Proceedings of the 1st International Workshop on Robot Learning and Planning (RLP 2016)
in conjunction with 2016 Robotics: Science and Systems
June 18, 2016 – Ann Arbor, Michigan, USA

Robust Reinforcement Learning with Relevance Vector Machines

Minwoo Lee
Computer Science Department
Colorado State University
Fort Collins, Colorado 80523–1873
Email: lemin@cs.colostate.edu

Charles W. Anderson
Computer Science Department
Colorado State University
Fort Collins, Colorado 80523–1873
Email: anderson@cs.colostate.edu

***Abstract*—Function approximation methods, such as neural networks, radial basis functions, and support vector machines, have been used in reinforcement learning to deal with large state spaces. However, they can become unstable with changes in the samples state distributions and require many samples for good estimations of value functions. Recently, Bayesian approaches to reinforcement learning have shown advantages in the exploration-exploitation tradeoff and in lower sampling costs. This paper proposes a novel reinforcement learning framework that uses the relevance vector machines (RVM) as a function approximator, which incrementally accumulates knowledge from experiences based on the sparseness of the RVM model. This gradual knowledge construction process increases the stability and robustness of reinforcement learning by preventing possible forgetting. In addition, RVM's low sampling costs improve the learning speed. The approach is examined in the popular benchmark problems of pole-balancing and mountain car.**

I. Introduction

To solve a reinforcement learning (RL) problem, an agent must develop a good estimate of the sum of future reinforcements, called the value of the current state and action. A common problem in RL in continuous actions and states is that there is an infinite number of state and action pairs. Successful RL often requires fine discretization, but this can result in the need for a prohibitive amount of experience [27]; high-dimensional discrete representations result in a curse of dimensionality. To overcome the limited amount of experience available in practical RL tasks, an agent must be able to generalize based on limited experience. To do so, the value function or action-value (Q) function must be approximated using parameterized, continuous representations.

A variety of function approximation methods have been studied in RL. Cellular approximations such as CMAC tile-coding [1] and radial basis function [31, 3] have been applied to various RL problems. Most popularly, neural networks have been selected as function approximators for backgammon [32], robot shaping [8], agent survival game [17], robot walking [5], robot soccer [29], and octopus arm control [10, 16]. Recently, Mnih, et al., [19, 20] successfully applied convolution neural networks to Atari games and overcame the challenges of applying deep networks to RL problems—small numbers of samples with delayed or noisy reinforcements, dependent data samples from sequences of state transitions, and different sample distributions for diverse, new behaviors. Anderson, et al., [2] suggested that RL with neural networks as function approximators can be more efficient by pretraining deep networks without giving any information about the goal tasks. They trained deep networks to predict state changes and showed that the weights learned for predicting state changes reduced the RL training time. Silver, et al., [28] maintained three different deep networks for policy and value evaluation and trained them with a combination of supervised learning and reinforcement learning. They applied them to play Go successfully.

Online support vector regression (SVR) has been applied as a value function approximator [14, 15]. As online SVR removes useless support vectors, it provides good generalization and performance on solving RL problems. Even with the successful applications of various function approximations, problems with the stability of learning remain. Furthermore, SVR is well-known for its impracticality caused by the increasing number of support vectors as the number of training samples grows [40].

Recently, Bayesian learning models have shown some success and efficiency in reinforcement learning. Bayesian Q-learning [7], Gaussian process temporal difference learning (GPTD) [9, 10, 26], and linear Bayesian reinforcement learning [38] have been applied to estimate value functions. Unlike classical RL, Bayesian RL suggests feasible solutions to the exploration-exploitation tradeoffs [25, 7]. Moreover, Bayesian RL can choose samples to learn especially when the sampling cost is expensive [12].

Relevance vector machines (RVMs) [33, 34, 35] can be viewed as a Bayesian regression model that extends Gaussian processes (GP), and also we can consider RVMs as an extension of support vector machine (SVMs). With respect to the first point, RVMs can represent more flexible models by providing separate priors for each dimension. With respect to the second point, while SVMs suffer from the explosive growth of support vectors, RVMs provide sparse solutions by capturing the significant mass. To accommodate those advantages of Bayesian RL and to overcome the limitations of SVR, we use RVMs as function approximators.

In this paper, we suggest a novel RL framework that utilizes RVMs as function approximators. Observing that the sparse relevance vectors capture the importance of the value esti-

mations, we design a learning framework that incrementally builds up a knowledge base from the key experiences. To maintain learning efficiency and to prevent forgetting, we extend Riedmiller's fitted Q [24] and train an RVM function approximator with a mini-batch of samples and transfer relevance vectors after examining the learned model. This approach can be comparable to directed rollout classification policy iteration (DRCPI-RVM) [23] in that both adopt RVMs to overcome the limitations of the growing number of the support vectors (SVs), but the proposed approach focuses on value iteration with gradual shaping of the data-centered features. By comparing the learning performance with other function approximations such as neural networks, SVR and GP, we examine the efficiency of the framework. Also, we examine how the learned relevance vectors (RVs) capture the significant mass of the agent's optimal behavior. The major contribution of this paper is this new function approximation in a constructive framework for various applications. Also, with sparse solutions, it can provide computational benefits by reducing the required number of samples to explore. Most significantly, RV placement analysis facilitates an understanding of the solutions that can lead to further investigation of algorithms and problems in the discovery of an unknown optimal policy. Finally, empirically we observe the improvement of learning speed, which is caused by augmented knowledge (or experiences).

