Binary Action Search for Learning Continuous-Action Control Policies

Jason Pazis JPAZIS@INTELLIGENCE.TUC.GR
Michail G. Lagoudakis LAGOUDAKIS@INTELLIGENCE.TUC.GR
Department of Electronic and Computer Engineering, Technical University of Crete, Chania, 73100, Crete, Greece

Abstract

Reinforcement Learning methods for controlling stochastic processes typically assume a small and discrete action space. While continuous action spaces are quite common in real-world problems, the most common approach still employed in practice is coarse discretization of the action space. This paper presents a novel method, called Binary Action Search, for realizing continuous-action policies by searching efficiently the entire action range through increment and decrement modifications to the values of the action variables according to an internal binary policy defined over an augmented state space. The proposed approach essentially approximates any continuous action space to arbitrary resolution and can be combined with any discrete-action reinforcement learning algorithm for learning continuous-action policies. Binary Action Search eliminates the restrictive modification steps of Adaptive Action Modification and requires no temporal action locality in the domain. Our approach is coupled with two well-known reinforcement learning algorithms (Least-Squares Policy Iteration and Fitted Q-Iteration) and its use and properties are thoroughly investigated and demonstrated on the continuous state-action Inverted Pendulum, Double Integrator, and Car on the Hill domains.

1. Introduction

A large number of real-world applications involve continuous control actions, such as the torque applied to the joints of a robot or the translational velocity of a mobile robot. The majority of known algorithms for learning control policies using Reinforcement Learning (RL) delivers policies which make decisions over a small set of discrete actions, but become inefficient when the number of actions grows beyond a certain number, not to mention the inability to handle cases with inherently continuous actions. A number of research efforts have attempted to find a way around this problem; some have met more success than others, yet none can be characterized as a comprehensive solution. This is especially true for domains that exhibit no or reduced temporal action locality, meaning that consecutive actions from an optimal policy may have quite distant values.

This paper addresses the problem of learning control policies in stochastic domains where actions (and states) are continuous. Our contribution is an efficient, generic algorithm which allows the majority of known RL algorithms to learn and execute effective continuous-action control policies, even in domains with reduced or no temporal action locality. The proposed approach, Binary Action Search (BAS), essentially approximates any continuous action space to arbitrary resolution and requires only binary decisions on behalf of the learning algorithm to efficiently identify the optimal action (up to the allowed resolution) in each state. The viability of the proposed approach is demonstrated in conjunction with two well-known RL algorithms (Fitted Q-Iteration and Least-Squares Policy Iteration) for learning continuous-action controllers on three well-known RL domains (Inverted Pendulum, Double Integrator, and Car on the Hill).

The paper is organized as follows. Section 2 provides the necessary background material, including a description of our recent work on Adaptive Action Modification. Binary Action Search (BAS) is presented in Section 3 and related work is discussed in Section 4. The benefits of the BAS algorithm are experimentally demonstrated in Section 5 and the paper concludes with a detailed discussion in Section 6.

2. Background

2.1. Markov Decision Processes

A *Markov Decision Process* (MDP) is a 6-tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma, D)$, where $\mathcal{S}$ is the state space of the process, $\mathcal{A}$ is the action space of the process, P is a Markovian transition model $\big(P(s, a, s')$ denotes the probability of a transition to state s' when taking action a in state $s\big)$, R is a

Appearing in *Proceedings of the 26th International Conference on Machine Learning*, Montreal, Canada, 2009. Copyright 2009 by the author(s)/owner(s).

reward function $\big(R(s,a)$ is the expected reward for taking action a in state $s\big)$, $\gamma \in (0,1]$ is the discount factor for future rewards, and D is the initial state distribution. A *deterministic policy* π for an MDP is a mapping $\pi : \mathcal{S} \mapsto \mathcal{A}$ from states to actions; $\pi(s)$ denotes the action choice in state s. The value $Q_\pi(s)$ of a state-action pair (s,a) under a policy π is defined as the expected, total, discounted reward when the process begins in state s, action a is taken at the first step, and all decisions thereafter are made according to policy π:

$$Q_\pi(s,a) = E_{a_t \sim \pi; s_t \sim P}\left[\sum_{t=0}^{\infty} \gamma^t R\big(s_t, a_t\big) \middle| s_0 = s, a_0 = a\right] .$$

The goal of the decision maker is to find an optimal policy π^* for choosing actions, which maximizes the expected, total, discounted reward for states drawn from D:

$$\pi^* = \arg\max_\pi E_{s \sim D}\left[Q_\pi\big(s, \pi(s)\big)\right] .$$

If the value function Q_{π^*} is known, a greedy policy, which simply selects actions that maximize Q_{π^*} in each state s, is an optimal policy. For every MDP, there exists at least one optimal deterministic policy. Value iteration, policy iteration, and linear programming are well-known methods for deriving an optimal policy from the MDP model.

