Directed Policy Search using Relevance Vector Machines

Ioannis Rexakis and Michail G. Lagoudakis
Department of Electronic and Computer Engineering
Technical University of Crete
Chania 73100 Greece
Email: {yr, lagoudakis}@intelligence.tuc.gr

***Abstract*—Several recent learning approaches based on approximate policy iteration suggest the use of classifiers for representing policies compactly. The space of possible policies, even under such structured representations, is huge and must be searched carefully to avoid computationally expensive policy simulations (rollouts). In our recent work, we proposed a method for directed exploration of policy space using support vector classifiers, whereby rollouts are directed to states around the boundaries between different action choices indicated by the separating hyperplanes in the represented policies. While effective, this method suffers from the growing number of support vectors in the underlying classifiers as the number of training examples increases. In this paper, we propose an alternative method for directed policy search based on relevance vector machines. Relevance vector machines are used both for classification (to represent a policy) and regression (to approximate the corresponding relative action advantage function). Exploiting the internal structure of the regressor, we guide the probing of the state space only to critical areas corresponding to changes of action dominance in the underlying policy. This directed focus on critical parts of the state space iteratively leads to refinement and improvement of the underlying policy and delivers excellent control policies in only a few iterations, while the small number of relevance vectors yields significant computational time savings. We demonstrate the proposed approach and compare it with our previous method on standard reinforcement learning domains.**

I. Introduction

The goal in sequential decision making is to find good policies for selecting actions in order to achieve a long term reward; such problems are typically modeled as Markov Decision Processes (MDPs) [1]. In reinforcement learning, the goal of finding a good policy must be accomplished through interaction with an unknown process typically modeled as an MDP. In either case, good deterministic action policies over the state space of the process can be approximately represented using (multi-class) classifiers; each action is viewed as a distinct class and the states are the instances to be classified. In addition, such policies for common domains are not arbitrary, but rather exhibit significant structure. Several recent learning approaches based on the approximate policy iteration framework suggest the use of classifiers for capturing this structure and representing policies compactly [2]–[13]. Such classifiers can be learned using appropriate training data sets that reveal the desired action choice over a finite set of states. The most attractive benefit of this approach is the elimination of value function representation and the focus instead directly on policy learning. While it is known [14] that it is easier to represent a policy, rather than a value function, the full potential of this reinforcement-learning-through-classification approach for various learning problems has barely been explored.

The space of possible policies, even under such structured representations, is huge and needs to be explored carefully to avoid computationally expensive simulations (rollouts). Our work aims at providing guidance on how to select the subset of the state space where the improved policy is probed to form the training set for the classification problem. This aspect has been given little attention in the past, nevertheless it plays a crucial role, considering that each probe requires a significant amount of resources (simulation) and therefore they better be focused on critical states which can potentially lead to policy improvement. In our previous work [12], we proposed a method for directed exploration of policy space using support vector machines. In this paper, motivated by some limitations of our previous approach, we propose a new method that relies on relevance vector machines. We use a collection of binary relevance vector classifiers to represent policies, whereby each of these classifiers corresponds to a single action and captures the parts of the state space where this action dominates over the other actions. A relevance vector regressor is trained on readily available data to capture an estimation of the importance of each state while improving upon the current policy. The policy is subsequently improved by probing the state space at a set of states sampled carefully using the regressor. This directed focus on critical parts of the state space iteratively leads to the gradual refinement and improvement of the underlying policy, while making conservative use of rollouts. The method proposed in this paper outperforms our previous method for direct policy search. We demonstrate our approach on two standard reinforcement learning domains: inverted pendulum and mountain car.

The remainder of the paper is organized as follows. Section II provides the necessary background and Section III reviews reinforcement learning as classification. In Section IV we present our approach on how to direct policy search through careful selection of probes. Section V describes the two benchmarks domains on which we validate our proposed algorithm. Section VI presents experimental results demonstrating the effectiveness of our approach on these domains and finally Section VII discusses our results and concludes.

II. Background

A *Markov Decision Process* (MDP) [1] 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 a finite set of actions, $P(s'|s,a)$ is a Markovian transition model denoting the probability of a transition to state s' when taking action a in state s, $R(s,a)$ is a non-negative reward function indicating the expected reward for taking action a in state s, $\gamma \in (0,1]$ 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 at state s. The value $V_\pi(s)$ of a state s under a policy π is the expected total discounted reward when the process begins in state s and all decisions at all steps are made according to policy π:

$$V_\pi(s) = E_{s_t \sim P}\left[\sum_{t=0}^{\infty} \gamma^t R\big(s_t, \pi(s_t)\big) \Big| s_0 = s\right] .$$

The goal of the decision maker is to find an optimal policy π^* that maximizes the expected total discounted reward from the initial state distribution D:

$$\pi^* = \arg\max_\pi E_{s\sim D; s_t \sim P}\left[\sum_{t=0}^{\infty} \gamma^t R\big(s_t, \pi(s_t)\big) \Big| s_0 = s\right] .$$

It is well-known that 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 given the full MDP model [15].

In reinforcement learning, the learner interacts with an unknown process assumed to be 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, thus experience comes in the form of (s, a, r, s') tuples. In many cases, it is further assumed that the learner has the ability to reset the process to any arbitrary state s [16]. This amounts to having access to a generative model of the process (a simulator) from where the learner can draw arbitrarily many times a next state s' and a reward r for performing any given action a in any given state s. Several algorithms have been proposed for learning good policies from samples [16], [17].