In Section II, we introduce the reinforcement learning and its terms and notation along with well-known approaches as baseline algorithms. In Section III, we briefly introduce the relevance vector machine. Section IV describes the proposed framework, RVM-RL. Benchmark experiments and results of the proposed approach are summarized in Section V, and follow-up discovery about RV analysis is discussed in Section VI.

II. Reinforcement Learning (RL)

Reinforcement learning problems that involve the interaction with an environment can be modeled as Markov decision processes (MDP). MDP is defined as a tuple $(S, A, P^a_{ss'}, R, \gamma)$, where for each time step $t = 0, 1, 2, \ldots$, with probability $P^a_{ss'}$, action $a_t \in A$ in state $s_t \in S$ transitions to state $s_{t+1} = s' \in S$, and the environment emits a reward $r_{t+1} \in R$.

In an environment specified by the given MDP, a reinforcement learning agent aims to maximize the reward in the long run. For control problems, to estimate how good an action is in a given state, we can define the action value method for policy π, $Q^\pi(s, a)$, as expected sum of rewards:

$$Q^\pi(s,a) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r_{t+1} | s_t = s, a_t = a, \pi]$$

where $\gamma \in (0, 1]$ is a discounting factor. To see the relationship to the next state, the action-value function Q can be rewritten with Bellman equation:

$$Q^\pi(s,a) = \mathbb{E}[r_{t+1} + \gamma V^\pi(s_{t+1}) | s = s_t, a = a_t].$$

Reinforcement learning looks for an optimal policy that maximizes Q^π, which can be denoted Q^*.

$$Q^*(s,a) = \mathbb{E}[r_{t+1} + \gamma \max_{a'} Q^*(s_{t+1}, a') | s = s_t, a = a_t]$$

Without an environmental model, temporal difference (TD) learning learns directly from experience and bootstraps to update value function estimates—it updates the estimates based on the previously learned estimates. The simplest TD updates the value function as follows:

$$V(s_t) \leftarrow V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)]).$$

Since $V(s_{t+1})$ is not known, the current estimate is used.

For control problems, on-policy TD, SARSA [31], estimates $Q^\pi(s, a)$ for the current behavior policy π. The Q estimate for next state and action s_{t+1} and a_{t+1} is fed in for bootstrap update as follows:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)].$$

Here, the action value function Q is for current behavior policy π. For simplicity, π superscript is omitted. Independently from the current behavior policy, off-policy TD, Q-learning [39], directly approximates Q^*. From $Q^*(s, a) = \max_{a'}(s, a')$, Q-learning updates is defined by

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)].$$

From the estimated Q, the current best policy can be chosen greedily:

$$\pi^*(s) \leftarrow \arg\max_a Q^\pi(s, a).$$

However, greedy action selection can result in not enough samples collected for correct estimates of value function. In this paper, we use ϵ-greedy that selects a random action with probability ϵ and chooses a greedy action with probability $1-\epsilon$. By decreasing ϵ as learning goes on, an agent exploits the learned best actions more.

III. Relevance Vector Machine (RVM)

The relevance vector machine [35] is a kind of kernel method that provides sparse solutions while maintaining Bayesian interpretability. The basic idea is that defining hyper-parameters to determine the prior distribution of weights that favors smooth, sparse hypotheses. RVMs eliminate redundant parameters and results in sparse solutions. For this, RVMs define a linear model with an independent Gaussian prior that is imposed on weights. This differs from Gaussian processes with unified priors.

RVMs can be viewed as a Bayesian extension of support vector machines that resembles the linear model. RVMs share many characteristics of SVMs while avoiding the limitations such as point-estimate output, necessity of parameter search, kernel requirement for Mercer's Theorem, and no guarantee of sparse solutions [6]. The probabilistic output of RVMs captures the uncertainty in their predictions. RVMs are less sensitive to hyper-parameters than SVR, and the kernel function does not need to be positive definite.

The learning algorithm obtains a posterior distribution over the weights from Bayes rule. Thereafter, it acquires a marginal likelihood (or evidence). Maximizing the likelihood iteratively creates an eventual smooth hypotheses with a small number of data points used. Thus, RVMs learn weight distributions and hyper-parameters alternatively.

From kernel k, we define the basis function $\phi_i(\mathbf{X}) = k(\mathbf{x}, \mathbf{x_i})$. We define $\mathbf{\Phi}$ as a matrix composed of training vectors transformed by the basis function; that is, $\mathbf{\Phi} = [\phi(\mathbf{x_1}), \phi(\mathbf{x_2}), \cdots, \phi(\mathbf{x_n})]$, where $\phi(\mathbf{x_n}) = [1, \phi_1(\mathbf{x_n}), \phi_n(\mathbf{x_n}), \cdots, \phi_n(\mathbf{x_n})]^\top$. $\boldsymbol{\alpha}$ and β represent the hyper-parameters for the prior distribution of weights $\mathbf{w}$ and target $\mathbf{t}$. γ_i is interpreted as a measure of how well-determined the weight w_i is by the data. First, it computes the mean and covariance of the weights:

$$\boldsymbol{\mu} = \beta \mathbf{\Sigma} \mathbf{\Phi}^\top \mathbf{t}, \tag{1}$$

$$\mathbf{\Sigma} = (\beta \mathbf{\Phi}^\top \mathbf{\Phi} + \boldsymbol{\alpha}\mathbf{I})^{-1}. \tag{2}$$

