DISSERTATION

SPARSE BAYESIAN REINFORCEMENT LEARNING

Submitted by

Minwoo Lee

Department of Computer Science

In partial fulfillment of the requirements

For the Degree of Doctor of Philosophy

Colorado State University

Fort Collins, Colorado

Summer 2017

Doctoral Committee:

Advisor: Charles W. Anderson

Asa Ben-Hur
Michael Kirby
Peter Young

ABSTRACT

SPARSE BAYESIAN REINFORCEMENT LEARNING

This dissertation presents knowledge acquisition and retention methods for efficient and robust learning. We propose a framework for learning and memorizing, and we examine how we can use the memory for efficient machine learning. Temporal difference (TD) learning is a core part of reinforcement learning, and it requires function approximation. However, with function approximation, the most popular TD methods such as TD(λ), SARSA, and Q-learning lose stability and diverge especially when the complexity of the problem grows and the sampling distribution is biased. The biased samples cause function approximators such as neural networks to respond quickly to the new data by losing what was previously learned. Systematically selecting a most significant experience, our proposed approach gradually stores the *snapshot memory*. The memorized snapshots prevent forgetting important samples and increase learning stability. Our sparse Bayesian learning model maintains the sparse snapshot memory for efficiency in computation and memory. The Bayesian model extends and improves TD learning by utilizing the state information in hyperparameters for smart decision of action selection and filtering insignificant experience to maintain sparsity of snapshots for efficiency.

The obtained memory can be used to further improve learning. First, the placement of the snapshot memories with a radial basis function kernel located at peaks of the value function approximation surface leads to an efficient way to search a continuous action space for practical application with fine motor control. Second, the memory is a knowledge representation for transfer learning. Transfer learning is a paradigm for knowledge generalization of machine learning and reinforcement learning. Transfer learning shortens the time for machine learning training by using the knowledge gained from similar tasks. The dissertation examines a *practice* approach that transfers the snapshots from non-goal-directive random movements to goal-driven reinforcement

learning tasks. Experiments are described that demonstrate the stability and efficiency of learning in 1) traditional benchmark problems and 2) the octopus arm control problem without limiting or discretizing the action space.

ACKNOWLEDGEMENTS

This dissertation is the result of many invaluable discussions with several great scientists. First, I would like to thank Charles W. Anderson, the best advisor ever! He has rendered me his valuable time, deep insight, and kindness. He has given me the freedom to find my way in research, encouraged me to keep focused, and helped me to proceed in a right direction. Under his guidance, I have explored many interesting machine learning topics, which helped to develop this topic and many others. Our weekly meetings and email exchange discussed a broad range of topics including non-research related ones and helped me a lot in every aspect that I can think of. I feel lucky to have him as my advisor and research collaborator, and I wish to continue to have him as a mentor for my life in the future.

I also wish to express my gratitude to my graduate committee: Asa Ben-Hur, Michael Kirby, and Peter Young. I thank my examining committee for reading the manuscript and their comments. Asa always asks valuable questions to understand or formulate problems better and to express my ideas clearly. Discussion with Dr. Kirby always broaden my view about the problem by suggesting a different viewpoint. Dr. Young is so flexible to help and support me as well and gives me insightful advise.

I owe thanks to the inspiring and supportive professors and researchers at CSU like Dr. Bruce Draper, Dr. A.P. Willem Bohm, Dr. Darrel Whitley, Dr. Sanjay Rajopadhye, Dr. Ross Beveridge, Dr. Ross M. McConnell, Dr. Hamidreza Chitsaz, Dr. Sangmi Lee Pallickara, Dr. Shrideep Pallickara. Dr. Adele Howe, and Dr. Robert France. My condolences on the passing of Adele and Robert who will be in my memory forever as nice persons and professors. I would like to thank the graduate students at CSU: Elliott Forney, Fereydoon Vafaei, Awad Younis, Manf Gharaibeh, Tomojit Ghosh, Phillipa Bennett, Alex Fout, Kathleen Ericson, Matthew Malensek, and Fayyas Minhas. A special thank goes also to my world-traveller friend Reinaldo and Edwin, for interesting discussion about reinforcement learning.

Great thanks are also due to Elliott Forney for building this LaTeX document class.

Finally and most importantly, I would like to thank my family. I thank my wife, Heehwa for love and support, and faith in me. She was the one who encouraged me to choose to join PhD program at Colorado State University, which has been a fruitful experience. Without her kind support and caring, none of this would ever have come to pass. I thank my lovely kids, Annabel and Liam. You are the people who make me happy always.

DEDICATION

To Heehwa, Annabel, and Liam ...

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

1.1 Overview of the Problem

Memory is an essential part of the learning process. This poses questions of how to represent knowledge and how to store and retrieve it efficiently. This idea is closely related to cognitive information processing in psychology and cognitive science. Information processing goes through three stages: acquisition of knowledge, retention, and transfer (Figure 1.1) [3–6]. From learning, humans achieve new knowledge and store it for future usage. Based on the stored information, we generalize it to solve other problems. For instance, when we learn how to play soccer, we first learn how to kick a ball. From the learned kicking skill, we remember the location of the non-kicking foot, leg swing and contact positions between the ball and the kicking foot. From the memorized information, we generalize or transfer this kicking skill to shooting a goal or passing to another player. Similarly, a number of *transfer learning* approaches have been attempted [7–28] to improve the performance of machine learning algorithms by establishing the “good" knowledge basis from the other solutions of the related tasks. The memory plays a key role for the retention. There are a few questions that arise such as how to store and retrieve the data and what to store for future usage. In this dissertation, we examine the latter one, what is most valuable to remember for generalization of knowledge in the future. Also, we examine how to maintain a sparse memory for efficient and robust learning.

In machine learning and reinforcement learning research, one of the challenges is stability of learning. When the sample distribution or explored samples are biased, the algorithms are prone

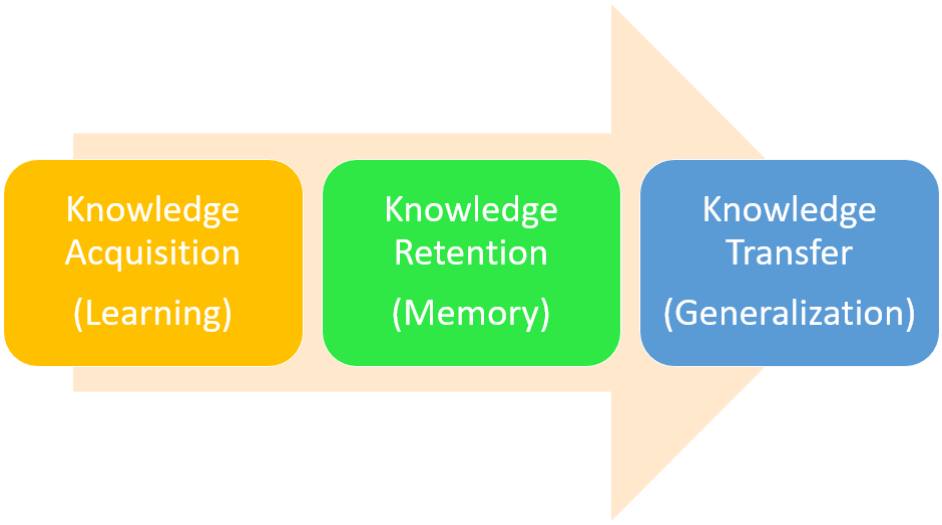


Figure 1.1: Cognitive information processing steps

to fit on the biased data and to lose generalization ability. One of these examples is the neural networks' catastrophic forgetting [29–32]. Neural networks quickly respond to new data and make a good approximation while losing what they learned previously. This learning behavior makes its learning process unstable. To prevent forgetting, we focus on data retention process. If we do not forget its experience, we will be able to make learning stable. However, what to remember and how much memory we need to keep are significant questions. This dissertation examines these memory related issues to improve learning stability and sparse memory management.