III. Reinforcement Learning as Classification

A key distinction among reinforcement learning algorithms relies on whether they are based on representing value functions, policies, or both. Value-function-based algorithms have received much criticism in recent years due to difficulties associated with the estimation and representation of value functions. Many learning problems of interest exhibit non-linear and non-smooth value functions that can hardly be compactly represented. On the other hand, advocates of direct policy learning have employed parametric representations that differ little from their value-function-representation counterparts. Most of them rely on representations of stochastic policies which take the form of a softmax over a parametric real-valued function (similar to a value function).

A recent trend in policy learning [2]–[13] attempts to exploit the generalization abilities of modern classification technology under the assumption that optimal or good deterministic policies for real learning problems are not arbitrarily complex, but rather exhibit a high degree of structure. It should, therefore, be plausible to learn a good policy using only a small set of training data consisting of selected states over the state space labeled with the actions that are deemed to be best in those states. To illustrate the learning process under such representations, we briefly review the *Rollout Classification Policy Iteration* (RCPI) algorithm [2]. The key idea behind RCPI is to cast the problem of policy learning as a classification problem. Finding a good policy becomes equivalent to finding a classifier that generalizes well over the state space and maps states to "good" actions, where the goodness of an action is measured in terms of its contribution to the long-term goal of the agent. The state-action value function Q_π

$$Q_\pi(s,a) = R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) V_\pi(s') ,$$

provides such a measure given a fixed base policy π; the action that maximizes Q_π in state s is a "good" action in that state, whereas any action with strictly smaller Q_π value is a "bad" one. The policy π' formed by choosing maximizing actions

$$\pi'(s) = \arg\max_{a \in \mathcal{A}} Q_\pi(s,a)$$

is guaranteed to be at least as good as π, if not better. A training set for π' could be easily formed, if the Q_π values for all actions were available for a subset of states. The Monte-Carlo estimation technique of *rollouts* [16] provides a way of accurately estimating Q_π at any given state-action pair (s,a) without requiring an explicit representation of the value function. A rollout for (s,a) amounts to simulating a trajectory of the process beginning from state s, choosing action a for the first step and actions according to policy π thereafter up to a certain horizon H, and computing the total discounted reward along this trajectory. The observed total discounted reward is averaged over a number of rollouts to yield an accurate estimate of $Q_\pi(s,a)$. Thus, using a sufficient amount of rollouts, it is possible to create a training example of the improved policy in state s. A collection of such examples over a finite set of states forms a valid training set for the improved policy π' over any base policy π.

The goal of the learner is not only to improve a policy, but rather to find a good or optimal policy, therefore RCPI employs an approximate policy iteration scheme as described in Algorithm 1. At each iteration, a new policy/classifier is produced using training data obtained by rolling out the previous policy on a generative model of the process. Beginning with any initial policy π_0, at each iteration k a training set over a subset of states S_k is formed by querying the rollout procedure

Algorithm 1 Rollout Classification Policy Iteration (RCPI)

Input: policy π_0, trials K, horizon H

$k \leftarrow -1$
repeat
  $k \leftarrow k+1$
  select the rollout states $S_{k+1} \subseteq \mathcal{S}$
  $T_{k+1} = \varnothing$
  for (each $s \in S_{k+1}$) **do**
    for (each $a \in \mathcal{A}$) **do**
      estimate $Q_{\pi_k}(s,a)$ using K rollouts of length H
    end for
    if (a dominating action a^* exists in state s) **then**
      $T_{k+1} = T_{k+1} \cup \{(s,a^*)^+\}$ for dominating actions
      $T_{k+1} = T_{k+1} \cup \{(s,a)^-\}$ for dominated actions
    end if
  end for
  π_{k+1} = TRAINCLASSIFIER(T_{k+1})
until (π_{k+1} is not better than π_k)
return π_k

to identify dominating actions in the states of S_k. Notice that the training set contains both positive $(+)$ and negative $(-)$ examples for each state, where a clear domination is found. A new classifier is trained using these examples to yield an approximate representation of the improved policy over the previous one. This cycle is then repeated until a termination condition is met. Given the approximate nature of this policy iteration, the termination condition cannot rely on convergence to a single optimal policy. Rather, it terminates when the performance of the new policy (measured via simulation) does not exceed that of the previous policy. The empirical expected total discounted reward from D obtained from and averaged over multiple runs is used as the policy performance criterion.

The RCPI algorithm yielded promising results in the pendulum and the bicycle domains using Support Vector Machines (SVMs) and Multi-Layer Perceptrons (MLPs) as classifiers. A similar algorithm by Fern et al. [3], [5] yielded satisfying results in seven planning domains mainly from the AIPS-2000 planning competition using Decision Lists as the underlying classifier. Subsequent work on reinforcement learning as classification focused on formal reductions to classification [4], effective management of rollouts [6], [11], effective classifier-based representations of policies [7], theoretical analysis of the proposed algorithms [8]–[10], directed selection of rollout states [12], and reduction of rollout-based return estimation bias with value function estimates [13].