2.2. Reinforcement Learning

In reinforcement learning, a learner interacts with a stochastic process modeled as an MDP and typically observes the state of the process and the immediate reward at every step, however P and R are not accessible. The goal is to gradually learn an optimal policy using the experience collected through interaction with the process. At each step of interaction, the learner observes the current state s, chooses an action a, and observes the resulting next state s' and the reward received r, essentially sampling the transition model and the reward function of the process. Thus, experience comes in the form of (s,a,r,s') samples. Several algorithms have been proposed for learning good or even optimal policies from (s,a,r,s') samples (Sutton & Barto, 1998).

Fitted Q-Iteration (FQI) (Ernst et al., 2005) is a batch-training version of the popular Q-Learning algorithm. It uses an iterative scheme to approximate the optimal value function, whereby an improved value function Q is learned at each iteration by fitting a function approximator to a set of training examples generated using a set of samples from the process and the Q-Learning update rule.

Least-Squares Policy Iteration (LSPI) (Lagoudakis & Parr, 2003) is a learning algorithm based on the approximate policy iteration framework, whereby at each iteration an improved policy is produced as the greedy policy over an approximation of the value function of the previous policy. LSPI uses linear approximation architectures consisting of a weighted sum of a set of basis functions ϕ for representing value functions. The value function of each policy is learned by solving a linear system formed using a set of samples from the process and the fixed-point property of the value function under the Bellman equation.

2.3. Adaptive Action Modification

A policy for an MDP typically selects the action to be performed next. However, when dealing with continuous action spaces over a range of values, successive decisions usually exhibit a great deal of temporal locality. Therefore, instead of making a decision about what particular action to perform, one can make a decision on how to modify the current action. In its simplest form, this decision boils down to increasing or decreasing the current action value by a fixed amount. A more flexible approach would change the size of the modification step adaptively. If for two successive time steps the sign of the decision remains the same (with increase taken as positive and decrease as negative), the step-size Δ is increased by a real-valued factor $K > 1$ to $K \times \Delta$, whereas when the sign of two successive decisions is different, the size of the step is decreased to $K^{-1} \times \Delta$.

The Adaptive Action Modification (AAM) algorithm (Pazis & Lagoudakis, 2009), summarized in Algorithm 1, is a scheme for realizing efficient continuous-action control policies in domains with temporal action locality. AAM uses a learned discrete, binary-decision policy over the space $(\mathcal{S}, \mathcal{A})$ for choosing whether to increase or decrease the current continuous-action value $a \in \mathcal{A}$ in the current state $s \in \mathcal{S}$. The modification step size Δ is adjusted adaptively as described above, according to the last two modification decisions and the scaling factor K. The value of a is updated according to Δ and this procedure is repeated at each time step. Policies learned using AAM in conjunction with Q-Learning, LSPI, and FQI reach superior performance on the Inverted Pendulum and the Bicycle Balancing and Riding domains compared to discrete-action policies and make full use of the continuous-action range.

3. The Proposed Approach

3.1. Binary Action Search

A policy typically decides which specific action to perform based on the current state. The AAM algorithm breaks away from this pattern and decides instead how to modify the current action. The internal binary policy essentially answers the following question: "*When in state s, would you rather increase or decrease your current action a?*"

Algorithm 1 Adaptive Action Modification

```
Input: state $s$, policy $\pi$, previous action $a_{t-1}$
Static: previous decision $e_{t-1}$, modification step $\Delta_{t-1}$
Output: continuous action $a_t$
$e_t \leftarrow \pi(s, a_{t-1})$      // binary decision ($+1$ or $-1$)
$\Delta_t \leftarrow \Delta_{t-1} K^{e_t e_{t-1}}$
$\Delta_{t-1} \leftarrow \Delta_t$
$e_{t-1} \leftarrow e_t$
$a_t \leftarrow a_{t-1} + e_t \Delta_t \left(a_{\max} - a_{\min}\right) / \left(a_{res} - 1\right)$
```

Algorithm 2 Binary Action Search

```
Input: state $s$, policy $\pi$, resolution bits $N$
Output: continuous action $a$
$a \leftarrow (a_{\max} + a_{\min})/2$
$\Delta \leftarrow (a_{\max} - a_{\min})2^{N-1}/(2^N - 1)$
for $i = 1$ to $N$ do
  $\Delta \leftarrow \Delta/2$
  $e \leftarrow \pi(s, a)$          // binary decision ($+1$ or $-1$)
  $a \leftarrow a + e\Delta$
end for
```

One potential problem with this approach is that the modification step cannot be chosen directly. Even if it is clear that the action value must be increased or decreased, it is still unclear what the magnitude of this modification step should be. As a result, the final action choice may over- or under- shoot depending on the current value of Δ. To minimize the effects of this phenomenon, experimental work with AAM had to rely on increasing the control frequency (yielding a smaller time step) to allow for quick recovery from a bad action choice. Unfortunately, adjustment of the control frequency is not always an option, nor does it constitute a principled way of coping with the lack of temporal action locality.

