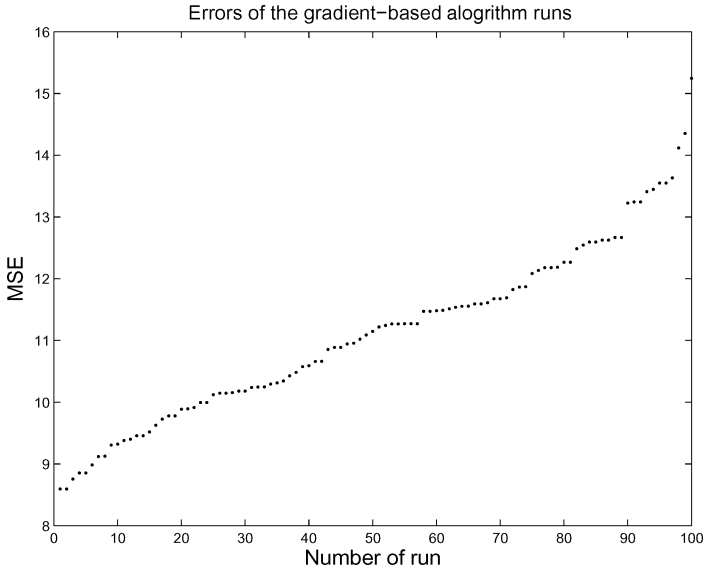

Figure 3. Final mean squared errors (after convergence) of the gradient-based algorithm, sorted from the lowest error to the highest.

the 44 parameters of the basis functions, the meta-parameter v is comprised of a total of 88 parameters. The pdfs of the basis function parameters are updated according to the set of rules given in (21).

The parameters of the CE method were set to $N = 100$ (samples per iteration) and $\rho = 0.1$ (fraction of retained best samples).[1] The initial values for the parameter pdfs were determined in a way that the initial experiment conditions were consistent with the runs of the gradient-based algorithm. Thus, the *means* of the centers (i.e., $\mu_{c_k^x}$, $\mu_{c_k^y}$) and the means of the widths (i.e., $\mu_{W_k^x}$, $\mu_{W_k^y}$) originally correspond to a selection of basis functions parameters θ, which covers (on average) the entire state space, with some overlap between neighboring basis functions. In figure 4 we present the true value function (left most) of the maze world along with two approximations: the center figure is the approximated value function obtained by the initial basis function placement, and the rightmost figure is the approximated value function using the basis function placement, which is obtained from the CE adaptation algorithm. The final value function is clearly closer to the true value function. In each case the linear weights r have been optimized as usual, using the *LSTD* algorithm.

Performance curves for the CE method are presented in figure 5. The benefit of the adaptation process is evident. The score (11) improved monotonously until it converged. In addition, using the true value function, the left graph shows the improvement of the (real) mean square error. Comparing the mean square error in figures 5 and 3 we observe that the CE method obtains a lower approximation error than the bulk of the gradient runs.