IV. Directed Policy Search with RVMs

Previous studies identified sensitivities related to the distribution of states in S_k, however, barely any of them examined closely the structure of each learned policy π_k and the possibility of exploiting this structure for guiding the selection of states in S_{k+1}. A common choice is to pick states uniformly from the state space, but this is unlike to work adequately in high-dimensional spaces. A notable exception is the idea of sampling states from the γ-discounted future state distribution of the improved policy [2], [3], however this is based purely on process dynamics ignoring any information contained in the structure of the represented policies themselves.

Our recent work [12] took a first step in the direction of exploiting policy information by considering Support Vector Machine (SVM) classifiers that reveal the internal structure of policy π_k, in particular, the boundaries between different action choices over the state space. These boundaries, defined by the support vectors and the corresponding separating hyperplanes, are used to identify critical areas of the state space for probing and to guide the selection of states in S_{k+1} towards these areas. The intuition behind this idea is the observation that in typical control learning problems each policy improvement step refines these boundaries and therefore it is important to probe for dominating actions in states around, but not on, these boundaries. We chose to identify such states by sampling from the ones lying along the perpendicular projections of the support vectors on the separating border. The net effect of this method is that a much smaller number of such selected rollout states suffices to sustain the gradual policy improvement during policy iteration and the simulation cost (rollouts) reduces by an order of magnitude. While effective, this method suffers from the growing number of support vectors in the underlying classifiers as the number of training examples increases. In turn, the increase in the number of support vectors slows down the projection operations (solution of a non-linear system) and yields significant overhead.

The Relevance Vector Machine (RVM) [18] is a kernel-based learning machine that has the same functional form as the Support Vector Machine (SVM). It is formed as a linear combination of data-centered basis functions, called relevance vectors, which are in general non-linear. The RVM has been shown to provide equivalent and often superior results to the SVM both in generalization ability and sparseness of the model. The number of relevance vectors in an RVM is typically low and not proportional to the number of samples. For these reasons, we turned our attention to RVM classifiers as a promising alternative candidate for representing policies. At the same time, deeper inspection and consideration of RVMs gave rise to a new directed selection method for rollout states that avoids the costly projections of the previous one.

Probing a particular state s for the improved policy over a base policy π boils down to estimating the Q_π values for all actions in that state using rollouts and identifying the dominating action(s) (if any). Say that these estimated values are $\hat{Q}_\pi(s,a)$, $a \in \mathcal{A}$. In order to include state s in the training set, we need to identify at least one dominating action, whose value exceeds significantly the value of some other action. To quantify this difference we use the ΔQ quantity:

$$\Delta Q(s,a) = \frac{\max_{a' \in \mathcal{A}}\{\hat{Q}_\pi(s,a')\} - \hat{Q}_\pi(s,a)}{\max_{a' \in \mathcal{A}}\{\hat{Q}_\pi(s,a')\}}$$

$$\Delta Q(s) = \max_{a \in \mathcal{A}} \Delta Q(s,a)$$

Since R is non-negative, $Q_\pi(s,a)$ and $\hat{Q}_\pi(s,a)$ are non-negative. When $\max_{a' \in \mathcal{A}}\{\hat{Q}_\pi(s,a')\} = 0$, $\Delta Q(s,a) = 0$ by definition. Clearly, $\Delta Q(s,a)$ and $\Delta Q(s)$ are non-negative. If $\Delta Q(s) > \epsilon$, then any action $a^* = \arg\max_{a' \in \mathcal{A}}\{\hat{Q}_\pi(s,a')\}$ that maximizes $\hat{Q}_\pi$ in state s is considered *dominating* and a pair (s,a^*) is inserted in the training set as a positive $(+)$ example. Any action a with $\Delta Q(s,a) > \epsilon$ is considered *dominated* and a pair (s,a) is inserted in the training set as a negative $(-)$ example. Note that some actions may be neither dominating, nor dominated; no training data will be produced for such actions. States with $\Delta Q(s) \leq \epsilon$ do not yield any training data at all. This simple dominance criterion is sufficient in most cases for dealing with estimation noise. We should stress that our approach requires the estimation of action values at few isolated points (rollout states) only; these values are obtained as unbiased estimates from Monte-Carlo simulation (policy rollouts) and can be estimated to any desired accuracy provided sufficient simulation time.

The relative action advantage function $\Delta Q(s)$, as we may call it, is defined over the entire state space and captures important information about the underlying improved policy. It should be noted that ΔQ is non-negative; it takes high values within areas dominated by some action and a zero value only at the boundaries, where no action dominates. Focusing the probes for the improved policy at states with high values of ΔQ yields little information, because it only reinforces a more or less clear action dominance with known locality [9]. On the other hand, probing at states with a value of ΔQ close to zero will likely result in a blurring of action advantage values with estimation noise and rarely will it reveal any useful information about action dominance. Our experience with classification-based policy representation and learning suggests that the important areas of the state space that need to be probed lie close to and around, but not on, the current policy boundaries, in other words at parts where ΔQ is rapidly ascending or descending. One way to characterize these areas is to look at the magnitude of the gradient vector of ΔQ. But, where can one find this gradient vector, when even ΔQ itself is not analytically known?