From the weight estimation, it computes the hyper-parameters:

$$\gamma_i = 1 - \alpha_i \Sigma_{ii}, \tag{3}$$

$$\alpha_i \leftarrow \frac{\gamma_i}{\mu_i^2}, \tag{4}$$

$$\beta \leftarrow \frac{N - \sum_i \gamma_i}{\|\mathbf{t} - \mathbf{\Phi}\boldsymbol{\mu}\|^2}. \tag{5}$$

These two update steps are repeated until convergence.

When the hyperparmeter $\alpha_i = \infty$, the corresponding weights w_i becomes zero because of the assumed zero-mean Gaussian prior distribution. Thus, the related samples can be deleted and remaining samples are retained to construct base features. Obtained sparse data are treated as a key experience to build RV bases in RVM-RL in the following section.

IV. RVM-RL

The reinforcement learning framework with RVM function approximator is depicted in Figure 1. The approach extends Fitted-Q minibatch learning [24] with sequential accumulation of key experiences. From the agent's interaction with an environment, the learning framework collects samples for training an RVM. The RVM regression model at each step estimates the Q values with Q-learning updates. As learning continues, it shapes the target function approximator for the next Q value estimation. For this, the evaluation and decision steps are necessary to establish a better target function approximator. We apply transfer learning of learned RVs for coherent learning. Our hypothesis in this paper is that RVM regression on the Bellman target can obtain significant RVs for good Q estimation. By transferring and shaping the knowledge from RVM training, we expect to achieve good Q function approximation.

From an interaction with a simulated or real world, an RL agent collects samples $(s_t, a_t, s_{t+1}, r_{t+1})$. For mini-batch training, the agent collects N samples and updates the RVM model via a Q-learning update (Step 2). The sparse solution of RVM training embedded the RVs that can refer the corresponding input samples. For RV transfer and feature computation,

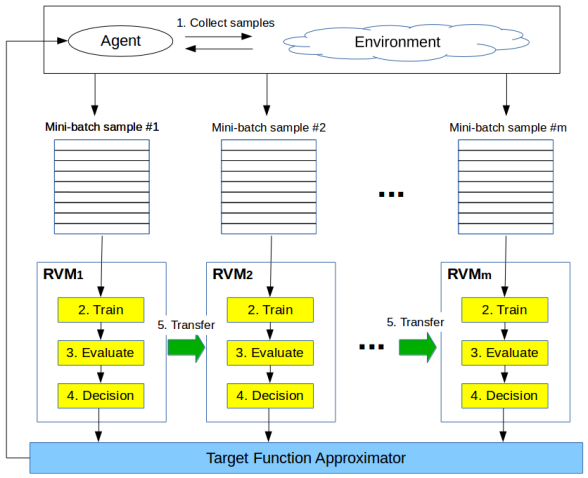


Fig. 1: RVM-RL learning framework

Algorithm 1 RVM-RL

Initialization: empty features for $\mathbf{RVM}_0$ and set target function approximation $\mathbf{FA} = \mathbf{RVM}_0$.
Choose discounting factor $\gamma \in (0, 1]$, learning rate c, and threshold τ.
for each episode **do**
  Select action $\mathbf{a_t}$ given state $\mathbf{s_t}$ by ϵ-greedy with **FA**. Apply $\mathbf{a_t}$ to arrive at $\mathbf{s_{t+1}}$.
  Observe N samples, $(\mathbf{s_t}, \mathbf{a_t}, r_t, \mathbf{s_{t+1}})$ at time step t.
  Initialize the base features of $\mathbf{RVM}_t$ with transferred RVs.
  Add the RVs to training data and target.
  Set target $y = r_t + \gamma max_a Q_{\mathbf{w},\boldsymbol{\alpha},\beta}(s_{t+1}, a)$
  Train $\mathbf{RVM}_t$ with alternate iteration of (1) to (5).
  Evaluate $\mathbf{RVM}_t$ with train RMSE
  if RMSE $< \tau$ **then**
    $\mathtt{RV}(\mathbf{FA}) = \mathtt{RV}(\mathbf{RVM}_t) \cup \mathtt{RV}(\mathbf{RVM}_{t+1})$
    $\mathtt{W}(\mathbf{FA}) = (1 - c)\mathtt{W}(\mathbf{RVM}_t) + c\mathtt{W}(\mathbf{RVM}_{t+1})$
    Store $\mathtt{RV}(\mathbf{RVM}_{t+1})$ for transfer.
  end if
  Decreases ϵ
end for

we store these input samples. From now on, we regard the relevance vectors to transfer or store as the related input samples.

When regression training is done, the framework checks the trained RVM to decide if the RVM is good enough to shape the target function approximation (Step 3). Many different heuristics possibly exist for evaluating RVM training. In this paper, we chose the regression root-mean-square error (RMSE) as a measure for this decision. When the training RMSE is greater than a preset threshold, we ignore the current RVM for the target function approximator.