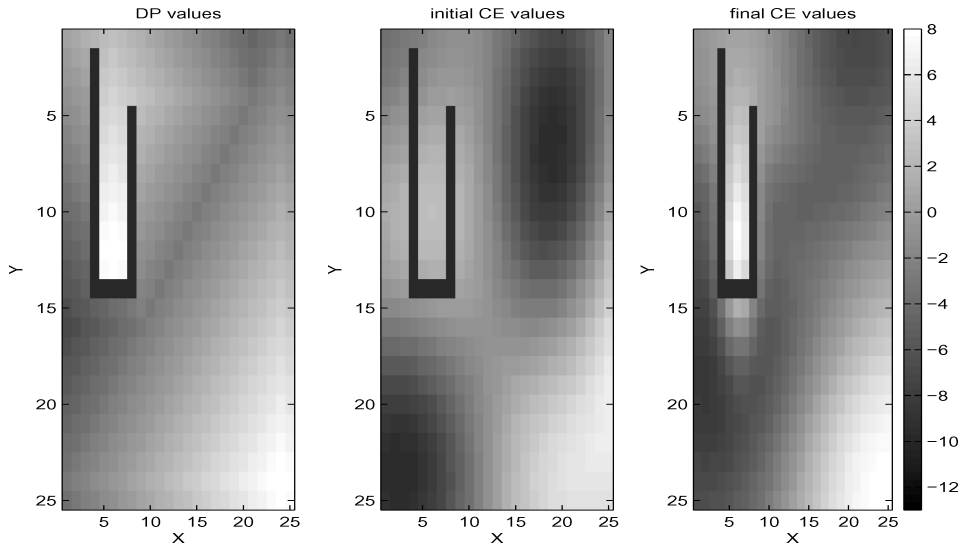

Figure 4. Maps of the value functions. Bright areas represent states with high value function. The barrier is denoted by black. The left map describes the true value function, as calculated using dynamic programming; the middle map is the estimated value which is calculated using the initial placement; the right map is the estimated value function obtained under the final placement of basis functions.

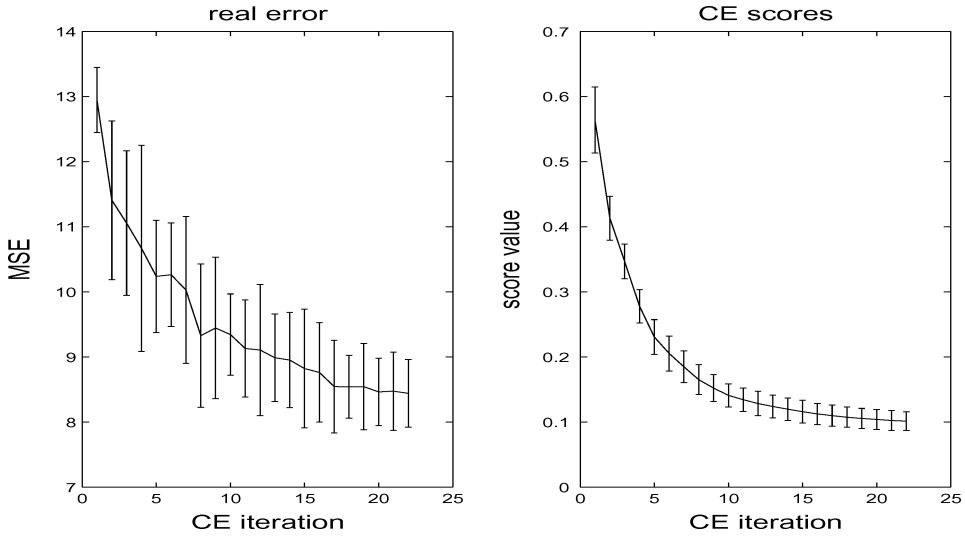

Figure 5. Performance graphs of the CE algorithm. On the left is the mean square error between the estimated value and the real value. On the right we plot the score of the CE given by equation (11). The x-axis in both graphs is the CE iteration counter. Results are averaged over 10 runs. The error bars represent the empirical standard deviation over these runs.