1.2 Motivation

The motivation of knowledge retention in reinforcement learning is to reduce the complexity of problems as in knowledge representation and reasoning [33, 34]. Knowledge representation is one area of machine learning and artificial intelligence that captures information about the environment. Generally, it looks for concise and sparse representations that help complex problem solving become easier. For instance, dimensionality reduction can minimize the number of variables to

consider and disentangle the complex problems. Combining with the traditional knowledge representation, if we can remember the significant experiences or samples from data, we can better represent the world. From stored knowledge, we can transfer the knowledge to other tasks (transfer learning). Comparing the snapshot memories from different tasks, we can compare similarity between problems. By maintaining the sparsity of retained memory, we can provide analysis tools to understand the problems or solutions (by reviewing the snapshots, we can analyze the critical causes).

1.3 Significance of Sparse Bayesian Reinforcement Learning

This dissertation first proposes a general reinforcement learning framework that can systematically memorize the significant experiences. The significance of the research in this dissertation is summarized by the following points.

- The sparse Bayesian reinforcement learning proposes the general learning architecture for machine intelligence. Following the human information processing chain, it suggests an efficient and stable learning algorithms. By augmenting knowledge, it prevents forgetting previously learned information and eventually achieve robustness of learning.

- The sparse Bayesian approach systematically remembers the most significant data samples. It is first proposed to memorize the important samples based on the selection of kernels, functions that measure the similarity between two inputs. Various heuristic approaches and the choice of kernels can improve the flexibility of learning.

- Stored snapshot memory can improve learning. It can be used to generalize the knowledge. It first suggests a practice approach that transfers knowledge from a non-reinforcement learn-

ing task to a reinforcement learning task. We examines how practice can be generalized to unknown tasks and how it can improve the performance of target task solving.

- By using the snapshot memory, we suggest the first solution of the simulated octopus arm control problem without limiting the number of actions. Allowing continuous action control in the octopus arm problem makes the problem difficult to solve with the high dimensional, infinite search space. Efficient ways to search a continuous action are examined.

1.4 Research Objectives

The objective of this dissertation is the design and study of a general purpose intelligent system that remembers the most significant samples for robust learning. Knowledge augmentation is a key for stable learning, but it is important to maintain the sparsity of knowledge. The goals of this research are summarized as follows.

- The sparse Bayesian learning design will be able to store the most important experience to improve learning performance.

- The stored snapshot memory can be reused for generalization of knowledge. It can be transferred to other tasks, or it can used to improve a learning algorithms.

- The proposed approach mimic the cognitive information processing for knowledge acquisition (learning), knowledge retention (snapshot memory), and transfer (generalization to other tasks).

- Bayesian learning could provide an extra information to make a smart decision during exploration and knowledge retention. By using the confidence information, sparse Bayesian learning explore the world efficiently and maintain the sparsity of snapshot memories.

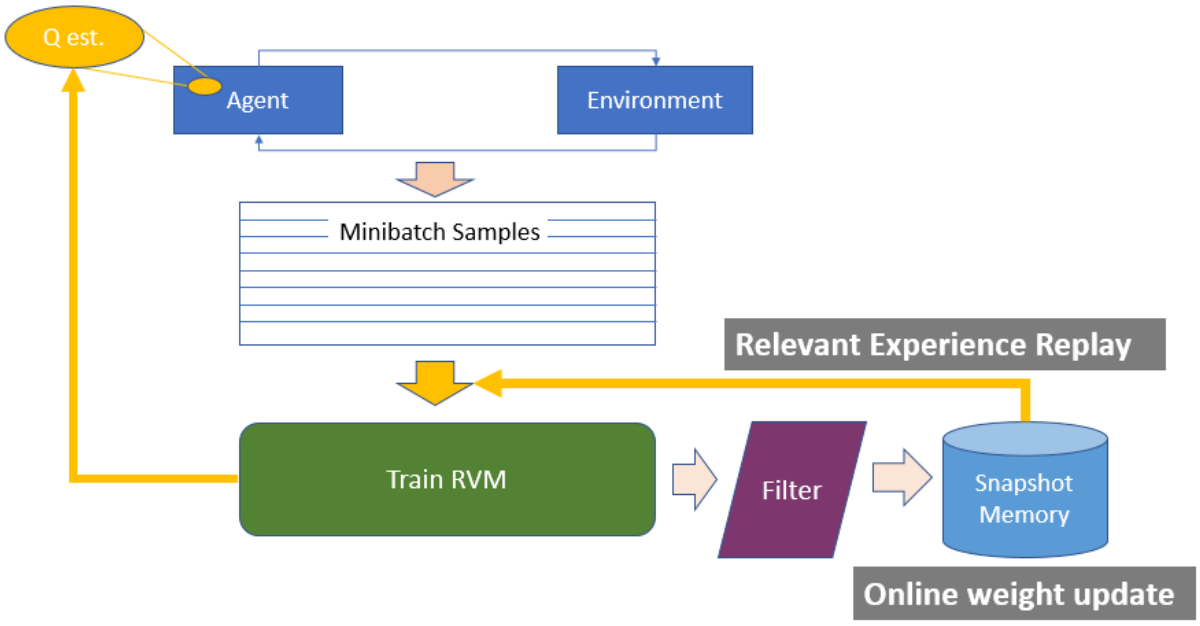


Figure 1.2: A Learning Framework for Knowledge Acquisition and Retention

1.5 Approaches for Knowledge Acquisition and Retention

The framework of the approach for knowledge acquisition and retention proposed here is depicted as in Figure 1.2. Based on the current value estimation of a state-action pair (Q value estimation in Section 2.1), an agent takes actions in an environment and collects minibatch samples. When we define the minibatch size to be one, it learns with online update. Based on the minibatch samples, the core function approximator, a relevance vector machine (see Section 2.5), estimates the Q values (see Section 4.1 and Section 5.2). By repeating sample collection and training, the agent gradually learns the solution for the reinforcement learning problems. For knowledge retention, the following are added.

Filtering: Not all experience is valuable. Some minibatch samples can be irrelevant to the solution that an agent want to learn. Also, in the middle of successive minibatch training, we can have

large training error that does not give any useful learning result. The estimated values with a large variance represent less confident approximation that are not helpful to learn. In these cases, we can simply ignore or *filter out* the experience to maintain sparsity of knowledge.

Storing: Whenever the experience is not filtered out, the raw samples (relevance vectors in Section 4.1 and Section 5.2) are stored. As learning proceeds, the are gradually augmented. We call this augmented knowledge as *snapshot memory*.

Online Weight Update: The snapshot memories are used as bases to estimate Q values along with the corresponding weights. Since the same snapshots can be reintroduced or unseen samples can be added to the snapshots, a proper weight management is necessary. Online weight update controls how quickly it will reflect new knowledge for the value estimation.

Relevant Experience Replay: Experience replay is one of the ways that we can mix input data with previous samples to avoid sampling bias [35, 36]. Without additional memory requirement, relevant experience relay avoids sampling bias problem with snapshot memory efficiently. Also, instead of simply reintroducing previous samples to training function approximator, it learns by building upon the snapshot bases and making relation between existing knowledge and new experience. This improves learning efficiency, shown in Chapter 5.

1.6 Overview of Dissertation

The rest of this dissertation is organized as follows.