The RVM that passes the evaluation test will be kept for updating the target function approximator while the RVM that

failed to pass the test will be ignored. The passed RVM is now ready to update the target function approximator (Step 4). There are possible heuristics for this step again. A new Q estimator can be constructed by averaging all successful RVMs. Another way, which is the method used here, is to shape the target function approximator with stochastic updates:

$$\mathtt{RV}(\mathbf{RVM}_{target}) = \mathtt{RV}(\mathbf{RVM}_{target}) \cup \mathtt{RV}(\mathbf{RVM}_{t+1}),$$
$$\mathtt{W}(\mathbf{RVM}_{target}) = c\mathtt{W}(\mathbf{RVM}_{target}) + (1-c)\mathtt{W}(\mathbf{RVM}_{t}),$$

where $\mathtt{RV}(\cdot)$ retrieves the relevance vectors of the RVM and $\mathtt{W}(\cdot)$ retrieves the weights (mean estimates) of the RVM. c is a stochastic update scalar to control the learning speed. When $c = 1$, it completely ignores previous RVMs and uses the new RVM. $c = 0$ means the target function approximator is not affected by the new RVM. When the target decision is made, the agent uses this function approximator to collect next samples. The weights for discarded RVs become zero. In this approach, the increment of the RVs can lead to an very large number of them. The set of important states, however, that are captured by the RVM converges to sparse solutions as we will see in Section V.

For the next sample training, instead of starting from scratch, we transfer the relevance vectors. The new, next RVM initializes the initial features with the transferred RVs. That is, the transferred RVs are added to the training samples, and the initial RVs are set to the indices of the transferred RVs.

$$\mathbf{X}_t = \mathbf{X}_{transfer} \cup \mathbf{X}_t$$
$$\mathtt{RV}(\mathbf{RVM}_t) = \mathbf{X}_{transfer}$$

where $\mathtt{RV}(\mathbf{RVM}_0) = \phi$. When the collected samples are biased in a certain space, learned RVs can be forgotten. By transferring RVs and using it as the initial base features of next RVM, it helps learning to be unbiased on each stage and alleviate forgetting.

Algorithm 1 summarizes the framework.

V. Experiments

Two reinforcement learning benchmark problems were used to investigate the efficiency of the RVM-RL framework. The first is the mountain-car problem, in which an under-powered car must be controlled to climb up a hill by using inertia. The second problem is a pole balancing problem.

We selected baseline algorithms based on function approximators. First, we compare RVM-RL with a value iteration approaches with neural networks. For this, we test neural fitted Q learning that is most similar structure to the suggested approach. Considering the two different viewpoints of RVM, an extension of SVM and GP, we test online SVR function approximation (oSVR-RL) [14] and GPTD [9].

In our experiments, oSVR-RL fails to learn in a given relatively short samples (200 episodes). Furthermore, it suffers from the huge number of support vectors that limits the choice of good kernel parameters because of the computation and memory limit. Thus, oSVR-RL results are not presented. To illustrate the SV explosion in RL problems, we examine the number of SVs with the actor-critic SVR-RL [15], which generates less SVs than oSVR-RL.

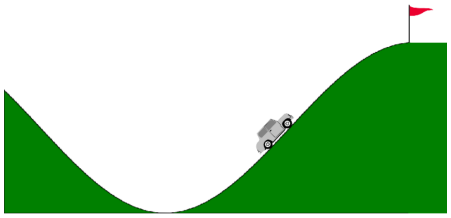


Fig. 2: Mountain car problem

A. Mountain Car

The mountain car (Figure 2) is a popular dynamics problem having an under-powered car that cannot climb a hill directly, but must be pushed back and forth to gain the momentum needed to reach the goal on top of the right hill. There are three actions: move forward $(+1)$, move backward (-1), and no acceleration (0). The optimal solution of pushing the car away from the goal makes the problem difficult. This continuous control problem is described in detail in [30].

The state is two dimensional, consisting of the car position x_t and its velocity $\dot{x}_t$. Following the classic mountain car problem, we assign the reward -1 on each time step. When it reaches the goal ($x_t = 0.5$) at the top of the right hill, the agent gets the reward 0 and is restarted from a random position. After each restart, a fixed number of samples are collected for training. The described reinforcement function is defined as follows:

$$r_t = \begin{cases} 0 & \text{if } x_t \geq 0.5 \\ -1 & \text{otherwise} \end{cases}$$

1000 samples are collected for each mini-batch, and 200 mini-batches are used to train the proposed RVM-RL framework. Parameter values for each function approximator were approximately optimized. For all experiments, the discount factor γ was set to 0.99 and ϵ decreased from 1 exponentially by the factor 0.9885 to a minimum of 0.1. For RVM-RL, the radial basis function (RBF) kernel with normalized input was used with the kernel parameter $\gamma_{\mathrm{k}}^{(\mathbf{RVM})} = 0.1$. The learning rate was $c = 1.0$. For each RVM training, the maximum number of iterations was set to 100 and tolerance threshold was set to 1×10^{-5}. For neural networks, two layers of 20 hidden units were used, which consist of one input layer (3 inputs), two hidden layers (20-20 units), and one output layer (one output). For the gradient update in backpropagation, we used Moller's scaled conjugate update [21] with a maximum number of iterations of 20 to avoid overfitting. GPTD parameters were chosen to be the accuracy threshold $v = 0.1$, the convergence threshold $\eta = 1 \times 10^{-4}$, and initial standard deviation $\sigma_0 = 0.1$. The RBF kernel was used for GPTD and the kernel parameter $\gamma_{\mathrm{k}}^{(\mathbf{GPTD})} = 0.01$.