The key observation in designing an improved algorithm is that nothing prevents us from querying the internal binary policy more than once before we apply our decision. If, for example, the answer to the first question is to increase the action value, we can always compute the new value and check what the binary policy suggests for that new action value in the same state. If the suggestion is to decrease, then we have an over-shoot. If the suggestion is to increase, then we have an under-shoot. Repeated look-ahead queries will allow us to search for an action value that reduces the effects of over-shooting and under-shooting. Going a step further, we do not even have to adopt the adaptive modification step sizes suggested by AAM during our search, nor do we have to use the previous action value as the starting point of our search. Instead, we can start at any point and use any convenient step sizes to search the action range and successively approximate the value of the best continuous action choice in the current state. In the absence of domain knowledge, this search can be accomplished in an optimal way using a binary search scheme.

The proposed algorithm, called Binary Action Search (BAS), looks for the best action choice in the continuous action range $[a_{\min}, a_{\max}]$ using a finite number of binary search steps. The first query will be at the center of the range: "*When in state s, would you rather increase or decrease the value of the action $a = (a_{\max} + a_{\min})/2$?*" The answer will eliminate half of the action range. The next query will be at the center of the remaining range, and so on. In general, each binary decision will eliminate half of the remaining possible choices. The number of decisions required to come to a final decision is the same as the number of bits of the desired resolution. For example, if we want to have 256 values for the continuous action (8-bit resolution), we can use BAS to reach a final decision within 8 queries. The first query will eliminate 128 of the 256 potential choices, the second query will eliminate 64 of the remaining 128 choices, and so on, up to the eighth query which will leave us with just one choice. The Binary Action Search approach is summarized in Algorithm 2. Note that Δ is initialized to a value that allows for proper coverage of the entire action range (including $a_{\min}$ and $a_{\max}$) within a finite number of steps N.

3.2. Learning

The binary policy required by BAS answers the question: "*When in state s, would you rather increase or decrease the continuous action a?*". Most existing RL algorithms can be used in conjunction with BAS to learn such binary-action policies. There are only two requirements on the learning algorithm of choice: (a) it must be able to handle continuous state spaces, and (b) it must be able to produce a binary decision for each continuous action variable. The first requirement is dictated by the fact that the original state space $\mathcal{S}$ will have to be augmented with the latest value of the continuous action a, therefore the binary policy must be learned over the augmented state space $(\mathcal{S}, \mathcal{A})$. The second requirement is dictated by the need of a separate modification decision for each action variable.

One possible solution is to learn such policies using the learning scheme of the AAM algorithm, which interacts with the environment and generates one sample for each binary decision made. Although empirical tests verified that such an approach does indeed work well in practice, it represents a mismatch in learning conditions and excludes the use of online, on-policy learning methods within BAS. We would like a learning scheme which is good for both online and offline learning. The problem is that, even though we have as many binary decisions per step as we have resolution bits, there is only one interaction with the environment, and thus only one reward and one state transition observed.

A sound learning scheme for the binary decision policy, which supports both online and offline learning, needs one sample for each binary decision made. Therefore, from each action applied to the environment we need to generate samples for all the binary decision steps that led to this continuous action choice. The action in each such sample is the corresponding binary decision (increase or decrease). The (augmented) state in each such sample is the combination of the state of the process with the current search point in the action range. The last sample of each decision cycle, includes the observed state transition in the original state space $\mathcal{S}$ after the interaction with the environment, as well as a resetting of the search point to the center of the action range for the observed next state. Resetting the search point is required in order to keep the samples along the entire trajectory of binary decisions "connected". The reward is zero for all samples, but the last, which carries the actual reward observed and therefore it is the only one that should be discounted. For example, consider a resolution of 3 bits, the continuous action range $[1.0, 8.0]$, and an agent who internally makes the binary decisions -1, -1, $+1$ to find the continuous action $a = 2.0$ which is applied to state s. The sample from interacting with the environment will be $(s, 2.0, r, s')$, however the three samples used for learning the binary policy will be $\big((s, 4.5), -1, 0, (s, 2.5)\big)$, $\big((s, 2.5), -1, 0, (s, 1.5)\big)$, and $\big((s, 1.5), +1, r, (s', 4.5)\big)$.