Chapter 2 briefly introduces reinforcement learning. First, we review the definition of value functions and Q functions for control problems. Comparing to dynamic programming and Monte

Carlo approach, we review the temporal difference learning (TD). After that, we introduce function approximations to overcome the memory limit by approximating the value functions. To help understanding of the development of sparse Bayesian reinforcement learning, we review support vector machines, Bayesian learning, and relevance vector machines.

In Chapter 3, we introduce the online learning approach for sparse Bayesian learning. Applying the online learning approach to various problems, we examine the possibilities and limitations.

Based on Fitted-Q batch learning, Chapter 4 introduces the sparse Bayesian reinforcement learning framework to memorize the most important snapshots. From the suggested framework, we examine the sparsity of proposed solution and robustness of learning. Examining the snapshots, we analyze the learned solutions.

Chapter 5 examines how the snapshot memory can be used to improve learning. Although the efficiency is well known [37], problems with continuous action control are difficult because of the infinite search space. By using the snapshot memory, we suggest an efficient real-valued action sampling.

In Chapter 6, we examine the generalization of the achieved knowledge. Unlike traditional transfer learning, we consider harder problems of transferring the experience from supervised learning to reinforcement learning task. We call this approach *practice* to improve learning speed and performance.

Chapter 7 concludes the dissertation, summarizes the main contributions and discusses the future directions that can use the snapshot memory efficiently.

Chapter 2

Background

In this chapter, we introduce the formulation of reinforcement learning and review function approximations for large state space problems. For the choice of function approximations, we review support vector machines, Bayesian learning and relevance vector machines. This chapter provides the introduction to the core elements of sparse Bayesian reinforcement learning. The complete overview of reinforcement learning, support vector machines and relevance vector machines are available in [1, 35, 38–41].

2.1 Reinforcement Learning

By interacting with an environment, a reinforcement learning agent collects training samples. From the collected samples, it evaluates its value on each move to achieve a policy to reach a goal. The interaction with an environment contains the agent's selection of actions and the environment's responses to them. Series of interactions present new situations to the agent along with rewards, numerical values that the agent tries to maximize.

This sequence of interactions is formulated as a Markov decision process (MDP). An MDP is defined as a tuple $(S, A, P^a_{ss'}, R, \gamma)$, where for each time step $t = 0, 1, 2, \cdots$, with probability $P^a_{ss'}$ action $a = a_t \in A$ in state $s = s_t \in S$ transitions to state $s' = s_{t+1} \in S$, 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. In general, the long term reward is represented by the expected sum of discounted rewards as follows:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1},$$

where $\gamma \in (0, 1]$ is a discounting factor.

To achieve a goal, reinforcement learning algorithms estimate the value of being in a state or the value of performing a certain action in a given state. Here, the value can be defined as the expected return, R_t. For a policy $\pi : S \rightarrow A$, the value function V^π that estimates the value of a state s is defined as:

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

The value function satisfies the following Bellman equation:

$$V^\pi(s) = T^\pi V^\pi(s) = R^\pi(s) + \gamma P^\pi V^\pi(s) = \mathbb{E}[r_{t+1} + \gamma V^\pi(s_{t+1}) | s = s_t],$$

where $R^\pi(s_t)$ is a reward function for the state s_t, P^π is the state transition probability under the policy π, and T^π is known as a Bellman operator. This Bellman equation shows a relationship between the value of current state and its successor.

For control problems, to estimate how good an action is in a given state, we can define the action value method for policy π, $Q^\pi(s, a)$:

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

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 either V^{π} or Q^{π}, which can be denoted by V^{*} and Q^{*} respectively.

$$
\begin{aligned}
V^{*}(s) &= \max_{\pi} V^{\pi}(s) \\
V^{*}(s) &= \max_{a} Q^{*}(s,a) = \max_{a} \max_{\pi} Q^{\pi}(s,a) \\
Q^{*}(s,a) &= \max_{\pi} Q^{\pi}(s,a) \\
&= \mathbb{E}[r_{t+1} + \gamma V^{*}(s_{t+1}) | s = s_t, a = a_t] \\
&= \mathbb{E}[r_{t+1} + \gamma \max_{a'} Q^{*}(s_{t+1}, a') | s = s_t, a = a_t]
\end{aligned}
\tag{2.1}
$$

Dynamic programming (DP) [42, 43] can be used to solve an MDP for optimal control. By using value functions, DP algorithms learn good policies from additional information to the policy—they require complete knowledge of the transition and reward models as input. DP can compute optimal policies with a given perfect model of the environment, but there exist a major difficulty, as Bellman stated, the curse of dimensionality. As the number of discrete states or actions grows, or as the cost of discretization of the continuous state or action space grows exponentially, it is impossible to compute complex high dimensional problems.

Without input of the environment model, Monte Carlo (MC) methods [44] learn from online experience. We call the subsequences in the interaction between an agent and an environment episodes and call the tasks with the episodes that end in terminal states episodic tasks. An MC collects a complete trajectory per episode from an episodic task. Based on the average of sample returns, MC algorithms solve an MDP by analyzing the full history of state-action steps from complete episodes.

Temporal difference (TD) learning algorithms combine DP and MC ideas. Without an environmental model, TD learns directly from experience and bootstraps to update value function estimates—it updates the estimates based on the previously learned estimates. The simplest TD(0) 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.

The iterative value update, TD(0) provides an estimation of the V^π, but it cannot be used to find π for control tasks. For control problems, we compute the optimal state-action value function Q^* in Equation (2.1). On-policy TD, SARSA [35], 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, the π superscript is omitted. Independently from the current behavior policy, off-policy TD, Q-learning [45], directly approximates Q^*. From $Q^*(s, a) = \max_{a'} Q(s, a')$, Q-learning updates are 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 by

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

However, greedy action selection results in not enough samples for correct estimates of the value function. The trade-off between exploration and exploitation occurs here. Several strategies such as greedy, ϵ-greedy, softmax, and Gaussian noise balance the trade-off between them. ϵ-greedy selects a random action with probability ϵ and chooses a greedy action with probability $1 - \epsilon$. Softmax uses Boltzmann distribution to have different selection probabilities for actions as the temperature τ decreases—as $\tau \to 0$, softmax action selection is equivalent to greedy selection. Gaussian noise method adds noise with some variance parameter as input to a greedy action.

2.2 Function Approximations

Finding an optimal solution is challenging because of computational cost or memory limit. Without a model or complete episodes, TD learning requires us to approximate the optimal state or action values [35]. This can lead us to approximate unseen state or action values with frequently encountered observations, but since more learning effort is concentrated on the observed samples, infrequent observations can result in misleading approximation. Thus, we need to *generalize* the approximation to estimate the values of unseen states and actions accurately. Extrapolation for the unobserved states and actions makes this problem difficult.

Fortunately, generalization from samples is well-studied in general machine learning research. Reinforcement learning adopts those generalization methods, which is called *function approxima-*

tion in reinforcement learning context. In this section, we review the existing function approximation methods.

First of all, when a parametric value function approximation Q_θ is linear, it is defined as

$$Q_\theta(s, a) = \phi(s, a)^\top \theta,$$

where ϕ is a feature vector for state s and action a, and $\theta \in \mathbb{R}^d$ is a d dimensional weight vector. From samples, linear regression trains and updates the weights θ. In general, gradient-descent methods search for the only one optimum θ^*. Thus, it is *almost surely* guaranteed to converge at the global optimum— Baird [46] and Tsitsiklis and Van Roy [47] presented counterexamples that lead to divergence and infinite mean squared error in off-policy learning.