Our approach towards estimating this gradient vector is a simple solution that takes advantage of the already available information to minimize overhead. At the end of each iteration, the estimated values of ΔQ that were used to form the training set T_{k+1} for the next classifier are still available. We use these values to form another training set T'_{k+1} of $\big(s, \Delta Q(s)\big)$ pairs in order to generalize and approximate the function ΔQ over the entire state space by regression. We train an RVM regressor on these data, not only because we seek to exploit the same technology and benefit from the advantages it offers, but also, more importantly, because the learned function can be analytically differentiated to yield the desired gradient vector. More specifically, the RVM regressor approximation of the relative action advantage function ΔQ takes the form

$$q(s) = \sum_{i=1}^{\ell} \alpha_i \Delta Q(s_i) k(s_i, s) + b \ ,$$

Algorithm 2 SAMPLE: Sampling Rollout States

Input: number of samples M, number of particles Z, regressor training set T', matrix Σ, weigh function g

for $j = 1$ to Z **do**
  sample $\big(s, \Delta Q(s)\big) \sim T'$ uniformly
  $s_j = s + \mathcal{N}(\mathbf{0}, \Sigma)$
  $w_j = \|g(s_j)\|_2$
end for
normalize w_j's

$S = \emptyset$, $r \sim$ uniform$(0, M^{-1})$, $c = w_1$, $j = 1$
for $m = 1$ to M **do**
  $u = r + (m-1) \times M^{-1}$
  while $u > c$ **do**
    $j = j + 1$
    $c = c + w_j$
  end while
  $S = S \cup \{s_j\}$
end for
return S

where the s_i's, $i = 1, \ldots, \ell$, are the relevance vectors of the regressor, $\Delta Q(s_i)$ are their target values from the corresponding training pairs, and α_i, b are the parameters of the RVM regressor. For the Gaussian (radial basis functions) kernel[1],

$$k_g(s', s) = \exp\big(-\beta \parallel s' - s \parallel^2\big), \quad \beta > 0,$$

the gradient of $q(s)$ can be analytically derived as

$$g(s) = 2\beta \sum_{i=1}^{\ell} \alpha_i \Delta Q(s_i) k_g(s_i, s)(s_i - s) \ .$$

This way we estimate the desired gradient vector, practically at no additional cost, other than a single RVM regression.

The next obstacle we have to overcome is the identification of areas of the state space where the magnitude of the gradient vector is large and direct sampling of rollout states to those areas. Despite the availability of the gradient in closed form, an analytical solution for the maxima poses a hard non-linear problem. Additionally, we have to take into account that the set of rollout states becomes more and more focused over iterations and, therefore, the RVM regressor ends up being trained on points representing only a small part of the state space. As a result the regressor function (and therefore its gradient) cannot be trusted in areas away from the rollout states of the previous iteration. To address these problems, we apply a resampling procedure inspired by the resampling operations in a particle filter [19]. Initially, we sample a large number of candidate states (particles), as follows: pick uniformly in random one point from the training set T'_{k+1} and add to it zero-mean normal noise to obtain a new sample from its neighborhood. This step ensures that all candidate states lie in areas where the regressor can be trusted. Next,

[1]Note that other kernels can be used as long as they can be differentiated.

we weigh each particle using the magnitude (L_2 norm) of the gradient vector. Finally, we apply a resampling procedure to keep only the desired number of rollout states for the next iteration. This step ensures that only those candidate states with large weight (large magnitude of gradient) are promoted. The entire resampling procedure is shown in Algorithm 2. Note that its time complexity is only $\mathcal{O}(Z)$, where Z is the number of particles. If Z is taken to be a multiple of M, the number of rollout states in each iteration, the cost of obtaining the next set of rollout states is only linear in its size.

Now, we can summarize the complete algorithm. Given a policy π_k at iteration k represented as an RVM classifier, the corresponding RVM regressor is used to select a new focused subset of states S_{k+1} at which the improved policy π_{k+1} will be probed. Each probe will yield pairs of data in the training set T_{k+1} for the next classifier, if domination is detected, and a pair of data in the training set T'_{k+1} of the corresponding regressor. Once the training sets are formed, a new classifier and a new regressor are trained to represent π_{k+1} and the process repeats from the beginning. Clearly, there is no guarantee for monotonic policy improvement in this iterative scheme, however the chances can be increased by repeating some iteration, if no improvement was achieved; since each iteration is randomized, it is likely that another run may produce better results. Therefore, if policy π_{k+1} is not better than π_k (tested through simulation), iteration k is repeated for a maximum of L attempts until an improved policy is found. If all attempts are exhausted without improvement, the algorithm terminates. Note that the RVM regressor is trained only when the policy improves, that is, only once per iteration; it should not change, if an improvement attempt fails. Note also that the first iteration begins with a purely random, but deterministic, policy and therefore S_1 cannot be formed in the way described above, since there is no RVM classifier and regressor for π_0. S_1 is simply a uniformly random selection of states from the entire state space to guarantee full coverage at the beginning[2]. The entire Directed RCPI-RVM (DRCPI-RVM) algorithm is shown in Algorithm 3.