To compare the performance of RVM-RL, GPTD, and neural networks, we repeated the experiment 10 times. Figure 3 shows the average number of steps to reach the goal with each

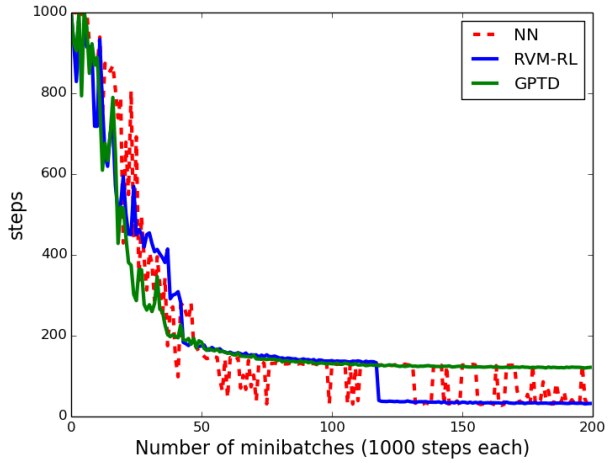


Fig. 3: Average of steps to reach the goal in mountain car problem.

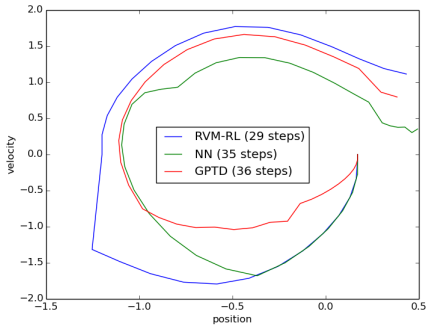


Fig. 4: Trajectory of successful control of mountain car by trained RVM from position -0.3.

function approximator. While the neural network's performance oscillates near the optimal point, both GPTD and RVM-RL show stable convergence. However, GPTD found a policy that reaches the goal with a greater number of steps with 200 mini-batches of training. Note that RVM-RL reaches the goal with the smallest number of steps. Example state trajectories followed by each of the three function approximation methods are shown in Figure 4. The RVM-RL agent consistently applies the -1 action to push the car farther up the left hill, then applies the +1 action to reach the largest velocity of the three methods, reaching the goal the fastest.

B. Pole Balancing

Adding a pole to a cart that swings in two dimensions, Barto et al. [4] first introduced the benchmark pole-balancing problem (Figure 5). The objective is to apply forces to the cart to keep the pole from falling over. Three actions to control the cart are defined: push left, push right, and apply zero force. When the cart reaches the end of track, it stops with zero velocity. In this instance, the pole is allowed to swing the full $360°$.

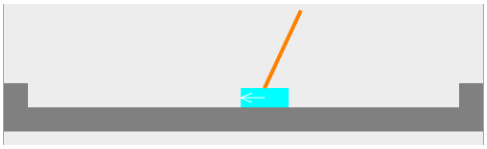


Fig. 5: Pole balancing task

The state of this system is four dimensional: the cart position x_t, its velocity $\dot{x}_t$, the pole angle θ_t, and the angular velocity $\dot{\theta}_t$. When the angle $\theta_t = \pi$, the pole is upright. The reward function is defined in the terms of the angle as follows:

$$r_t = \begin{cases} 1 & \text{if } |\theta_t - \pi| < \frac{\pi}{3} \\ 0 & \text{otherwise} \end{cases}$$

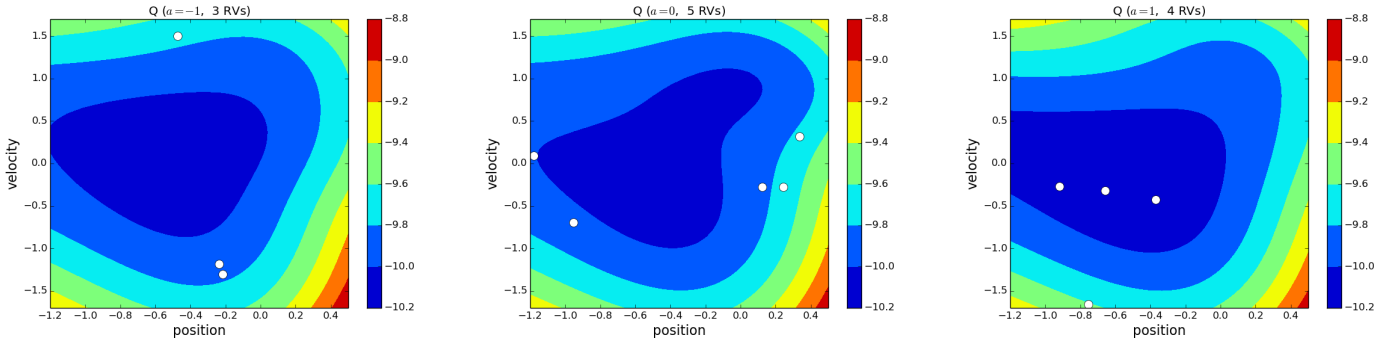
Thus, when it can balance the pole through the simulation time, the optimal policy will result in the average reward of 1.0.