Linear tile-coding function approximation, cerebellar model arithmetic computers (CMAC) [48], is a form of discretization over continuous state space into tiles. Allowing overlapping tiles, CMAC computes its output through the weighted sum of active tiles, where the contribution of i-th tile is controlled by the weight w_i. Thus, $f(x) = \sum_i w_i f_i(x)$ where $f_i(x) = 1$ when i-th tile is activated and $f_i(x) = 0$ otherwise. Increasing tile width can improve generalization and raising the number of tiles can allow more accurate representations.

Radial basis functions (RBFs) [35] is a continuous variant of CMAC. Rather than being discrete either 0 or 1, feature values can be any value in the interval between 0 and 1. Replacing a discrete tile, each feature ϕ_i has Gaussian response that is dependent on the center and its variance. RBFs produces smooth function approximation that is differentiable, and this nonlinear approach fits precisely on the target. For more precise approximation, restricted gradient descent [49] changes its centers and variances.

One of the most popular function approximation methods, neural networks (NNs), estimate the action value function for continuous real-valued states. Although it has difficulty in convergence with SARSA [46], it has been applied to many RL problems such as backgammon, robot shaping, agent survival game, robot walking, robot soccer, and octopus arm control [50–55] successfully. For off-policy learning, Maei, et al., [56] and Lee, et al., [57] has shown convergence of learning with nonlinear function approximation. The weights in all layers are updated in gradient descent manner through back-propagation. Recently, Mnih, et al., [58, 59] 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. Silver, et al., [36] 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.

As Martin [60] suggested, online support vector regression (SVR) [61, 62] can be applied to value function approximation. As online SVR removes useless support vectors, it provides good generalization and performance on solving RL problems.

Bayesian models such as Bayesian Q-learning [63], Gaussian process temporal difference learning (GPTD) [64–66], and linear Bayesian reinforcement learning [67] have been applied to estimate value functions. Learning the distributions over Q functions, they showed good generalization to solve various reinforcement learning problems.

Value function approximations have shown success in estimating value functions and action-value functions. Linear models have solved the discrete state problems such as tetris [68], dynamic channel allocation [69], and robot obstacle negotiation [70]. Tile coding also showed success-

ful performance on acrobot control [71] and robot soccer [55]. As previously mentioned, neural network function approximation successfully solved reinforcement learning problems such as backgammon [50], navigation control [51, 52], robot walking [53], and robot soccer [54, 55]. Support vector regression solved cart port balancing problem [61, 62], and Bayesian model solved simple octopus arm control [65] and blimp control [66] successfully.

Even with the successful application of the estimated value function approaches, there is no free lunch that can be applicable for all the reinforcement learning problems [72]. Octopus arm control as an example was successfully solved with GPTD, but it only showed the successful learning with six predefined discrete actions. Also, there are well-known problems of each function approximations. We will investigate them in next chapter in detail. Tackling these problems, we propose a new function approximation method to be applicable on some practical reinforcement learning problems that cannot be covered by other function approximations.

2.3 Support Vector Machines

Support vector machines (SVM) [73] [41] are a popular tool in solving classification, regression, and novelty detection. An important property of support vector machines is that the determination of the model parameters corresponds to a convex optimization problem, which enables a local solution to be a global optimum [41]. Figure 2.1 illustrates the geometric interpretation of SVM objective formulation. Separating the hyperspace with large margin allows us more room for possible future classification errors. Therefore, SVMs are characterized by their margin: it looks for the hyperplane that separates data into two classes with the maximum margin. Let the training data be $(\mathbf{x}_i, y_i)$ where m input vectors $\mathbf{x}_i$ and target values $y_i \in \{-1, 1\}$. The hyperplane can be defined as

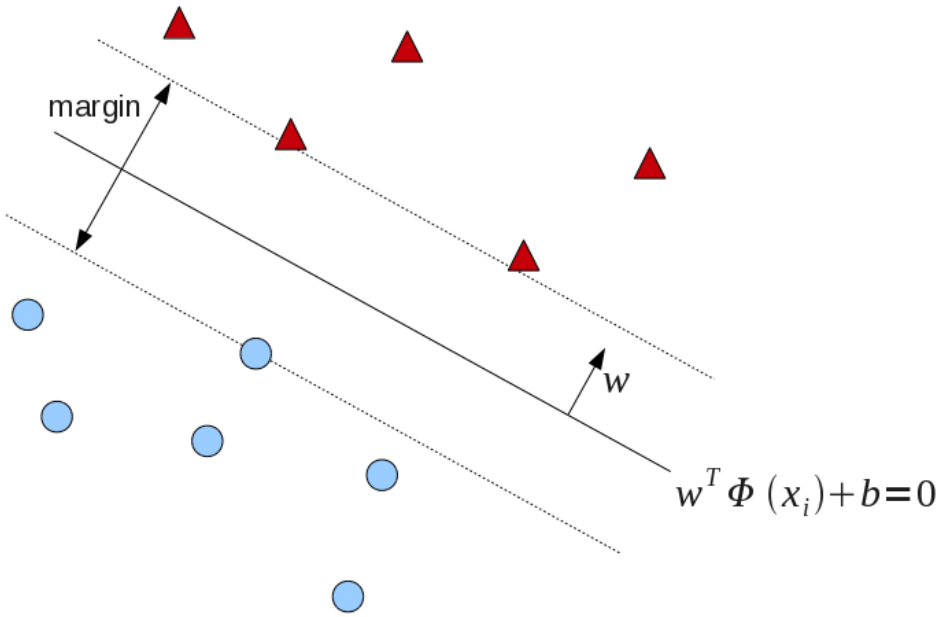


Figure 2.1: Support vector machine, the margin maximizer

$$f(\mathbf{x}) = \mathbf{w}^\top \Phi(\mathbf{x}) + b, \tag{2.2}$$

where $\Phi(\mathbf{x})$ denotes a basis function that maps the input data to feature space. The $\mathbf{w}$ is the weight vector, and scalar b is the bias. This linear combination in feature space makes the problem simpler with linearity in feature space while nonlinear basis functions can represent nonlinearity of data. Now, the constraints for the marginal separation into two half hyperspaces can be written as:

$$\begin{cases} y_i = -1 & \text{if } \mathbf{w}^\top \Phi(\mathbf{x}_i) + b \leq -1 \\ y_i = 1 & \text{if } \mathbf{w}^\top \Phi(\mathbf{x}_i) + b \geq 1 \end{cases} \quad \Rightarrow \quad y_i f(\mathbf{x}_i) \geq 1.$$

In Equation (2.2), $\mathbf{w}$ is an orthogonal vector to the hyperplane, so the distance of the margin is the sum of the distances between the hyperplane and the closest data samples in both classes. The distances are equivalent to both classes since the hyperplane is placed in the center of two data

sets. We find the distance $d = \frac{1}{\|\mathbf{w}\|}$, and the margin is $\frac{2}{\|\mathbf{w}\|}$. Since maximizing $\frac{2}{\|\mathbf{w}\|}$ is equivalent to minimizing $\frac{\|\mathbf{w}\|^2}{2}$, the hard margin SVM that seeks for a maximum margin can be written with correct classification constraints as described below:

$$\begin{aligned} \text{minimize} \quad & \frac{\|\mathbf{w}\|^2}{2} \\ \text{subject to} \quad & y_i(\mathbf{w}^\top \Phi(\mathbf{x}) + b) \geq 1. \end{aligned}$$