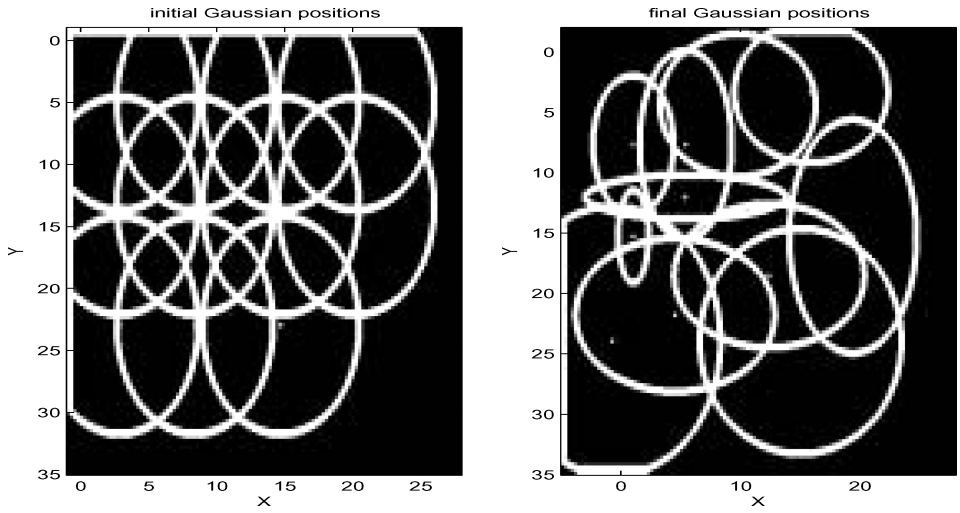

Figure 6. The Gaussian basis functions, before and after the adaptation process. We note that more basis functions are placed at areas of discontinuity, in our case at the zone of the barrier.

Figure 6 presents the initial and final arrangement of the radial basis functions in a typical CE run. One may notice that the basis function centers immigrate to "interesting areas", where the value function is not smooth (the barrier location in our case).

It is apparent that the CE method involves testing a large number of RBF parameter sets. Therefore, It is interesting to compare the computational effectiveness of the CE method with a random search over the basis function parameters. In figure 7 we plot the lowest score produced by the CE method as a function of the number of tested parameterizations. On the same graph we show the lowest score produced by a random search (starting with the same initial conditions as the first CE iteration). The superior performance of the CE method can be clearly observed. This indicates that the selection mechanism inherent in the CE method achieves the goal of guiding the search to preferred areas in the parameter space.

4.3. A larger state space

In this subsection we demonstrate the scalability of the proposed CE-based adaptation algorithm to larger state-spaces. We use the same geometric structure of the maze-world of the preceding experiments (figure 1). However we increase the density of states to 500×500, giving 250,000 states (instead of 625 states). The type of the basis functions (RBF), their number and initial arrangement, as well as the environment characteristics (i.e., the state transition probabilities and the evaluated policy) remain the same. Since the average number of states (until reaching a goal state) has increased by $\frac{500}{25} = 20$, we reduced the cost-per-step to $\frac{0.5}{20} = 0.025$ to keep the value function on the same scale.

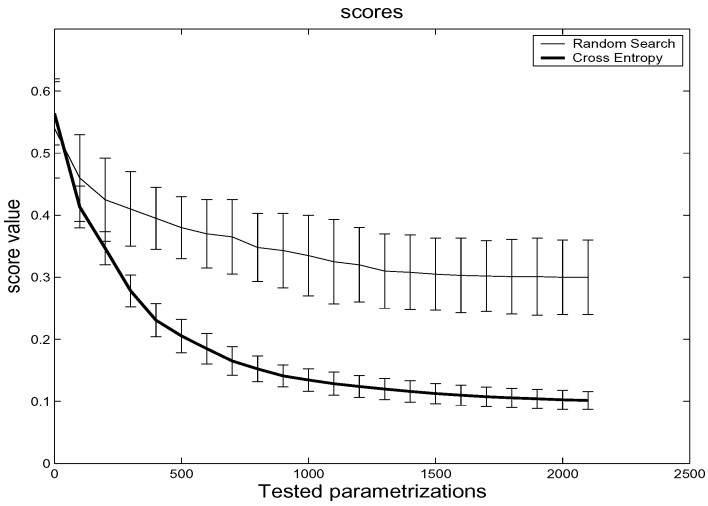

Figure 7. Average score as a function of the number of tested parameterizations for random search and CE-based adaptation. Results are averaged over ten runs with an error bar representing the empirical standard deviation.