Algorithm 3 DRCPI-RVM: Directed RCPI with RVMs

Input: policy π_0, attempts L, trials K, horizon H, threshold ϵ, size U, points M, particles Z, matrix Σ

$k \leftarrow -1$
repeat
  $k \leftarrow k+1$
  $l \leftarrow 0$
  repeat
    $l \leftarrow l+1$
    if $(k=0)$ **then**
      S_{k+1} = a uniformly random subset of $\mathcal{S}$ of size U
    else
      $S_{k+1} = \textsc{Sample}(M, Z, T'_{k+1}, \Sigma, g_{k+1})$
    end if
    $T_{k+1} = \varnothing$, $T'_{k+1} = \varnothing$
    for (each $s \in S_{k+1}$) **do**
      for (each $a \in \mathcal{A}$) **do**
        estimate $Q_{\pi_k}(s,a)$ using K rollouts of length H
      end for
      if $(\Delta Q(s) > \epsilon)$ **then**
        $T_{k+1} = T_{k+1} \cup \{(s,a^*)^+\}$ for dominating actions
        $T_{k+1} = T_{k+1} \cup \{(s,a)^-\}$ for dominated actions
      end if
      $T'_{k+1} = T'_{k+1} \cup \{(s, \Delta Q(s))\}$
    end for
    π_{k+1} = $\textsc{TrainClassifier}(T_{k+1})$
  until ((π_{k+1} is better than π_k) or $(l=L)$)
  q_{k+1} = $\textsc{TrainRegressor}(T'_{k+1})$
  g_{k+1} = $\textsc{Differentiate}(q_{k+1}(s))$
until (π_{k+1} is not better than π_k)
return π_k

V. Benchmark Domains

We chose to study the proposed algorithm on two standard domains in reinforcement learning: inverted pendulum and mountain car. Both are defined on two-dimensional continuous state spaces, therefore they are appropriate for visualization and inspection.

A. Inverted Pendulum

The *inverted pendulum* problem is to balance a pendulum of unknown length and mass at the upright position by applying forces to the cart it is attached to. Three actions are allowed: left force LF (-50 Newtons), right force RF ($+50$ Newtons), or no force NF (0 Newtons). All three actions are noisy; Gaussian noise with $\mu = 0$ and $\sigma^2 = 10$ is added to the chosen action. The state space of the problem is continuous and consists of the vertical angle θ and the angular velocity $\dot{\theta}$ of the pendulum. The transitions are governed by the non-linear dynamics of the system [20] and depend on the current state and the current (noisy) control u:

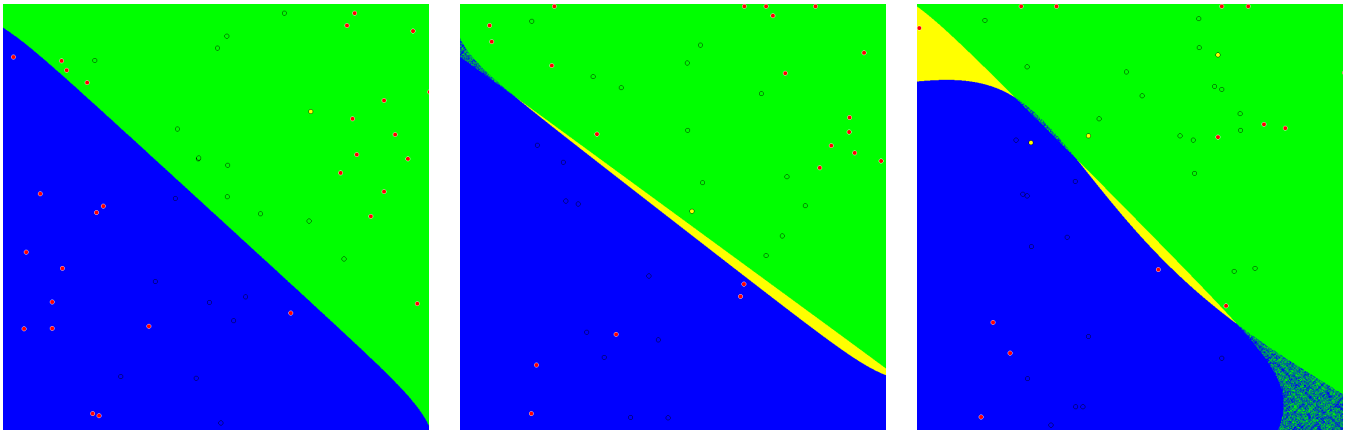
$$\ddot{\theta} = \frac{g\sin(\theta) - \alpha m l (\dot{\theta})^2 \sin(2\theta)/2 - \alpha\cos(\theta)u}{4l/3 - \alpha m l \cos^2(\theta)} \ ,$$

where $g = 9.8\text{m/s}^2$ is the gravity constant, $m = 2.0\text{kg}$ is the mass of the pendulum, $M = 8.0\text{kg}$ is the mass of the cart, $l = 0.5\text{m}$ is the length of the pendulum, and $\alpha = 1/(m+M)$. The control step is 0.1 seconds and the control input is kept constant between successive steps. A reward of 1 is given as long as the angle of the pendulum does not exceed $\pi/2$ in absolute value (the pendulum is above the horizontal line). An angle greater than $\pi/2$ in absolute value signals the end of the episode and a reward of 0. The discount factor of the process is set to 0.95.