3.3. Efficiency, Applicability, Variations

Evaluating the binary policy several times before acting does increase the computational demands relatively to AAM where only a single such evaluation is needed. Nevertheless, this internal binary decision cycle is very efficient. The number of evaluations needed grows logarithmically to the desired resolution, or, said differently, it grows linearly to the number of resolution bits. For most practical applications the required number of resolution bits is relatively low and as an added benefit the algorithm can be seen as an anytime algorithm. Computation may proceed refining the resolution as long as there is time and return the current action value upon expiration.

The BAS approach allows for selecting any desired action within the allowed resolution directly, in contrast to AAM which allows for reaching any desired value only after a number of interaction steps. This feature comes at the cost of increased computational complexity. Such a trade-off may be very reasonable not only in domains that by nature exhibit no temporal locality, but also in domains, where the control frequency cannot be increased. For example, if the control frequency of a robot actuator is fixed and the results obtained by AAM are unsatisfactory, then BAS could be used to significantly improve performance at a modest computational cost.

In some applications there may be areas of the action space that require finer resolution, while others are less important. BAS does not require that the action range is partitioned in equally-sized intervals. Any linear, non-linear, or even discontinuous, monotonous skewing function could be used to distort the action space, creating a more appropriate fit for the available resolution, as long as a total ordering is preserved. BAS will still eliminate half of the available action choices at each iteration, even if these action choices are not evenly distributed within the action space. That way we can achieve the same effective resolution with fewer decisions, thus reducing computational costs. The skewing function can be defined by the designers using domain knowledge, or it can be constructed automatically. It is important to note that the skewing function can be changed at any time without necessarily invalidating the current binary policy. Therefore, we may choose to use a skewing function if and when needed, or modify it during execution.

If the controller has to make simultaneous decisions for n continuous action variables, a policy choosing among 2^n discrete joint action choices is required (one binary decision for each one of the n action variables), which is admittedly expensive. Nevertheless, a discrete controller over a discretized action space offering m choices per action variable, would need a policy choosing among m^n joint actions for n action variables. Clearly, m cannot be less than 2 and, in fact, m is usually much larger than 2, therefore the requirements of our controller on the policy do not exceed those of a minimalist discrete controller. [1]

4. Related Work

There is a rich literature on learning continuous-action controllers. Most existing approaches rely on various forms of continuous-valued function approximators, such as neural networks (Strösslin & Gerstner, 2003), wire fitting (Baird & Klopf, 1993; Gaskett et al., 1999), tile coding (Santamaría et al., 1998), topological maps with interpolation (Touzet, 1997; Gross et al., 1998; Millán et al., 2002), and probabilistic models (Sallans & Hinton, 2004). In most cases, this function approximator delivers a value function over the combined state-action space and the main problem in these approaches is how to determine the maximizing continuous action in each state, which is a hard non-linear optimization problem. Monte-Carlo methods (Sallans & Hinton, 2004; Lazaric et al., 2008) have been used to alleviate this problem using sampling. It should be noted that, even though the optimal value function over the com-

[1]Experiments on the "Bicycle Balancing" problem (Ernst et al., 2005) have demonstrated the viability of this approach where BAS can learn successful control policies with as little as 1000 training episodes. The efficient application of BAS in multidimensional action spaces is current work in progress.

bined state-action space may contain several local action maxima in any given state, the optimal BAS value function yields binary decisions that point to the global action maximum and uses only twice as much memory space. Therefore, the identification of the maximizing action becomes a fairly easy task within BAS, avoiding intensive optimization methods.

Policy gradient methods (Prokhorov & Wunsch, 1997; Kimura & Kobayashi, 1998; Peters & Schaal, 2006) rely on (approximate) policy representations, which output continuous-action values directly and use estimated gradients to update the representation parameters and gradually improve the quality of the policy. Such methods typically make specific assumptions about the smoothness of the representation (so that it is differentiable) and require huge amounts of samples to make accurate improvement steps. These local optimization steps can only guarantee a locally good policy. The BAS approach makes no smoothness assumptions and can exploit (approximate) policy iteration learning methods, which are able to make large steps and explore the policy space much more efficiently.

Specialized methods exist for exploiting certain domain properties, such as temporal locality of actions. While such methods have shown promising results, their performance is limited by the implicit presence or explicit use of a low pass filter (Riedmiller, 1997; Pazis & Lagoudakis, 2009).