Similarly, the number of state-reward pairs in the history trace was increased to 500,000. We used RS to save memory resources. The relevant statistics were collected for just 1% of the states, distributed evenly in the state space. In addition, in order to further save memory we switched to the score criterion (13), with $\beta(s) = \frac{1}{|RS|}$. Each episode was initiated in a randomly selected RS. The simulation results for $N = 200$ and $\rho = 0.05$ are presented in figures 8 and 9. The CE algorithm improves the score by a factor of 10 and the average error by a factor of more than 7 compared to the initial placement of the basis functions.[2] It may be observed that the relative improvement in the score is similar to the smaller maze. However, the improvement in the MSE (relative to the true value function) is significantly better, indicating the appropriateness of the score (13) in this case. It is interesting to observe the distribution of errors across the state space, which is depicted in figure 10. States suffering from high errors are those around the obstacle. This can be attributed to the discontinuity in the value function in this area, which the (smooth) basis functions find hard to approximate, and possibly to the low frequency of visits to these areas, which are not on an ordinary path to one of the goals (meaning that the only way to visit these states is to start an episode from a nearby state).

4.4. Additional comparisons

So far we proposed a method which couples RL with *supervised* selection of basis functions. The adaptation process is guided so it minimizes the Bellman error of the approximated value function. We now consider two alternatives to this process.

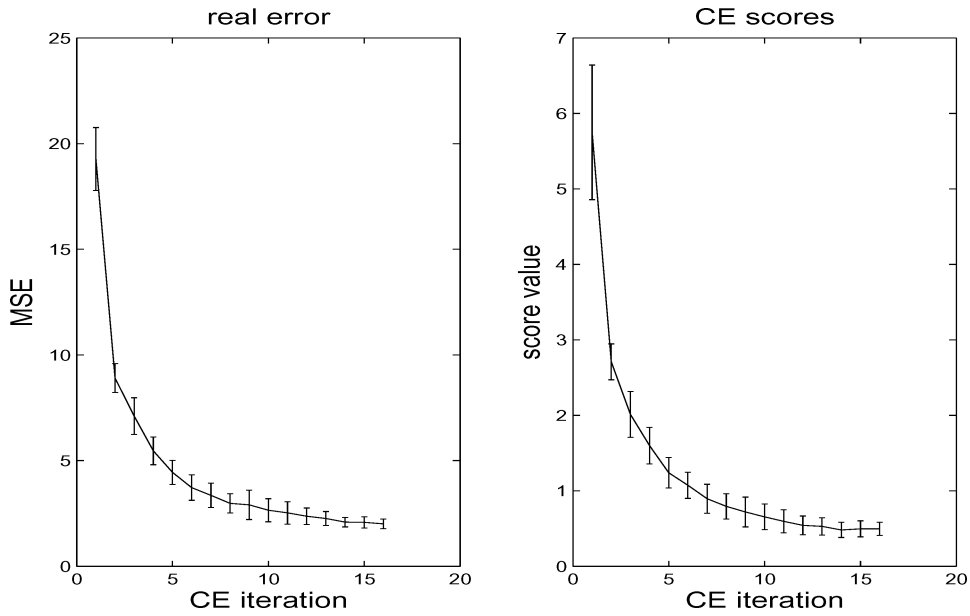

Figure 8. Performance of the CE adaptation algorithm for the 500 × 500 maze. On the left is the mean square error between the estimated value and the real value. On the right we plot the score of the CE given by equation (13). Results are averaged over 10 runs.

1. Increasing the number of basis functions, while keeping the uniform placement. This may avoid altogether the need for basis function tuning. The obvious drawback, besides a less compact representation, is the additional effort required to tune a larger number of linear weights (for fixed basis functions).
2. Unsupervised placement of basis functions, based on the empirical frequencies of the state samples alone.

The latter point is motivated by a common practice of RBF network training, where the basis function parameters (centers and widths in our case) are first selected; the linear weights of the (fixed) network are then calculated as a least squares problem. As reviewed in Ghosh and Nag (2000), the common methods for an initial unsupervised placement of basis function are (i) Basis functions centers are placed at randomly selected subsets of the data points; (ii) Clustering algorithms (e.g., K-Means); (iii) The basis functions are fitted as a mixture-of-Gaussians distribution to an empirical sample.