B. Mountain Car

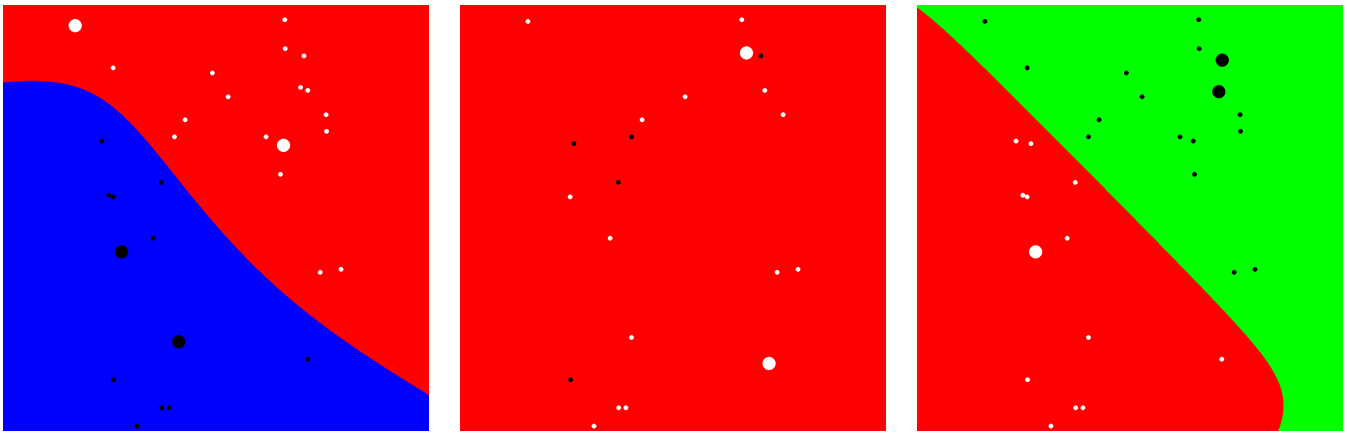
The *mountain car* problem is to drive an underpowered car from the bottom of a valley between two mountains to the top

[2]Without domain knowledge, uniform sampling is the only option to guarantee that no part will remain initially unexplored. In certain cases, domain knowledge could be used to perform a focused non-uniform sampling.



policy π_1 and rollout states S_1 | policy π_2 and rollout states S_2 | policy π_3 and rollout states S_3

Fig. 1. Inverted Pendulum: three successive policies from a typical run (action selection is shown in color: LF-blue, NF-yellow, RF-green) and the corresponding rollout states (shown as little circles colored with the dominating action; those discarded from the training set of the classifier ($\Delta Q(s) \leq \epsilon$) are colored red).



π_3: binary classifier for action LF | π_3: binary classifier for action NF | π_3: binary classifier for action RF

Fig. 2. Inverted Pendulum: the binary RVM classifiers for each action of the final balancing policy π_3 in the experiment of Figure 1 (in class: LF-blue, NF-yellow, RF-green; out of class: red) and the corresponding training set T_3 (positive examples: black, negative examples: white, relevance vectors: bold).

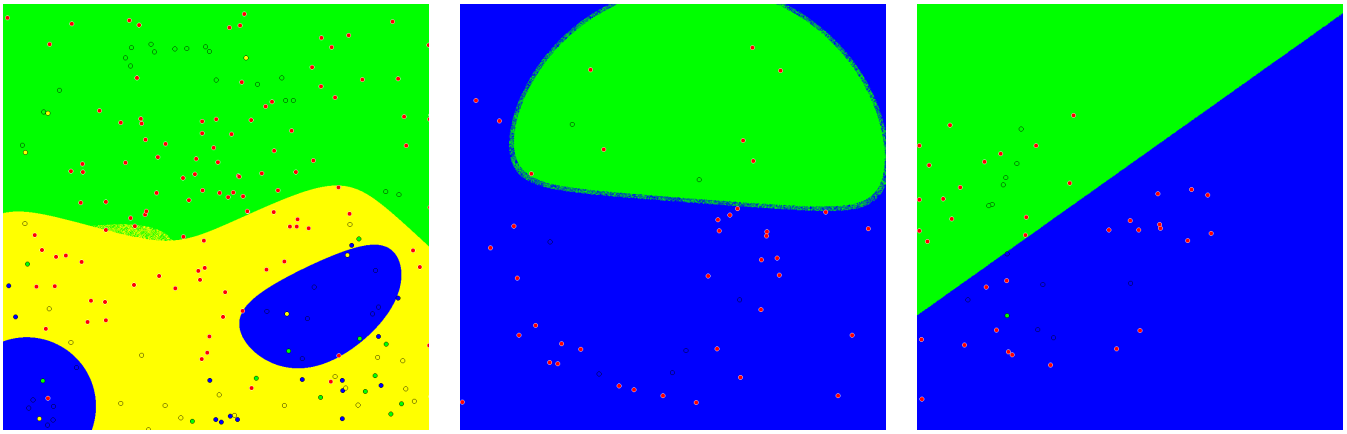
of the mountain on the right. The car is not powerful enough to climb any of the hills directly from the bottom of the valley even at full throttle; it must build enough energy by climbing first to the left (moving away from the goal) and then to the right. Three actions are allowed: forward throttle FT ($+1$), reverse throttle RT (-1), or no throttle NT (0). Gaussian noise with $\mu = 0$ and $\sigma^2 = 0.2$ is added to the chosen action. The state space of the problem is continuous and consists of the position x and the velocity $\dot{x}$ of the car along the horizontal axis. The transitions are governed by the discrete-time non-linear dynamics of the system [17] and depend on the current state $(x(t), \dot{x}(t))$ and the current control $u(t)$:

$$\begin{aligned} x(t+1) &= \textsc{Bound}_x\big(x(t) + \dot{x}(t+1)\big) \\ \dot{x}(t+1) &= \textsc{Bound}_{\dot{x}}\big(\dot{x}(t) + 0.001u(t) - 0.0025\cos(3x(t))\big) \end{aligned},$$

where $\textsc{Bound}_x$ is a function that keeps x within $[-1.2, 0.5]$, while $\textsc{Bound}_{\dot{x}}$ keeps $\dot{x}$ within $[-0.07, 0.07]$. If the car hits the bounds of the position x, the velocity $\dot{x}$ is set to zero. Reward of 0 is given at each step as long as the position of the car is below the right bound (0.5). As soon as the car position hits the right bound, it has reached the goal; the episode ends successfully and a reward of 1 is given. The discount factor of the process is set to 0.99.