5. Experimental Results

We have integrated BAS with two well-known reinforcement learning algorithms: Least-Squares Policy Iteration (LSPI) and Fitted Q-Iteration (FQI) with either Least-Squares Regression or Extremely Randomized Trees.

5.1. Inverted Pendulum

The inverted pendulum problem (Wang et al., 1996) requires balancing a pendulum of unknown length and mass at the upright position by applying forces to the cart it is attached to. The 2-dimensional continuous state space includes the vertical angle θ and the angular velocity $\dot{\theta}$ of the pendulum. Three discrete actions are typically used, left force (-50 Newtons), right force ($+50$ Newtons), or no force (0 Newtons), and the problem is formulated as an avoidance task, whereby the goal is to keep the pendulum above the horizontal axis. In our experiments we chose to formulate the problem as a regulation task with a reward of $-((2\theta/\pi)^2 + (\dot{\theta})^2 + (F/50)^2)$, as long as $|\theta| \leq \pi/2$, and a reward of -1000, as soon as $|\theta| > \pi/2$, which also signals the termination of the episode. The discount factor of the process was set to 0.95. This formulation is significantly more difficult, since smoothness of motion and use of minimum force are also required while balancing the pendulum.

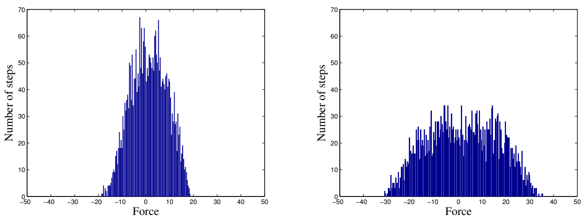

Figure 1. Inverted pendulum: Force histograms of policies learned by LSPI. Left: $10N$ noise. Right: $20N$ noise.

The action space in our formulation consists of the entire range of forces $[-50N, 50N]$ approximated to a resolution of 2^8 equally spaced actions. The state was augmented with the current action value F during the binary search, therefore it becomes a 3-dimensional vector $(\theta, \dot{\theta}, F)$. All actions are noisy (uniform noise in $[-10N, 10N]$ is added to the chosen action) and the transitions are governed by the nonlinear dynamics of the system (Wang et al., 1996), which depend on the current state and the current (noisy) control u:

$$\ddot{\theta} = \frac{g\sin(\theta) - \alpha ml(\dot{\theta})^2\sin(2\theta)/2 - \alpha\cos(\theta)u}{4l/3 - \alpha ml\cos^2(\theta)} \ ,$$

where g is the gravity constant ($g = 9.8m/s^2$), m is the mass of the pendulum ($m = 2.0$ kg), M is the mass of the cart ($M = 8.0$ kg), l is the length of the pendulum ($l = 0.5$ m), and $\alpha = 1/(m + M)$. A control interval of 100msec was used, which is an update frequency low enough to render approaches depended on temporal locality of actions, such as AAM, inapplicable.

The approximation architecture for representing the value function in this problem consists of a total of 56 basis functions. In particular, for each one of the two choices of the modification policy (increase/decrease), there is a separate block of 28 basis functions, including a constant term and 27 radial basis functions arranged in a $3 \times 3 \times 3$ grid over the 3-dimensional normalized augmented state space:

$$\phi = \Big(1 \quad , \quad e^{-\frac{\sqrt{(\theta/n_\theta - \theta_1)^2 + (\dot{\theta}/n_{\dot{\theta}} - \dot{\theta}_1)^2 + (F/n_F - F_1)^2}}{2\sigma^2}} \ ,$$
$$\cdots \quad , \quad e^{-\frac{\sqrt{(\theta/n_\theta - \theta_3)^2 + (\dot{\theta}/n_{\dot{\theta}} - \dot{\theta}_3)^2 + (F/n_F - F_3)^2}}{2\sigma^2}} \Big)^\top \ ,$$

where the θ_i's, $\dot{\theta}_i$'s and F_i's are in $\{-1,\ 0,\ +1\}$, while $n_\theta = \pi/2$, $n_{\dot{\theta}} = 2$, $n_F = 50$, and $\sigma = 1$. For the discrete controllers, a similar 3×3 grid over the 2-dimensional normalized state space (without the F term) was used along with a constant term, giving a block of 10 basis functions for each discrete action.