It is therefore of interest to compare the efficiency of the above (simpler) methods to the value function driven adaptation process, described in our work. We examined the third method, where the motivation is to concentrate basis functions in areas where there are many inputs points (states). Since we use RBFs, it is only natural to perform a maximum likelihood estimation for the steady-state occupancy measure. We solved the above problem using the celebrated Expectation-Maximization (EM) algorithm (e.g.,

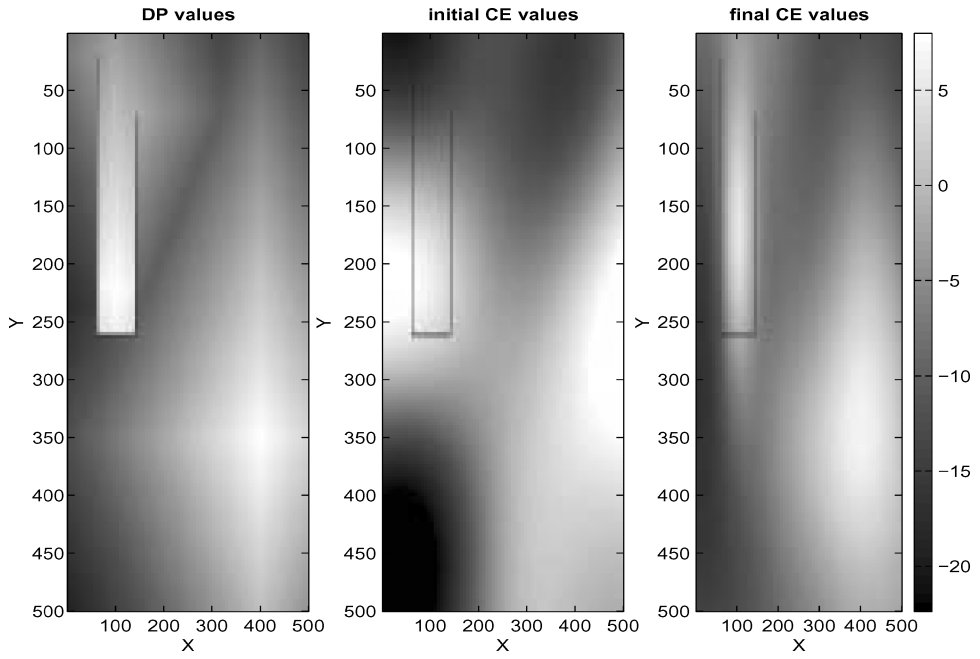

Figure 9. Maps of the value functions for the 500×500 maze. Bright areas represent states with high value function. The barrier is denoted by black. The left map describes the true value function, as calculated using dynamic programming; the middle map is the estimated value which is calculated using the initial placement; the right map is the estimated value function obtained under the final placement of basis functions.

McLachlan and Krishnan, 1997), which gives a convenient solution for density estimation with a mixture of Gaussians. The linear weights of the system are still calculated by the $LSTD$ algorithm.

We compared the above approaches (i.e., adding basis functions, and the latter unsupervised approach) with the CE-based adaptation on a 50×50 maze-world with the same topology as in figure 1. The mean squared error was calculated under each of the three approaches for different number of basis functions (4, 6, 9, 16, 25, and 36 basis functions). We used the gmm function in the Netlab neural network software[3] in order to perform EM for the Gaussian mixture model. A state-reward trace of $T = 20000$ steps was used. The CE-based adaptation algorithm using score function (13) was executed for 10 iterations, while N was set to $20\ K$ (K is the number of basis functions; we chose N in the spirit of Remark 3.6 in de-Boer et al. (2005), whereby in the stochastic node networks (SNN) model, N is recommended to be a multiple of a constant C and the number of adjustable parameters; here $C = 5$) and ρ to 0.05. The results are presented in figure 11. In that figure we plot the mean squared error as a function of the number of basis functions for: a uniform placement of basis functions (which is the starting point of the CE-based adaptation algorithm), an unsupervised placement of basis functions, namely the outcome of the EM algorithm, and the CE-based adaptation algorithm. The