With the ϵ-greedy policy with an ϵ that decreases exponentially with the factor of 0.9332, we use the discounting factor $\gamma = 0.99$. For training, we use 100 mini-batches, each with 1000 steps of samples. For RVM-RL, RBF kernel parameter $\gamma_{\mathrm{k}}^{(\mathbf{RVM})} = 1.0$ was the best from our pilot tests. The learning rate $c = 0.1$ was chosen, RVM max iteration was 100, and tolerance was 1×10^{-5}. From pilot tests, even with the best performing parameters, neural networks and GPTD were not able to find an optimal policy in 100 mini-batches. With 200 minibatches with more random explored samples, neural networks and GPTD converged to good policies. Neural networks with two hidden layers (10 hidden units per each) was the best structure. SCG max iteration was set to 80. GPTD required a finer accuracy threshold $\upsilon = 1 \times 10^{-5}$ and relaxed convergence threshold $\eta = 0.1$. Initial standard deviation $\sigma_0 = 10$ and the RBF kernel parameter $\gamma_{\mathrm{k}}^{(\mathbf{GPTD})}$ is set to 1×10^{-5}.

Figure 6 compares the average reward curves in pole-balancing task and shows the good performance of the proposed framework. RVM-RL shows fast learning to balance the pole most of the time. Neural networks and GPTD fail to learn with the given 100 mini-batches. They need twice as many samples to learn the policy, compared to RVM-RL. Figure 7 confirms the good performance of the RVM-RL when applying the learned policy. The upper plot shows the position changes over the 1000 steps and the bottom shows the angle trajectory. It moves slightly toward the right but keeps the pole balanced near π.

VI. Discussion

As Tipping, et al., [33] state, one of the issues in support vector regression is the growing number of support vectors as the number of samples increases. This impact gets severe when we apply SVR to reinforcement learning function approximator. Repeating 10 mountain car experiments, Table I compares the number of support vectors and relevance vectors. The SVM model that we use for comparison is the SVR-RL, actor-critic model by Lee, et al., [15]. The mean and median of the



(a) left (b) no action (c) right

Fig. 8: RVM-RL trained on mountain-car problem. Relevance vectors shown as white dots over the Q contour plot.

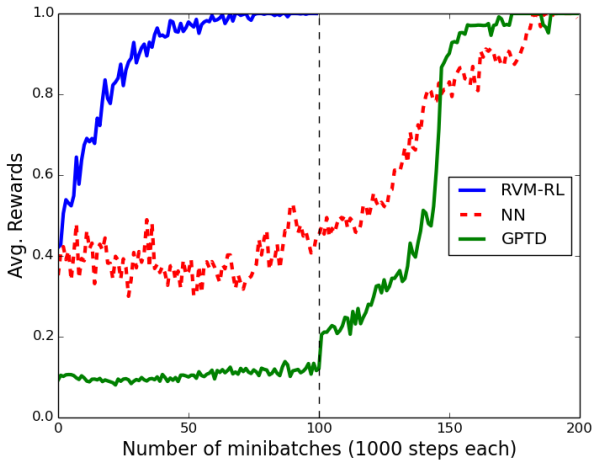


Fig. 6: Average of rewards of an episode in pole balancing.

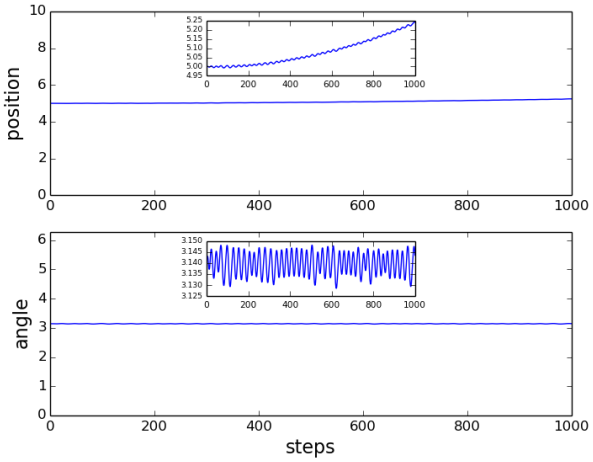


Fig. 7: Trajectory of successful control for pole balancing. With positive samples, RVM quickly adjusts learning. The cart and pole stay center and upright. Magnifying inner plots show slight wiggle and movement to right.

number of SVs are 286 and 128.5, and the mean and median of the number of RVs are 11.5 and 10.5. The table illustrates the sparseness of our RVM approach, even after the gradual augmentation of RVs. This suggests that RVM-RL may be the more practical approach for complex RL problems such as high-dimensional or continuous-state tasks. In the light of this, we have recently applied RVM-RL with continuous action control to high dimensional octopus arm control problem, and we intend to publish further results shortly.

	Mean	Median	Min	Max
SVM	286	128.5	19	852
RVM	11.5	10.5	8	45

TABLE I: The number of support vectors and relevance vectors in mountain car problem. The numbers are collected from 10 runs for each FA.