VI. Experimental Results

We applied our approach to the two benchmark domains described above to test its effectiveness. All policies were represented using RVM classifiers and RVM regressors with the radial basis function (RBF) kernel (with $\beta = d\sigma^2$, where d is a constant and σ^2 is the standard deviation of the data in the training set) and were implemented using the Dlib package [21]. Note that we did not try to optimize the parameters of the classifier/regressor, since our ultimate goal is to employ simple off-the-shelf classification/regression technology for the benefit of reinforcement learning. The parameters of our algorithm were tested selectively within their range to find operational values. The values used in the experiments reported here were $U = 200$, $M = 50$, $Z = 10M$, $\Sigma = \text{diag}\{0.2\}$, $L = 4$, $K = 50$, $H = 100$, $\epsilon = 0.02$ for all domains, and $d = 2$ for the inverted pendulum



policy π_1 and rollout states S_1 | policy π_2 and rollout states S_2 | policy π_3 and rollout states S_3

Fig. 3. Mountain Car: three successive policies from a typical run (action selection is shown in color: RT-blue, NT-yellow, FT-green) and the corresponding rollout states (shown as little circles colored with the dominating action; those discarded from the training set of the classifier ($\Delta Q(s) \leq \epsilon$) are colored red).

π_3: binary classifier for action RT | π_3: binary classifier for action NT | π_3: binary classifier for action FT

Fig. 4. Mountain Car: the binary RVM classifiers for each action of the final exiting policy π_3 in the experiment of Figure 3 (in class: RT-blue, NT-yellow, FT-green; out of class: red) and the corresponding training set T_3 (positive examples: black, negative examples: white, relevance vectors: bold).

and $d = 5$ for the mountair car. The initial policy π_0 was a random deterministic policy. Each dimension of the state spaces was scaled to $[-1, +1]$.

Figure 1 shows a typical run on the inverted pendulum domain. Data are displayed over the 2-dimensional state space (the horizontal axis is the angle and the vertical axis is the angular velocity). A fully-balancing policy was found in three iterations (unsuccessful improvement attempts are not shown) starting from a random policy (not shown). The first iteration delivered a fairly good policy, which was subsequently improved in the second and third iterations, while the fourth iteration did not improve the already fully-balancing policy. Notice that, after the initial uniform distribution, the rollout states are positioned mostly around the boundary. Figure 2 shows the three binary RVM classifiers representing the final policy π_3. It is worth noting the small number of relevance vectors in each classifier.

Figure 3 shows a typical run on the mountain car domain. Once again, data are displayed over the 2-dimensional state space (the horizontal axis is the position and the vertical axis is the velocity). A successful exiting policy is found in the third iteration, starting from a random policy (not shown). The policies in the first and second iterations were exiting as well, but they needed many more steps to succeed. The fourth iteration did not improve performance (unsuccessful improvement attempts are not shown) and the algorithm terminated. Figure 4 shows the three binary RVM classifiers representing the final policy π_3. Note that each classifier is defined by only two relevance vectors.

Table I shows collective results and statistics from multiple runs. Each row represents averages from 100 different executions with the same settings, but with different random seeds. These data offer a comparison between the proposed algorithms RCPI-RVM (vanilla RCPI with RVM policy representation) and Directed RCPI (DRCPI-RVM) based on relevance vector machines against the corresponding algorithms RCPI-SVM and Directed RCPI (DRCPI-SVM) based on support vector machines from our previous work [12]. For fairness, we added the multiple improvement attempts in RCPI, so that the only difference between the algorithms is the choice of

TABLE I
COLLECTIVE RESULTS (100 RUNS) AND COMPARISON OF ALGORITHMS IN TERMS OF COMPUTATIONAL AND LEARNING PERFORMANCE.

	Kernel	Algorithm	States	Simulation	Rollouts	Attempts	Time	Return	Success
Pendulum	RBF	RCPI-SVM	200	4182303	1336	6.68	86.60	20.00	95.78%
	RBF	RCPI-RVM	200	725734	1206	6.03	16.71	19.99	95.78%
	RBF	DRCPI-SVM	200/50	2122569	469	6.34	33.10	20.00	92.88%
	RBF	DRCPI-RVM	200/50	581715	401	5.03	9.02	19.99	98.05%
Mountain	RBF	RCPI-SVM	200	14714292	1690	8.45	65.10	0.42	98.89%
	RBF	RCPI-RVM	200	3170390	1826	9.13	29.12	0.26	99.00%
	RBF	DRCPI-SVM	200/50	6541460	585	7.39	29.70	0.29	88.45%
	RBF	DRCPI-RVM	200/50	5606046	540	7.02	31.66	0.18	94.23%