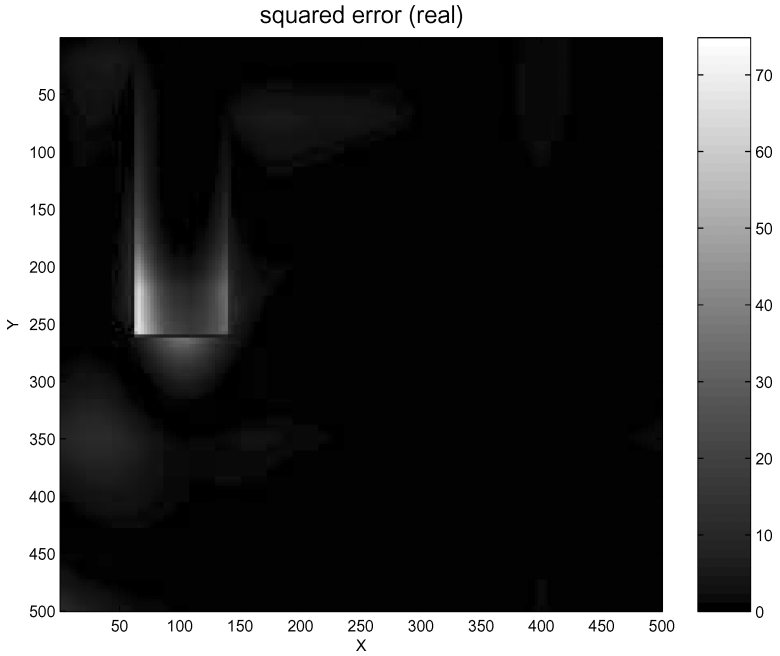

Figure 10. The real error per state for the 500×500 maze, i.e., the absolute difference between the DP value and the value after the CE adaptation process. Bright areas represent states with high value estimation error.

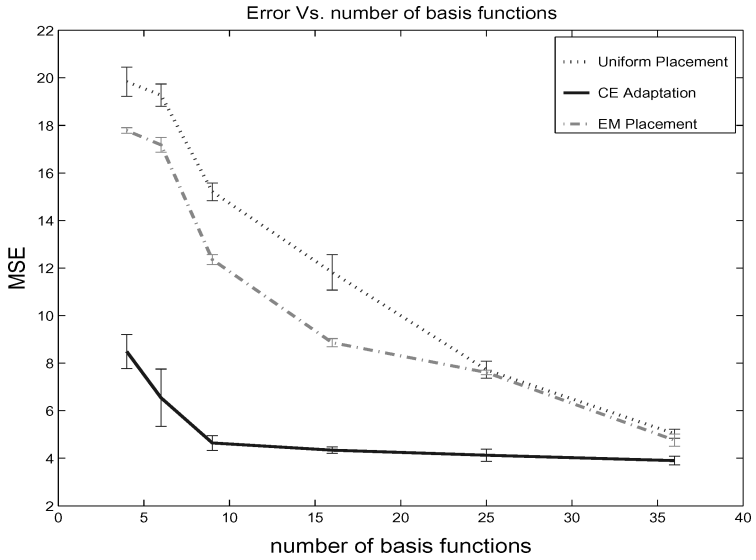

Figure 11. Performance graphs of the CE adaptation, comparing it with EM placement and uniform placement of Gaussian basis functions. Here the x-axis represent the number of basis functions that were used. The y-axis represents the (final) mean square error. Results are averaged over 10 runs.

experiment indicates that adding more basis function improves performance, yet the error with just 9 basis functions and the CE-based adaptation is smaller than the error obtained by the other two approaches using 36 basis functions. One can also observe that unsupervised placement of basis functions outperforms uniform placement, but is still considerably inferior to the performance of the CE-based adaptation. Both observations point to the performance advantage of the supervised adaptation process.