2.3.1 Soft Margin

In practice, the data is not always linearly separable. In such data, relaxing the correct classification constraints can improve the performance. Thus, we apply the soft margin approach. Some previous theoretical and experimental study shows that soft margin generally performs better than hard margin SVM [73]. We can define the slack variables $\xi_i > 0$ to allow the classification errors:

$$y_i(\mathbf{w}^\top \Phi(x) + b) \geq 1 - \xi_i.$$

Now, adding control parameter C, we can rewrite the linear program.

$$\begin{aligned} \text{minimize} \quad & \frac{\|\mathbf{w}\|^2}{2} + C \sum_{i=1}^{m} \xi_i \\ \text{subject to} \quad & y_i(\mathbf{w}^\top \Phi(x) + b) \geq 1 - \xi_i \\ & \xi_i > 0 \end{aligned}$$

where m is the number of points. C controls the conflicting objectives, maximizing the margin and minimizing the sum of errors. When C is large, a large penalty is given to errors, it reduces the

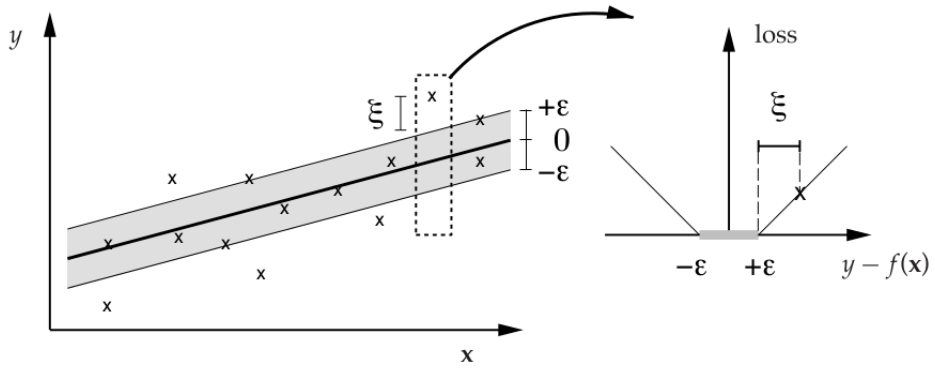


Figure 2.2: Soft margin linear support vector regression with ϵ-tube. The figure is from [1]

margin that minimizes the error term. When C is small, it allows more errors resulting in margin increase.

2.3.2 Support Vector Regression

For regression, let us first define the regression function f with a basis function Φ,

$$f(\mathbf{x}) = \mathbf{w}^\top \Phi(\mathbf{x}) + b \tag{2.3}$$

where $\mathbf{w}$ is a weight vector and b is a bias. ϵ-intensive loss function [74] in Figure 2.2 allows margin errors that estimation values of $f(\mathbf{x})$ are located inside the ϵ-tube while penalizing estimations outside the tube. The loss function can be written as:

$$L(\mathbf{x}_i) = \begin{cases} 0 & \text{if } |f(\mathbf{x}_i) - t_i| \leq \epsilon \\ |t_i - f(\mathbf{x}_i)| - \epsilon & \text{otherwise} \end{cases}$$

with a target value t_i for an input vector $\mathbf{x}_i$. This is beneficial for enforcing the possible estimation error under the ϵ value. To avoid overfitting, the weight vector $\mathbf{w}$ needs to be as simple as possible,

which leads to the following objective function:

$$\begin{aligned} \text{minimize} \quad & \frac{\|\mathbf{w}\|^2}{2} \\ \text{subject to} \quad & (\mathbf{w}^\top \Phi(\mathbf{x}_i) + b) - t_i \leq \epsilon \\ & t_i - (\mathbf{w}^\top \Phi(\mathbf{x}_i) + b) \leq \epsilon. \end{aligned}$$

Similar to SVM for classification, by introducing slack variables ξ^+ and ξ^-, soft margin SVR can provide robust estimation without increasing the ϵ precision to cope with outliers. The objective function evolves as in Vapnik [74]:

$$\begin{aligned} \text{minimize} \quad & \frac{\|\mathbf{w}\|^2}{2} + C \sum_{i=1}^{m} (\xi^+ + \xi^-) \\ \text{subject to} \quad & (\mathbf{w}^\top \Phi(\mathbf{x}_i) + b) - t_i \leq \epsilon - \xi^- \\ & t_i - (\mathbf{w}^\top \Phi(\mathbf{x}_i) + b) \leq \epsilon + \xi^+ \\ & \xi^+, \xi^- \geq 0. \end{aligned}$$

The constant $C > 0$ controls the tradeoff between the conflicting goals, the tolerance to errors and simple weight solutions.

2.3.3 Reinforcement Learning with SVM

For reinforcement learning application, the function approximation needs to be updated gradually as the samples from interaction of the world are gathered. This needs an incremental version of support vector regression. Ma, et al., [75] proposed an accurate online support vector regression (AOSVR) that gradually adds or removes a sample from training set while updating SVR function.

Lee, et al., [61, 62] successfully adopted the online SVR as a function approximator in reinforcement learning. However, their approach suffers from the increasing number of support vectors as the complexity of the problem grows.

2.4 Bayesian Learning

Bayesian learning [41, 76] uses probability to represent statistical problems. With random variables, it mathematically models our beliefs of an event and updates them as we observe more data. This approach is different from the *frequentist*'s approach that measures the frequency of an event without considering any belief about the experiment. Bayesian learning framework is based on the Bayes' rule:

$$P(\theta|D) = \frac{P(D|\theta)P(\theta)}{P(D)}$$

There are three key elements in Bayes' rule, *likelihood* $P(D|\theta)$, *prior* $P(\theta)$, and *posterior* $P(\theta|D)$. The prior distribution represents the belief before we observe the data D. When we consider the coin flip example, if you think the coin that in your pocket is fair with 90% confidence, the prior can be modeled with a probability of 0.9. The likelihood is the probability of observed D when the data is generated by the model parameter θ. For instance, with a fair coin, the likelihood that we see the head will be 0.5. Last, the posterior is updated our belief after collecting evidence from observations. To take a fair coin case in, a series of observations can reinforce or weaken our belief of the fairness of the coin in your pocket. After formulating a probability model for the data, Bayesian learning follows the three steps to learn:

1. decide prior distributions for weights,

2. compute likelihood from observation,

3. determine posterior distribution.

As an example, the next section describes the Bayesian linear regression model, which is the basis of Gaussian processes and relevance vector machines in the following sections.

2.4.1 Bayesian Linear Regression

Similar to the linear regression, Bayesian linear regression makes a linear model assumption y. In probabilistic framework, in addition to the linear model, we define an explicit Gaussian noise $\epsilon \simeq p(\epsilon|\sigma^2) = \mathcal{N}(0, \sigma^2)$. Let $y(x_n; \mathbf{w}) = \sum_{m=1}^{M} w_m \Phi_m(x_n) = \mathbf{\Phi w}$. The target can be modeled as $t_n = y(x_n; \mathbf{w}) + \epsilon_n$. The zero mean Gaussian noise leads the target distribution to be Gaussian as follows:

$$\begin{aligned} p(t_n|x_n, \mathbf{w}, \sigma^2) &= \mathcal{N}(y(x_n; \mathbf{w}), \sigma^2) \\ &= \frac{1}{\sqrt{2\pi\sigma^2}} \exp\Big[- \frac{(t_n - y(x_n; \mathbf{w}))^2}{2\sigma^2} \Big] \end{aligned}$$

With an assumption that each target is independent, we define the likelihood over all N data samples with product rule where the vectors $\mathbf{t} = (t_1, t_2, \cdots, t_n)^\top$ and $\mathbf{x} = (x_1, x_2, \cdots, x_n)^\top$:

$$p(\mathbf{t}|\mathbf{x}, \mathbf{w}, \sigma^2) = \prod_{n=1}^{N} p(t_n|x_n, \mathbf{w}, \sigma^2). \tag{2.4}$$