The performance of the learned BAS controllers coupled with both LSPI and FQI was excellent; the pendulum

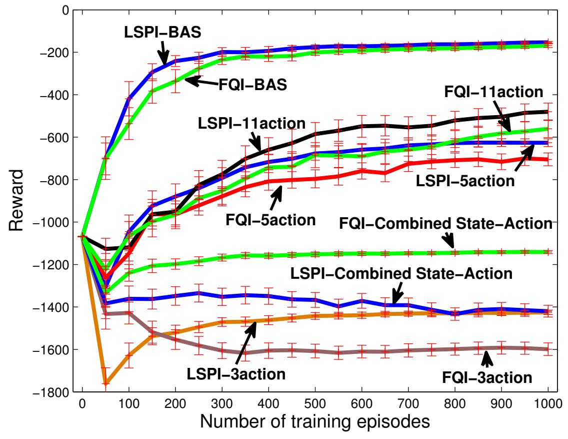

Figure 2. Inverted pendulum : Total accumulated reward.

stayed at the upright position ($\theta \approx 0$) for the duration of testing (3,000 steps or 5 minutes). Angular velocity and acceleration were concentrated close to zero, with no sudden spikes. Figure 1 (left) is a histogram of the applied force for a test run with a BAS controller learned using LSPI. The majority of applied forces is concentrated around zero. The average mean force magnitude over 100 such learned policies was $6.65N$ for LSPI and $6.53N$ for FQI. In contrast, 100 3-action controllers achieve an average mean force magnitude of $17.91N$ for LSPI and $20.09N$ for FQI.

In the above experiments, the BAS controllers yield small mean force magnitudes, since they do not need very large forces to counteract the $[-10N, 10N]$ uniform noise. It is natural to ask whether the learned controllers can use the rest of the action range when the situation requires it. To answer this question, we fixed 100 learned BAS controllers and tested them (100 times each) under increasing noise levels. It was observed that the higher the noise level, the larger the force magnitudes used. Figure 1 (right) shows the force histogram of a policy learned under $[-10N, 10N]$ noise and tested under $[-20N, 20N]$ noise. The BAS controllers were able to keep the pendulum balanced up to noise levels in the low 20's. In particular, for a $[-20N, 20N]$ noise level the success rate of the BAS controllers was 99.64%. In contrast, 100 3-action discrete controllers in the same experiment were able to succeed less than half of the time (39.49%).

A number of systematic experiments were conducted to assess the effectiveness of learning under the BAS scheme. Training samples were collected in advance from "random episodes", that is, starting the pendulum in a randomly perturbed state close to the equilibrium state $(0, 0)$ and following a purely random policy. For each batch of training episodes, the learned policy was evaluated 100 times to estimate accurately the average cumulative reward. This experiment was repeated 100 times for the entire horizontal axis to obtain average results and the 95% confidence intervals over different sample sets. Each episode was allowed to run for a maximum of 3,000 steps corresponding to 5 minutes of continuous balancing in real-time. Figure 2 shows the total accumulated reward as a function of the number of training episodes. Clearly, the more actions available to the controller, the better its performance on the regulation task. The BAS controllers learn much faster than their discrete counterparts and achieve far better rewards.

We also experimented with learning an approximation to the optimal value function over the combined state-action space using a single block of the same 28 basis functions described above as the approximation architecture. The policy over this value function was extracted by exhaustively identifying the maximizing action over a discrete set of points uniformly spread along the action range. This approach is commonly used for approximating continuous actions. However, these controllers were unable to balance the pendulum when more than 5 actions were used. This can be explained by the limited opportunity for generalization over neighboring actions in this highly non-linear problem.

5.2. Double Integrator

The double integrator problem requires the control of a car moving on a one-dimensional flat terrain. The 2-dimensional continuous state space (p, v) includes the current position p and the current velocity v. The goal is to bring the car to the equilibrium state $(0, 0)$ by controlling the acceleration a, under the constraints $|p| \leq 1$ and $|v| \leq 1$. The cost function $p^2 + a^2$ penalizes positions differing from the home position ($p = 0$), as well as large acceleration (action) values. The linear dynamics of the system are: $\dot{p} = v$ and $\dot{v} = a$.