In Figure 8, we examine the selected RVs for the mountain car problem over the contour plot of Q values for each action. From the samples experienced during RL training, RVM-RL discovered the key samples that can be used as a basis and they are plotted as white dots. A total of 12 RVs were chosen as a basis; 3, 5, and 4 RVs for actions of -1, 0, and $+1$, respectively. It appears that for action -1 (Figure 8a), two RVs represent positions near the bottom of the valley and negative velocity. These probably contribute to a higher Q value for the -1 action for these states. Most RVs for action 0 (Figure 8b) are placed in low velocity areas and near the top of the both hills. RVs for action $+1$ (Figure 8c) allow the selection of action 1 when the car has moved far enough up the left hill. Figure 9 shows the policy that depicts the action that maximizes the RVM-RL Q estimations for each state. The figure illustrates the policy of pushing right when moving down the left hill with high positive velocity, in the upper portion of the displayed velocity-position state space. The low velocity range in the middle is governed by no action. When the velocity is negative, in the lower half of the velocity-position space, pushing left rules.

The learned RVs can be considered as high level knowledge about the dynamics and goals of the task. They are abstractions

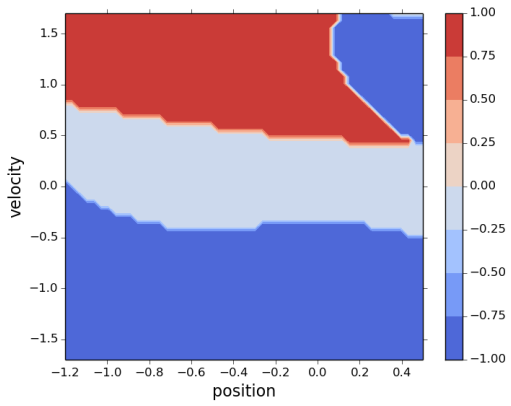


Fig. 9: Greedy action policy of RVM-RL trained on mountain-car problem.

of the experienced state-action pairs that are most relevant to the RL problem. The sharing of RVs, rather than the large number of state-action pairs, with other RL agents in the same or similar environment can be used to quickly initialize them with good base features. This can improve adaptability in multiagent systems or for environmental changes. This is similar to human learning that mimic others or following advise from a teacher or a coach. Similar approaches are studied in the transfer of learning context, such as imitation [18, 11] and advising [36, 37]. For instance, after RVM-RL training, the key experiences in RVs are transferred for a complex target task. By repeating the key actions, an agent imitates the learned behaviors in an easy task to quickly discover an optimal policy for the hard task.

VII. Conclusion

We have described a novel reinforcement learning framework that trains relevance vector machines (RVMs) as function approximators with mini-batch samples. By transferring RVs acquired during mini-batch training, RVM-RL maintains the learned knowledge, which are considered as important experiences. By first evaluating the new RVs and not transferring them if they are judged to be detrimental, we filter negative knowledge transfer that can deter learning.

The major contribution of our RVM-RL approach is a unique extension of the relevance vector machine for sparse Bayesian reinforcement learning. Policies learned by RVM-RL can lead to a useful analysis of the state-action space structure by examining the RVs after training. This analysis can also be utilized for a transfer of knowledge for imitation or advising. Rasmussen et al. [22] discussed the problem of RVM–as new samples are away from training samples, most bases do not correspond to new inputs and the predictive variance gets small. The authors suggested augmentation as a solution, but they pointed out that the solution resulted in losing sparsity of RVMs. Our approach is similar to theirs by augmenting RVs, but we have shown that the sparsity is maintained with additional heuristics in the framework.

Our future research will focus on using the Bayesian traits to improve the learning performance such as investigating alternative exploration-exploitation strategies [7, 13]. Another avenue we will investigate is human contributions in the form of prior distributions can affect the learning performance. These additions will improve the quality of the RVs learned from the directed exploration. Further study of kernel tricks can extend the application of RVM-RL to many real-world RL problems that have not been attempted previously. Other interesting questions remain regarding ways to efficiently utilize the transferred RVs to improve reinforcement learning.

References

[1] James Sacra Albus. *Brains, behavior, and robotics*. 1981.

[2] Charles W Anderson, Minwoo Lee, and Daniel L Elliott. Faster reinforcement learning after pretraining deep networks to predict state dynamics. In *International Joint Conference on Neural Networks (IJCNN)*, 2015.

[3] André Barreto and Charles W. Anderson. Restricted gradient-descent algorithm for value-function approximation in reinforcement learning. *Artificial Intelligence*, 172 (4-5):454–482, 2008.

[4] Andrew G Barto, Richard S Sutton, and Charles W Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man and Cybernetics*, (5):834–846, 1983.

[5] Hamid Benbrahim and Judy A Franklin. Biped dynamic walking using reinforcement learning. *Robotics and Autonomous Systems*, 22(3):283–302, 1997.

[6] Christopher M. Bishop et al. *Pattern recognition and machine learning*, volume 1. 2006.

[7] Richard Dearden, Nir Friedman, and Stuart Russell. Bayesian q-learning. In *Proceedings of the National Conference on Artificial Intelligence*, pages 761–768, 1998.

[8] Marco Dorigo and Marco Colombetti. Robot shaping: Developing autonomous agents through learning. *Artificial Intelligence*, 71(2):321–370, 1994.

[9] Yaakov Engel, Shie Mannor, and Ron Meir. Bayes meets bellman: The gaussian process approach to temporal difference learning. In *Proceedings of International Conference on Machine Learning (ICML)*, volume 20, page 154, 2003.