The goal is maximizing the likelihood, so we first define the logarithm of the likelihood for easier computation:

$$L(\mathbf{w}) = -\frac{N}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_{n=1}^{N}[t_n - y(x_n;\mathbf{w})]^2$$
$$= -\frac{1}{2\sigma^2}(\mathbf{t} - \mathbf{\Phi}\mathbf{w})^\top(\mathbf{t} - \mathbf{\Phi}\mathbf{w}) + const.$$

Setting the first derivative of log-likelihood w.r.t $\mathbf{w}$ to zero, we reach the equivalent to least square solution:

$$\frac{\partial L(\mathbf{w})}{\partial \mathbf{w}} = \frac{1}{\sigma^2}[\mathbf{\Phi}^\top\mathbf{t} - \mathbf{\Phi}^\top\mathbf{\Phi}\mathbf{w}] = 0$$
$$\mathbf{w} = (\mathbf{\Phi}^\top\mathbf{\Phi})^{-1}\mathbf{\Phi}^\top\mathbf{t}.$$

In ridge regression, to avoid over-fitting, we prefer a smooth solution that is achieved by penalizing weights. Similarly, in Bayesian framework, we add a prior distribution to represent the degree of belief over weights. In this way, we can control the complexity of model. The zero mean Gaussian prior for m dimensional weight $\mathbf{w}$ can be defined as

$$p(\mathbf{w}|\alpha) = \mathcal{N}(\boldsymbol{\mu}_0, \alpha^{-1}\mathbf{I}) = \prod_{m=1}^{M}\sqrt{\frac{\alpha}{2\pi}}\exp(-\frac{\alpha}{2}w_m^2),$$

where $\boldsymbol{\mu}_0 = 0$. From the definition of $\mathbf{t}$ as the zero-mean Gaussian noise addition to the linear model y, the likelihood is written

$$p(\mathbf{t}|\mathbf{w}) \sim \mathcal{N}(\Phi\mathbf{w}, \beta^{-1}\mathbf{I}),$$

where the inverse variance $\beta = \sigma^{-2}$ because $\mathbf{t} = \mathbf{y} + \epsilon$. The posterior distribution can be computed from the combination of the likelihood and the prior distribution as follows:

$$\begin{aligned} p(\mathbf{w}|\mathbf{t}) &\propto p(\mathbf{t}|\mathbf{w})p(\mathbf{w}|\alpha) \\ &\propto \mathcal{N}(\Phi\mathbf{w}, \beta^{-1}\mathbf{I})\mathcal{N}(\boldsymbol{\mu}_0, \alpha^{-1}\mathbf{I}). \end{aligned}$$

Now, as in Chapter 2 Bishop [41], let $\mathbf{z} = \begin{pmatrix} \mathbf{w} \\ \\ \mathbf{t} \end{pmatrix}$, the joint distribution for $\mathbf{w}$ and $\mathbf{t}$ can be written in terms of $\mathbf{z}$:

$$p(\mathbf{z}) = p(\mathbf{w}, \mathbf{t}) = p(\mathbf{w}|\mathbf{t}).$$

The logarithm on this joint distribution leads to

$$\begin{aligned} \log p(\mathbf{z}) &= \log p(\mathbf{t}|\mathbf{w}) + \log p(\mathbf{w}) \\ &= -\frac{1}{2}(\mathbf{t} - \boldsymbol{\Phi}\mathbf{w})^\top \beta\mathbf{I}(\mathbf{t} - \boldsymbol{\Phi}\mathbf{w}) - \frac{1}{2}(\mathbf{w} - \boldsymbol{\mu}_0)^\top \alpha\mathbf{I}(\mathbf{w} - \boldsymbol{\mu}_0) + \text{const} \\ &= -\frac{1}{2}(\mathbf{t}^\top\beta\mathbf{I}\mathbf{t} - \mathbf{t}^\top\beta\mathbf{I}\boldsymbol{\Phi}\mathbf{w} - \mathbf{w}^\top\boldsymbol{\Phi}^\top\beta\mathbf{I}\mathbf{t} + \mathbf{w}^\top\boldsymbol{\Phi}^\top\beta\mathbf{I}\boldsymbol{\Phi}\mathbf{w}) \\ &\quad - \frac{1}{2}(\mathbf{w}^\top\alpha\mathbf{I}\mathbf{w} - \mathbf{w}^\top\alpha\mathbf{I}\boldsymbol{\mu}_0 - \boldsymbol{\mu}_0^\top\alpha\mathbf{I}\mathbf{w} + \boldsymbol{\mu}_0^\top\alpha\mathbf{I}\boldsymbol{\mu}_0) + \text{const} \qquad (2.5) \\ &= -\frac{1}{2}\mathbf{w}^\top(\alpha\mathbf{I} + \boldsymbol{\Phi}^\top\beta\mathbf{I}\boldsymbol{\Phi})\mathbf{w} - \frac{1}{2}\mathbf{t}^\top\beta\mathbf{I}\mathbf{t} + \frac{1}{2}\mathbf{t}^\top\beta\mathbf{I}\boldsymbol{\Phi}\mathbf{w} + \frac{1}{2}\mathbf{w}^\top\boldsymbol{\Phi}^\top\beta\mathbf{I}\mathbf{t} + \text{const} \qquad (2.6) \\ &= -\frac{1}{2}\begin{pmatrix} \mathbf{w} \\ \\ \mathbf{t} \end{pmatrix}\begin{pmatrix} \alpha\mathbf{I} + \boldsymbol{\Phi}^\top\beta\mathbf{I}\boldsymbol{\Phi} & -\boldsymbol{\Phi}^\top\beta\mathbf{I} \\ \\ -\beta\mathbf{I}\boldsymbol{\Phi} & \beta\mathbf{I} \end{pmatrix}\begin{pmatrix} \mathbf{w} \\ \\ \mathbf{t} \end{pmatrix} \\ &= -\frac{1}{2}\mathbf{z}^\top\mathbf{R}\mathbf{z} \qquad (2.7) \end{aligned}$$

with the precision matrix $\mathbf{R}$

$$\mathbf{R} = \begin{pmatrix} \alpha\mathbf{I} + \mathbf{\Phi}^\top\beta\mathbf{I}\mathbf{\Phi} & -\mathbf{\Phi}^\top\beta\mathbf{I} \\ -\beta\mathbf{I}\mathbf{\Phi} & \beta\mathbf{I} \end{pmatrix} = \begin{pmatrix} \alpha\mathbf{I} + \beta\mathbf{\Phi}^\top\mathbf{\Phi} & -\beta\mathbf{\Phi} \\ -\beta\mathbf{\Phi} & \beta\mathbf{I} \end{pmatrix}.$$

From Equation (2.6), $\mathbf{\Phi}^\top\beta\mathbf{I}$ is symmetric. By completing square w.r.t. $\mathbf{w}$, we can get $\boldsymbol{\mu}$ and $\mathbf{\Sigma}$ for the posterior $p(\mathbf{w}|\mathbf{t}, \mathbf{x}, \alpha, \beta) \sim \mathcal{N}(\mu, \Sigma)$:

$$\boldsymbol{\mu} = \beta(\alpha\mathbf{I} + \beta\mathbf{\Phi}^\top\mathbf{\Phi})^{-1}\mathbf{\Phi}^\top\mathbf{t}$$

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

This solution is equivalent to the ridge regression solution with $\lambda = \frac{\alpha}{\beta}$.