5. Conclusion and future directions

We have addressed in this paper the problem of on-line adaptation of the basis function parameters in temporal difference learning. Gradient-based and a CE-based adaptation methods were proposed. The gradient-based adaptation algorithm was observed to quickly converge to local minima so that a global optimization approach was called for. The CE-based adaptation managed to avoid local minima, and achieved superior tuning of the basis functions parameters. Furthermore, the CE method outperformed both a uniform placement of a larger number of basis functions, and unsupervised density-based placement of basis functions.

The algorithms and simulations that were presented here demonstrate the feasibility of basis function tuning in an unsupervised, reward-driven environment. Additional development and experimentation is required in order to assess the efficacy of these methods in large real-world problems. On the algorithmic side, a promising idea which may lead to faster convergence is to modify only a subset of basis functions in each CE iteration, based on regional separation of the state space. The criteria for choosing which subset of parameters should be optimized is an interesting research direction.

Within an RL scheme, low order approximations are important both to reduce the complexity of subsequent operations (such as performing a policy improvement step based on a value function approximation), and to accelerate learning. It is evident, however, that optimal tuning of the basis function parameters requires a great deal of computational effort. It is therefore important to evaluate its benefits within the overall RL paradigm. A simple way to accelerate learning is to tune the basis functions after some initial experimentation phase, and then proceed with the learning process with those fixed basis functions. An important research direction, which was not pursued here, is to devise methods for tuning the basis functions in relation with other RL algorithms, such as Q-learning and direct learning in the policy space (Sutton and Barto, 1998).

Notes

1. An additional CE related parameter α, which describes the smoothing factor in the update of the meta-parameter vector (see de-Boer et al., 2005), was set to 1 in all the experiments of this paper; we did not check the effect of optional smoothing on performance.
2. The CPU time for each run was between 44 to 47 hours on a Pentium IV 2.4 GHZ, 1 GB RAM platform.
3. Software available from http://www.ncrg.aston.ac.uk/netlab/.

References

Alon, G., D.P. Kroese, T. Raviv, and R.Y. Rubinstein. (2005). "Application of the Cross-Entropy Method to the Buffer Allocation Problem in a Simulation-Based Environment." *Annals of Operation Research* 134, 137–151, a preliminary version appeared in the third Aegean International Conference on Design and Analysis of Manufacturing Systems.

Auer, P., M. Herbster, and M. Warmuth. (1996). "Exponentially Many Local Minima for Single Neurons." In D. Touretzky, M. Mozer, and M. Hasselmo (eds.), *Advances in Neural Information Processing Systems*, Vol. 8, MIT Press, pp. 316–322.

Barto, A., R. Sutton, and C. Anderson. (1983). "Neuron-Like Adaptive Elements that can Solve Difficult Learning Control Problems." *IEEE Transactions on Systems, Man, and Cybernetics* 13, 834–846.

Bertsekas, D. (1995). *Dynamic Programming and Optimal Control.* Athena Scientific.

Bertsekas, D. (1999). *Nonlinear Programming*, 2nd edition. Athena Scientific.

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

Boyan, J.A. (2002). "Technical Update: Least-Squares Temporal Difference Learning." *Machine Learning* 49, 233–246.

Bradtke, S. (1993). "Reinforcement Learning Applied to Linear Quadratic Regulation." In S. Hanson and J. Cowan (eds.), *Advances in Neural Information Processing Systems*, Vol. 5, Morgan Kaufmann, pp. 295–302.

Bradtke, S. and A. Barto. (1996). "Linear Least-Squares Algorithms for Temporal Difference Learning." *Machine Learning* 22(1/2/3), 33–57.

de-Boer, P., D.Y. Kroese, S. Mannor, and R.Y. Rubinstein. (2005). "A Tutorial on the Cross-Entropy Method." Available from http://www.cemethod.org. *Annals of Operation Research* 134, 19–67.

Dubin, U. (2002). "Application of the Cross-Entropy Method to Neural Computation." Unpublished Master's thesis, Technion.