As the control frequency becomes lower, the car becomes more and more difficult to control. Large control inputs can easily make the car overshoot the target or even move outside its operating range. A control interval of 500msec was chosen in order to make the problem more challenging. Acceleration was restricted in the range $[-1, 1]$ and was approximated with an 8-bit resolution (256 values) resolution for the BAS controllers. A simple polynomial approximator with 10 terms was used for each BAS action:

$$\phi = (1, p, v, a, p^2a, v^2a, a^2, pv, pa, va, a^2p, a^2v)^\top$$

For the discrete controllers, a similar polynomial approximator (without any a terms) was used. Once again training samples were collected in advance from "random episodes" with a maximum length of 200 steps. For accurate assessment of performance, 100 controllers were trained in each

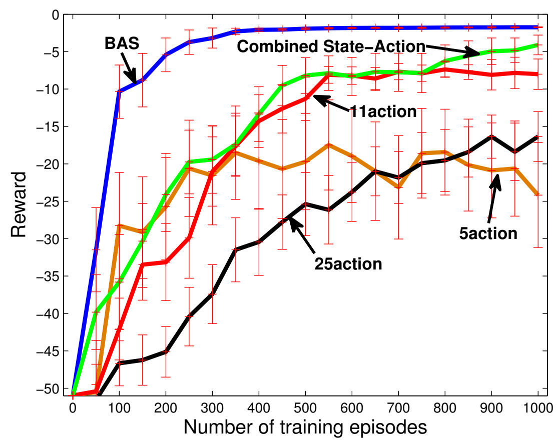

Figure 3. Double Integrator (LSPI) : Total accumulated reward.

case and tested starting at state $(1, 0)$ (maximum allowed p, zero v) for a maximum of 200 steps. The discount factor of the process was set to 0.98.

Figure 3 shows the total accumulated reward as a function of the number of training episodes. Once again, the BAS controllers learn much faster than their discrete counterparts and achieve far better rewards. Using a combined state-action approximator and evaluating for all 256 possible action choices yielded successful policies for this domain, albeit at a 16-fold increase in computational cost compared to BAS. This is to be expected since it is very easy to generalize over neighboring actions for the linear dynamics of the Double Integrator. Even though the performance of the combined state-action approximator is better than that of the discrete controllers, it still falls short of the performance achieved by the BAS controllers.

5.3. Car on the Hill

The car on the hill problem (Ernst et al., 2005) involves driving an underpowered car stopped at the bottom of a valley between two hills, to the top of the steep hill on the right. The 2-dimensional continuous state space (p, v) includes the current position p and the current velocity v. The controller's task is to (indirectly) control the acceleration using a thrust action u in $[-4, +4]$, under the constraints $p > -1$ and $|v| \leq 3$ in order to reach $p = 1$. The task requires temporarily driving away from the goal in order to gain momentum. The agent receives a reward of -1, if a constraint is violated, a reward of $+1$, if the goal is reached, and a zero reward otherwise. The discount factor of the process was set to 0.95. It has been noted that this is a domain where optimal controllers are of the "bang-bang" type (Ernst et al., 2005) and introducing more actions hurts performance. Therefore, we don't expect to achieve better results using BAS. Instead, we aim to assess performance in a domain that presents the worst possible match to our algorithm. Even in such domains, assuming learning has converged to an optimal policy, the BAS controller cannot be worse than any discrete controller, since all actions of the discrete controller are also available to BAS. However, learning the uselessness of the extraneous actions represents a hard learning problem and may have an impact on the learning performance of BAS.

Fitted Q-Iteration with Extremely Randomized Trees was used for learning both BAS (continuous, 8-bit resolution) and discrete (2 actions, -4 or $+4$) controllers. Our results for a 100msec control interval confirm that there is indeed a penalty for using continuous actions. For $10,000$ samples and starting at the bottom of the hill $(-0.5, 0)$ the discrete controller is able to reach the goal in $18-22$ steps in most runs. In contrast, the BAS controller requires $20-45$ steps on average. To a large extent this is because random sampling in the continuous action space has low chances of sampling good action choices (that is, the extreme actions at the limits of the range). As the number of samples increases and important parts of the state-action space are covered, the performance of BAS controllers improves significantly. Note that in an online learning setting, where the agent has the ability to focus on sampling promising parts of the state-action space, the performance of BAS controllers would be comparable to that of the discrete controllers. When the time step is increased to 500msec the advantage of discrete controllers over BAS controllers is lost. In fact, while discrete controllers frequently fail when the number of samples is limited (< 1000) BAS is able to reach the goal in $3-6$ steps.

6. Discussion and Conclusion

The proposed Binary Action Search approach offers several advantages: i) It is simple and easy to implement. ii) It requires no tuning of parameters. iii) It has low computational requirements. iv) It easily achieves fine resolutions impossible to reach with discrete actions. v) It makes no assumptions about the properties of the action space. vi) It requires only learning an internal binary-decision policy. vii) It can be used in conjunction with any RL algorithm supporting discrete actions and continuous states. viii) It can be used in online, offline, on-, or off- policy settings.