In Equation (2.7), the covariance matrix (the inverse precision) for the joint distribution is

$$\mathrm{Cov}[\mathbf{z}] = \mathbf{R}^{-1} = \begin{pmatrix} \alpha^{-1}\mathbf{I} & \alpha^{-1}\mathbf{\Phi}^\top \\ \mathbf{\Phi}\alpha^{-1} & \beta^{-1} + \alpha^{-1}\mathbf{\Phi}^\top\mathbf{\Phi} \end{pmatrix}. \tag{2.8}$$

From Equation (2.5), after inserting $\boldsymbol{\mu}$ for $\boldsymbol{\mu}_0$ and applying completing the square, we retrieve the first order terms to get the following expected value:

$$\mathbb{E}[\mathbf{z}] = R^{-1}\begin{pmatrix} \alpha\boldsymbol{\mu} \\ 0 \end{pmatrix} = \begin{pmatrix} \boldsymbol{\mu} \\ \mathbf{\Phi}\boldsymbol{\mu} \end{pmatrix}. \tag{2.9}$$

From the Equation (2.8) and Equation (2.9), the mean and covariance for the marginal probability for predicted target distribution $p(\mathbf{t}|\mathbf{x}, \alpha, \beta) \sim \mathcal{N}(\mathbf{m}, \mathbf{S})$ can be computed as

$$\begin{aligned} \mathbf{m} &= \mathbf{\Phi}\boldsymbol{\mu} \\ \mathbf{S} &= \beta^{-1} + \alpha^{-1}\mathbf{\Phi}^{\top}\mathbf{\Phi}. \end{aligned} \tag{2.10}$$

2.4.2 Gaussian Processes

Gaussian processes (GPs) are the generalization of the multivariate normal to infinite dimension for continuing time-series data. When we have time series data, if we assume a Gaussian distribution on each point, we can represent a window of the time-series as multivariate normal. In previous section, we describe the Bayesian learning that infers the posterior for the parameter, $p(\theta|D)$, since the function f is parametric with θ. Gaussian processes instead directly infer the function $p(f|D)$ by defining prior distribution $p(f)$ [77, 78]. Since representing an infinite set of function's values is not possible, when we observe the inputs $\mathbf{x}_i$ and the outputs $y_i = f(\mathbf{x}_i)$, GPs select m arbitrary samples $\mathbf{x}_k, ..., \mathbf{x}_{k+m}$ and assume the $p(f(\mathbf{x}_k), \ldots, f(\mathbf{x}_{k+m}))$ jointly as Gaussian distribution. Here, GPs model the mean and covariance as $m(\mathbf{x})$ and $\mathrm{Cov}^f(\mathbf{x}) = \mathrm{Cov}(f(\mathbf{x}))$. Usually a GP models the covariance function with a positive definite kernel $\mathrm{Cov}^f_{ij}(\mathbf{x}) = k(\mathbf{x}_i, \mathbf{x}_j)$. Now, we write Gaussian processes as follows:

$$f(\mathbf{x}) = GP(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')),$$

where

$$m(\mathbf{x}) = \mathbb{E}[f(\mathbf{x}))]$$

$$k(\mathbf{x}, \mathbf{x}') = \mathbb{E}[(f(\mathbf{x}) - m(\mathbf{x}))(f(\mathbf{x}') - m(\mathbf{x}'))^\top].$$

This can be represented with joint Gaussian:

$$p(f|\mathbf{X}) = \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}),$$

where $\boldsymbol{\mu} = [m(\mathbf{x}_1), \ldots, m(\mathbf{x}_N)]^\top$ and $\boldsymbol{\Sigma}_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$.

When the new data x_* comes, we can compute the prediction y_* with

$$p(y_*|\mathbf{x}_*, \mathbf{X}, \mathbf{y}) = \int p(y_*|f, \mathbf{x}_*)p(f|\mathbf{X}, \mathbf{y})df.$$

Gaussian processes are collection of random variables, so the distribution of a subset does not affect the distribution of the other subset. This means that when the joint distribution of $\mathbf{y} = (y_1, \ldots, \mathbf{y}_N) \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, the distribution of $\mathbf{y}_{1:k} = (y_1, \ldots, \mathbf{y}_k) \sim \mathcal{N}(\boldsymbol{\mu}_{1:k}, \boldsymbol{\Sigma}_{1:k,1:k})$ and the distribution of the other subset $\mathbf{y}_{k+1:N} \sim \mathcal{N}(\boldsymbol{\mu}_{k+1:N}, \boldsymbol{\Sigma}_{k+1:N,k+1:N})$. This is called *consistency* requirement, or the *marginalization property*. This property is used to make a prediction from the training samples.

Suppose we have n training samples $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_n]^\top$ and corresponding labels $\mathbf{y} = (y_1, y_2, ..., y_n)$. We want to make predictions $\mathbf{y}_*$ for new points $\mathbf{X}_*$. Then, the prior joint distribution with zero-mean Gaussian is

$$\begin{pmatrix} \mathbf{y} \\ \mathbf{y}_* \end{pmatrix} \sim N\left(\mathbf{0}, \begin{pmatrix} \boldsymbol{\Sigma} & \boldsymbol{\Sigma}_*^\top \\ \boldsymbol{\Sigma}_* & \boldsymbol{\Sigma}_{**} \end{pmatrix}\right),$$

where the covariance matrices are $\boldsymbol{\Sigma} = k(\mathbf{X}, \mathbf{X})$, $\boldsymbol{\Sigma}_* = k(\mathbf{X}, \mathbf{X}_*)$, and $\boldsymbol{\Sigma}_{**} = k(\mathbf{X}_*, \mathbf{X}_*)$. By conditioning the prior with zero mean Gaussian, the posterior predictive distribution is

$$p(\mathbf{y}_*|\mathbf{X}_*, \mathbf{X}, \mathbf{y}) \sim \mathcal{N}(\boldsymbol{\mu}', \boldsymbol{\Sigma}')$$

$$\boldsymbol{\mu}' = \boldsymbol{\Sigma}_* \boldsymbol{\Sigma}^{-1} \mathbf{y}$$

$$\boldsymbol{\Sigma}' = \boldsymbol{\Sigma}_{**} - \boldsymbol{\Sigma}_* \boldsymbol{\Sigma}^{-1} \boldsymbol{\Sigma}_*^\top.$$

Now, the $\mathbf{y}_*$ can be sampled from the posterior distribution for predictions.

2.4.3 Bayesian Reinforcement Learning

Bayesian learning models have shown some success and efficiency in reinforcement learning. Bayesian Q-learning [63], Gaussian process temporal difference learning (GPTD) [64–66], and linear Bayesian reinforcement learning [67] have been applied to estimate value functions. Unlike classical RL, Bayesian RL suggests feasible solutions to the exploration-exploitation tradeoffs [63, 79]. Moreover, Bayesian RL can choose samples to learn especially when the sampling cost is expensive [80]. Here are the list of the advantages of Bayesian reinforcement learning that is summarized by Ghavamzadeh, et al., [81]:

- Exploration-exploitation control: Bayesian reinforcement learning provides a principled way to tackle the tradeoff. By using the hyperparameters, the posterior captures the knowledge

about the states so that an agent can choose actions that maximizes the expected gain in the estimated values.

- Regularization: An assumption on a prior distribution relaxes the effects of the biased sampling from a finite data set. This results in effective regularization: we can avoid the quick drift-away of weights from true values and, at the same time, we can have slow convergence.

- Uncertainty Handling: Nilim and El Ghaoui [82] pointed out that when there is modeling error or uncertainty in the model, frequentist approaches are either conservative or computationally infeasible. Bayesian reinforcement learning provides a principled way to solve the difficult problems.