the rollout states in terms of quantity and location and, of course, the representation technology (RVM vs. SVM). All other settings were kept identical for all runs. The Simulation tab shows the amount of simulation steps needed for each run, while the Rollouts tab shows the number of rollouts executed in each run. The Attempts tab shows the number of improvement attempts (the number of iterations is less than or equal to that) before termination and the Time tab shows the real time (seconds) taken by each run. Finally, the Return and Success tabs show the total expected discounted reward (measured by policy rollout from the initial state) and the success rate respectively of the final learned policy. A successful run corresponds to 3000 steps (5 minutes) of balancing in the inverted pendulum domain and exiting from the valley in less than 3000 steps in the mountain car domain. Clearly, DRCPI-RVM yields significant savings in terms of rollouts and simulation compared to the other algorithms, while delivering policies of comparable performance. Despite the extra computations required by DRCPI-RVM, there were also significant savings in real computation time, which implies that the amount of simulation is the main cost factor.

VII. CONCLUSION

We have presented a scheme for directing classifier-based policy search. Our scheme exploits the properties of relevance vector machines to identify the critical parts of the state space where probing should focus to improve an existing policy. Our results indicate significant savings in simulation time (rollouts) and generation of very good policies in only a few iterations, without penalty in computational cost.

In our work, we have arrived to an approximation of the relative action advantage function in a differentiable closed form. An interesting direction of future research would be the analytical identification of the important areas of the state space for probing the improved policy, in place of the sampling method we proposed. An analytical approach could yield additional savings, revealing at the same time more information about the internal structure of the learned policies.

REFERENCES

[1] M. L. Puterman, *Markov Decision Processes – Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., 1994.

[2] M. G. Lagoudakis and R. Parr, "Reinforcement learning as classification: Leveraging modern classifiers," in *Proceedings of the 20th International Conference on Machine Learning (ICML)*, 2003, pp. 424–431.

[3] A. Fern, S. Yoon, and R. Givan, "Approximate policy iteration with a policy language bias," *Advances in Neural Information Processing Systems (NIPS)*, vol. 16, no. 3, 2004.

[4] J. Langford and B. Zadrozny, "Relating reinforcement learning performance to classification performance," in *Proceedings of the 22nd International Conference on Machine Learning (ICML)*, 2005, pp. 473–480.

[5] A. Fern, S. Yoon, and R. Givan, "Approximate policy iteration with a policy language bias: Solving relational Markov decision processes," *Journal of Artificial Intelligence Research*, vol. 25, pp. 75–118, 2006.

[6] C. Dimitrakakis and M. G. Lagoudakis, "Rollout sampling approximate policy iteration," *Machine Learning*, vol. 72, no. 3, pp. 157–171, 2008.

[7] I. Rexakis and M. G. Lagoudakis, "Classifier-based policy representation," in *Proceedings of the 7th IEEE International Conference on Machine Learning and Applications (ICMLA)*, 2008, pp. 91–98.

[8] C. Dimitrakakis and M. G. Lagoudakis, "Algorithms and bounds for rollout sampling approximate policy iteration," in *Proceedings of the 8th European Workshop on Reinforcement Learning (EWRL)*, 2008, pp. 27–40.

[9] E. Rachelson and M. G. Lagoudakis, "On the locality of action domination in sequential decision making," in *Proceedings of the 11th International Symposium on Artificial Intelligence and Mathematics (ISAIM)*, 2010.

[10] A. Lazaric, M. Ghavamzadeh, and R. Munos, "Analysis of a classification-based policy iteration algorithm," in *Proceedings of the 27th International Conference on Machine Learning (ICML)*, 2010, pp. 607–614.

[11] V. Gabillon, A. Lazaric, and M. Ghavamzadeh, "Rollout allocation strategies for classification-based policy iteration," in *Proceedings of the ICML Workshop on Reinforcement Learning and Search in Very Large Spaces*, 2010.

[12] I. Rexakis and M. G. Lagoudakis, "Directed exploration of policy space using support vector classifiers," in *Proceedings of the IEEE International Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*, 2011, pp. 112–119.

[13] V. Gabillon, A. Lazaric, M. Ghavamzadeh, and B. Scherrer, "Classification-based policy iteration with a critic," in *Proceedings of the 28th International Conference on Machine Learning (ICML)*, 2011, pp. 1049–1056.

[14] C. W. Anderson, "Approximating a policy can be easier than approximating a value function," Department of Computer Science, Colorado State University, Tech. Rep. CS-00-101, 2000.

[15] R. A. Howard, *Dynamic Programming and Markov Processes*. The MIT Press, 1960.

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

[17] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*. Cambridge, Massachusetts: The MIT Press, 1998.

[18] M. E. Tipping and A. Faul, "Fast marginal likelihood maximisation for sparse Bayesian models," in *Proceedings of the 9th International Workshop on Artificial Intelligence and Statistics*, 2003, pp. 3–6.

[19] A. Doucet, N. De Freitas, and N. Gordon, *Sequential Monte Carlo Methods in Practice*. Springer Verlag, 2001.

[20] H. O. Wang, K. Tanaka, and M. F. Griffin, "An approach to fuzzy control of nonlinear systems: Stability and design issues," *IEEE Transactions on Fuzzy Systems*, vol. 4, no. 1, pp. 14–23, 1996.

[21] D. E. King, "Dlib-ml: A machine learning toolkit," *Journal of Machine Learning Research*, vol. 10, pp. 1755–1758, 2009. [Online]. Available: dlib.net/ml.html