Ghosh, J. and A. Nag. (2000). "An Overview on Radial Basis Function Networks." In R.J. Howlett and L.C. Jain (eds.), *Radial Basis Function Neural Networks Theory and Applications.*, Physica-Verlag.

Haykin, S.S. (1998). *Neural Networks : A Comprehensive Foundation.* Prentice Hall.

Helvik, B.E. and O. Wittner. (2001). "Using the Cross-Entropy Method to Guide/Govern Mobile Agent's Path Finding in Networks." In *Proceedings of the 3rd International Workshop on Mobile Agents for Telecommunication Applications – MATA01.* Morgan Kaufmann.

Kaelbling, L.P., M. Littman, and A.W. Moore. (1996). "Reinforcement Learning – A Survey." *Journal of Artificial Intelligence Research* 4, 237–285.

Lagoudakis, M.G. and R. Parr (2001). "Model-Free Least-Squares Policy Iteration." In *Advances in Neural Information Processing Systems*, Vol. 14, Morgan Kaufmann, pp. 1547–1554.

Mannor, S., R.Y. Rubinstein, and Y. Gat (2003). "The Cross-Entropy Method for Fast Policy Search." In T. Fawcett and N. Mishra (eds.), *Machine Learning, Proceedings of the Twentieth International Conference*, AAAI press, pp. 512–519.

McGovern, A. and A.G. Barto. (2001). "Automatic Discovery of Subgoals in Reinforcement Learning Using Diverse Density." In *Proceedings of the 18th International Conference on Machine Learning*, Morgan Kaufmann, pp. 361–368.

McLachlan, G. and T. Krishnan. (1997). *The EM Algorithm and Extensions.* John Wiley & Sons.

Menache, I., S. Mannor, and N. Shimkin. (2002). "Q-cut-Dynamic Discovery of Sub-Goals in Reinforcement Learning." In *Proceedings of the 13th European Conference on Machine Learning*, Vol 2430, Springer, pp. 295–306.

Munos, R. (2003). "Error Bounds for Approximate Policy Iteration." In T. Fawcett and N. Mishra (eds.), *Machine Learning, Proceedings of the Twentieth International Conference*, AAAI press, pp. 560–567.

Nedic, A. and D. Bertsekas. (2001). *Least-Squares Policy Evaluation Algorithms with Linear Function Approximation.* LIDS Report LIDS-P-2537, to appear in *J. of Discrete Event Systems*.

Puterman, M. (1994). *Markov Decision Processes.* Wiley-Interscience.

Ratitch, B. and D. Precup. (2002). "Characterizing Markov Decision Processes." In *Proceedings of the 13th European Conference on Machine Learning*, Vol. 2430, Springer, pp. 391–404.

Rubinstein, R.Y. and D.P. Kroese. (2004). *The Cross-Entropy Method. A Unified Approach to Combinatorial Optimization, Monte-Carlo Simulation and Neural Computation*. Springer.

Rubinstein, R.Y. (1999). "The Cross-Entropy Method for Combinatorial and Continuous Optimization." *Methodology and Computing in Applied Probability* 1, 127–190.

Singh, S.P., T. Jaakkola, and M.I. Jordan. (1995). "Reinforcement Learning with Soft State Aggregation." In *Advances in Neural Information Processing Systems*, Vol. 7, MIT Press, pp. 361–368.

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

Sutton, R.S. and A.G. Barto. (1998). *Reinforcement Learning: An Introduction.* MIT Press.

Tsitsiklis, J. and B. Van-Roy. (1996). "Feature-Based Methods for Large Scale Dynamic Programming." *Machine Learning* 22, 50–94.

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

Werbos, P. (1990). "Consistency of HDP Applied to Simple Reinforcement Learning Problem." *Neural Networks* 3, 170–189.

Witten, I.H. (1977). "An Adaptive Optimal Controller for Discrete-Time Markov Environments." *Information and Control* 34, 286–295.