[10] Yaakov Engel, Peter Szabo, and Dmitry Volkinshtein. Learning to control an octopus arm with gaussian process temporal difference methods. In *Advances in neural information processing systems*, pages 347–354, 2005.

[11] Fernando. Fernández and Manuela Veloso. Probabilistic policy reuse in a reinforcement learning agent. In *Proceedings of the fifth International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 720–727, 2006.

[12] Arthur Guez, David Silver, and Peter Dayan. Efficient bayes-adaptive reinforcement learning using sample-based search. In *Advances in Neural Information Processing Systems*, pages 1025–1033, 2012.

[13] J Zico Kolter and Andrew Y Ng. Near-bayesian exploration in polynomial time. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 513–520, 2009.

[14] Dong-Hyun Lee, Vo Van Quang, Sungho Jo, and Ju-Jang Lee. Online support vector regression based value function approximation for reinforcement learning. In *IEEE International Symposium on Industrial Electronics (ISIE)*, pages 449–454, 2009.

[15] Dong-Hyun Lee, Jeong-Jung Kim, and Ju-Jang Lee. Online support vector regression based actor-critic method. In *IECON 2010-36th Annual Conference on IEEE Industrial Electronics Society*, pages 193–198, 2010.

[16] Minwoo Lee and Charles W. Anderson. Convergent reinforcement learning control with neural networks and continuous action search. In *Proceedings of IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*, 2014.

[17] Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3-4):293–321, 1992.

[18] Michael G. Madden and Tom Howley. Transfer of experience between reinforcement learning environments with progressive difficulty. *Artificial Intelligence Review*, 21(3):375–398, 2004.

[19] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. In *NIPS Deep Learning Workshop*. 2013.

[20] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 02 2015. URL http://dx.doi.org/10.1038/nature14236.

[21] Martin F. Møller. A scaled conjugate gradient algorithm for fast supervised learning. *Neural networks*, 6(4):525–533, 1993.

[22] Carl Edward Rasmussen and Joaquin Quinonero-Candela. Healing the relevance vector machine through augmentation. In *Proceedings of the 22nd international conference on Machine learning*, pages 689–696, 2005.

[23] Ioannis Rexakis and Michail G Lagoudakis. Directed policy search using relevance vector machines. In *2012 IEEE 24th International Conference on Tools with Artificial Intelligence (ICTAI)*, volume 1, pages 25–32, 2012.

[24] Martin Riedmiller. Neural fitted q iteration–first experiences with a data efficient neural reinforcement learning method. In *Machine Learning: ECML 2005*, pages 317–328. 2005.

[25] Stéphane Ross and Joelle Pineau. Model-based bayesian reinforcement learning in large structured domains. *arXiv preprint arXiv:1206.3281*, 2012.

[26] Axel Rottmann, Christian Plagemann, Peter Hilgers, and Wolfram Burgard. Autonomous blimp control using model-free reinforcement learning in a continuous state and action space. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, 2007.*, pages 1895–1900, 2007.

[27] Juan Carlos. Santamaria, Richard S.. Sutton, and Ashwin Ram. Experiments with reinforcement learning in problems with continuous state and action spaces. *Adaptive behavior*, 6(2):163, 1997.

[28] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529 (7587):484–489, 2016.

[29] Peter Stone, Richard S. Sutton, and Gregory Kuhlmann. Reinforcement learning for robocup soccer keepaway. *Adaptive Behavior*, 13(3):165–188, 2005.

[30] Richard S. Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44, 1988.

[31] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. 1998.

[32] Gerald Tesauro. Td-gammon, a self-teaching backgammon program, achieves master-level play. *Neural computation*, 6(2):215–219, 1994.

[33] Michael E. Tipping. Sparse bayesian learning and the relevance vector machine. *The Journal of Machine Learning Research (JMLR)*, 1:211–244, 2001.

[34] Michael E. Tipping and Anita C. Faul. Analysis of sparse bayesian learning. In *Proceedings of the Conference on Neural Information Processing Systems (NIPS)*, pages 383–390, 2002.

[35] Michael E. Tipping, Anita C. Faul, et al. Fast marginal likelihood maximisation for sparse bayesian models. In *Proceedings of the Ninth International Workshop on Artificial Intelligence and Statistics*, volume 1, 2003.

[36] Lisa Torrey, Trevor Walker, Jude Shavlik, and Richard Maclin. Using advice to transfer knowledge acquired in one reinforcement learning task to another. In *Machine Learning: ECML 2005*, pages 412–424. 2005.

[37] Lisa Torrey, Jude Shavlik, Trevor Walker, and Richard Maclin. Relational skill transfer via advice taking. In *ICML Workshop on Structural Knowledge Transfer for Machine Learning*, 2006.

[38] Nikolaos Tziortziotis, Christos Dimitrakakis, and Konstantinos Blekas. Linear bayesian reinforcement learning. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence*, pages 1721–1728, 2013.

[39] Christopher John Cornish Hellaby Watkins. *Learning from delayed rewards.* PhD thesis, University of Cambridge, 1989.

[40] David Wipf, Jason Palmer, and Bhaskar Rao. Perspectives on sparse bayesian learning. In *Advances in Neural Information Processing Systems 16*, 2003.