Of course, everything comes at a cost. The state space of the problem we are trying to handle is now more complex. It includes the original state variables and one new state variable (the continuous action). Although it definitely puts a strain to the underlying learning algorithm, its overhead is far less than naively increasing the number of possible discrete actions beyond a certain point. Using the learning scheme proposed, the number of samples is essentially multiplied by the number of resolution bits. For resource-intensive RL algorithms, the increased number of samples may have an impact on computational performance, but thankfully such algorithms are usually the ones that need less samples to learn a good policy.

One could argue that the proposed approach does not truly offer continuous actions the way other approaches do. Partitioning an action range to a resolution of 2^8 or even 2^{16} does provide a large number of actions over that range, but actions are still discrete, not continuous. While in principle this statement is true, one has to take into consideration that precision over a continuous-action range is practically limited due to hardware constraints, representational capacities, numerical errors, time discretization, etc. Our approach allows for approximating any continuous-action range to arbitrary precision with modest increase in computational cost.

In conclusion, this paper introduced Binary Action Search a generic approach for learning continuous-action control policies. The proposed approach offers several attractive features and can be used in conjunction with any RL algorithm. It is our belief that the proposed scheme will enable RL researchers to broaden their application domains and extend their favorite RL algorithms to a variety of practical real-world control problems.

Acknowledgments

The authors would like to thank Damien Ernst for sharing the code for the Extremely-Randomized Trees and the Car on the Hill and Bicycle domains, and the anonymous reviewers for their helpful comments. This work was partially supported by the Marie Curie International Reintegration Grant MCIRG-CT-2006-044980 within the EU FP6.

References

Baird, L. C., & Klopf, A. H. (1993). *Reinforcement learning with high-dimensional, continuous actions* (Technical Report WL-TR-93-1147). Wright Laboratory.

Ernst, D., Geurts, P., & Wehenkel, L. (2005). Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, *6*, 503–556.

Gaskett, C., Wettergreen, D., & Zelinsky, E. (1999). Q-learning in continuous state and action spaces. *Proceedings of the 12th Australian Joint Conference on Artificial Intelligence* (pp. 417–428).

Gross, H. M., Stephan, V., & Krabbes, M. (1998). A neural field approach to topological reinforcement learning in continuous action spaces. *Proceedings of the IEEE Intl Joint Conference on Neural Networks* (pp. 1992–1997).

Kimura, H., & Kobayashi, S. (1998). Reinforcement learning for continuous action using stochastic gradient ascent. *Proceedings of the 5th Intl Conference on Intelligent Autonomous Systems* (pp. 288–295).

Lagoudakis, M. G., & Parr, R. (2003). Least-squares policy iteration. *Journal of Machine Learning Research*, *4*, 1107–1149.

Lazaric, A., Restelli, M., & Bonarini, A. (2008). Reinforcement learning in continuous action spaces through sequential monte carlo methods. In *Advances in neural information processing systems 20*, 833–840.

Millán, J. D. R., Posenato, D., & Dedieu, E. (2002). Continuous-action Q-learning. *Machine Learning*, *49*, 247–265.

Pazis, J., & Lagoudakis, M. G. (2009). Learning continuous-action control policies. *Proceedings of the IEEE Intl Symposium on Adaptive Dynamic Programming and Reinforcement Learning* (pp. 169–176).

Peters, J., & Schaal, S. (2006). Policy gradient methods for robotics. *Proceedings of the IEEE/RSJ Intl Conference on Intelligent Robots and Systems* (pp. 2219–2225).

Prokhorov, D., & Wunsch, D. (1997). Adaptive critic designs. *IEEE Trans. on Neural Networks*, *8*, 997–1007.

Riedmiller, M. (1997). Application of a self-learning controller with continuous control signals based on the DOE-approach. *Proceedings of the European Symposium on Neural Networks* (pp. 237–242).

Sallans, B., & Hinton, G. E. (2004). Reinforcement learning with factored states and actions. *Journal of Machine Learning Research*, *5*, 1063–1088.

Santamaría, J. C., Sutton, R. S., & Ram, A. (1998). Experiments with reinforcement learning in problems with continuous state and action spaces. *Adaptive Behavior*, *6*, 163–218.

Strösslin, T., & Gerstner, W. (2003). Reinforcement learning in continuous state and action space. *Proceedings of the Intl Conference on Artificial Neural Networks*.

Sutton, R., & Barto, A. (1998). *Reinforcement learning: An introduction*. The MIT Press.

Touzet, C. (1997). Neural reinforcement learning for behaviour synthesis. *Robotics and Autonomous Systems*, *22*, 251–281.

Wang, H., Tanaka, K., & Griffin, M. (1996). An approach to fuzzy control of nonlinear systems: Stability and design issues. *IEEE Trans. on Fuzzy Systems*, *4*, 14–23.