2.5 Relevance Vector Machines

RVM [39, 40] is a Bayesian sparse kernel technique that shares many characteristics of SVM while avoiding the limitations of SVM such as point-estimate output, necessity of parameter search, kernel requirement for Mercer's Theorem, and no sparse solution guaranteed [41]. By using individual hyper-parameters for each weight, probabilistic models only for relevance vectors are maintained, which force RVM to be sparse. A probabilistic output of RVM captures the uncertainty in its predictions. RVM is less-sensitive to hyper-parameters than SVM, and the kernel function does not need to be positive definite.

For N samples $\{\mathbf{x_n}, t_n\}_{n=1}^{N}$ with data input $\mathbf{x_n}$ and target t_n, the regression can be modeled as

$$t_n = f(\mathbf{x_n}; \mathbf{w}) + \epsilon_n,$$

where $\mathbf{w}$ is the weight vector for the regression function f and ϵ_n is zero-mean Gaussian noise with variance $\beta^{-1} = \sigma^2$. This implies

$$\mathbb{P}(t_n|\mathbf{x_n}) = \mathcal{N}(f(\mathbf{x_n}; \mathbf{w}), \sigma^2).$$

Assuming independence, the likelihood function is given by

$$\mathbb{P}(\mathbf{t}|\mathbf{x}, \mathbf{w}, \beta) = \prod_{n=1}^{N} \mathcal{N}(f(\mathbf{x_n}; \mathbf{w}), \beta^{-1}),$$

where $\mathbf{t} = (t_1, t_2, \cdots, t_N)^\top$ and $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_N]$. RVM mirrors the structure of an SVM: a linear model with basis function that can be represented by a kernel function that associates one data point at a time:

$$f(\mathbf{x}) = \sum_{n=1}^{N} w_n k(\mathbf{x}, \mathbf{x_n}) + b.$$

Let $\phi_i(\mathbf{x}) = k(\mathbf{x}, \mathbf{x_i})$ be the basis function with kernel k. $\mathbf{\Phi}$ can be defined 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 $\phi(\mathbf{x_n}) = [1, \phi_1(\mathbf{x_n}), \phi_2(\mathbf{x_n}), \cdots, \phi_N(\mathbf{x_n})]^\top$. For smooth regression, like regularization in ridge regression, zero-mean Gaussian prior over the weight $\mathbf{w}$ is defined:

$$\mathbb{P}(\mathbf{w}|\boldsymbol{\alpha}) = \prod_{i=0}^{N} \mathcal{N}(w_i|0, \alpha_i^{-1})$$

with $\boldsymbol{\alpha}$, a vector of $N+1$ hyper-parameters. That is, small magnitude weights are more likely selected. Distributions for hyper-parameters $\boldsymbol{\alpha}$ and β are defined as Gaussian distributions. However, as Tipping, et al., noted [39, 83], Gamma distribution can be chosen for the distribution as written here:

$$\mathbb{P}(\boldsymbol{\alpha}) = \prod_{i=0}^{N} \mathrm{Gamma}(\alpha_i|a,b)$$

$$\mathbb{P}(\beta) = \mathrm{Gamma}(\beta|c,d)$$

where $\mathrm{Gamma}(\alpha|a,b) = \Gamma(a)^{-1}b^a\alpha^{a-1}e^{-b\alpha}$ with $\Gamma(a) = \int_0^\infty t^{a-1}e^{-t}dt$. With Gamma prior distribution, the posterior becomes a student-t distribution that can be more flexible to represent heavy-tail distribution. With Gamma prior, it can have more sparse solution; also, it can represent Gaussian with a very large degree of freedom parameter in t distribution.

A broad prior over hyper-parameters makes posterior probability mass go to infinity, so posterior probability of the associated weights to be concentrated at zero. This makes the corresponding inputs to be *irrelevant*. This is called *switching-off*. By assigning an individual hyper-parameter to each weight, an RVM can switch irrelevant weights or bases off and produce a *sparse* solution.

By marginalizing over uncertain weights and hyper-parameters, the predictive distribution for new data $\mathbf{x}', t'$ can be computed as

$$\mathbb{P}(t'|\mathbf{x}', \mathbf{t}, \mathbf{X}) = \int \mathbb{P}(t'|\mathbf{x}', \mathbf{w}, \boldsymbol{\alpha}, \beta)\mathbb{P}(\mathbf{w}, \boldsymbol{\alpha}, \beta|\mathbf{t}, \mathbf{X})d\mathbf{w}d\boldsymbol{\alpha}d\beta$$

by averaging the model for new target. Since $\mathbb{P}(\mathbf{w}, \boldsymbol{\alpha}, \beta|\mathbf{t}, \mathbf{X})$ is analytically intractable, so is $\mathbb{P}(t'|\mathbf{x}', \mathbf{t}, \mathbf{X})$. Thus, we need to approximate these. Commonly, type-2 maximum likelihood,

evidence approximation, can be used. Tipping et al. [39, 41], discussed the use of an expectation-maximization (EM) algorithm to maximize $\mathbb{P}(t'|\mathbf{x}', \mathbf{t}, \mathbf{X})$, yet direct optimization via evidence approximation converged faster. Before the approximation, we can decompose $\mathbb{P}(\mathbf{w}, \boldsymbol{\alpha}, \beta|\mathbf{t}, \mathbf{X})$ by using the product rule of conditional probability:

$$\mathbb{P}(\mathbf{w}, \boldsymbol{\alpha}, \beta|\mathbf{t}, \mathbf{X}) = \mathbb{P}(\mathbf{w}|\boldsymbol{\alpha}, \beta, \mathbf{t}, \mathbf{X})\mathbb{P}(\boldsymbol{\alpha}, \beta|\mathbf{t}, \mathbf{X}).$$

Analytically computing the weight posterior distribution, we have a convolution of Gaussian distributions that combines the linear model of Gaussian likelihood and a Gaussian prior:

$$\mathbb{P}(\mathbf{w}|\boldsymbol{\alpha}, \beta, \mathbf{t}, \mathbf{X}) \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$$

where the mean and covariance are given by

$$\begin{aligned} \boldsymbol{\mu} &= \beta \boldsymbol{\Sigma} \boldsymbol{\Phi}^\top \mathbf{t} \\ \boldsymbol{\Sigma} &= (\beta \boldsymbol{\Phi}^\top \boldsymbol{\Phi} + \boldsymbol{\alpha}\mathbf{I})^{-1}. \end{aligned} \tag{2.11}$$

Marginal likelihood over the weight parameters represents zero-mean Gaussian with $\mathbf{A} = \boldsymbol{\alpha}\mathbf{I}$:

$$\begin{aligned} \mathbb{P}(\mathbf{t}|\mathbf{X}, \mathbf{w}, \boldsymbol{\alpha}, \beta) &= \int \mathbb{P}(\mathbf{t}|\mathbf{X}, \mathbf{w}, \beta)\mathbb{P}(\mathbf{w}|\boldsymbol{\alpha})d\mathbf{w} \\ &\sim \mathcal{N}(\mathbf{m}, \mathbf{C}), \end{aligned}$$

where the mean and the variance are similar to Equation (2.10):

$$\begin{aligned}\mathbf{m} &= \mathbf{\Phi}\boldsymbol{\mu} \\ \mathbf{C} &= \beta^{-1}\mathbf{I} + \mathbf{\Phi}\mathbf{A}^{-1}\mathbf{\Phi}^{\top}.\end{aligned} \tag{2.12}$$

Using the evidence approximation and omitting the input $\mathbf{x}$ for uncluttered notation, we can rewrite the predictive distribution obtained by marginalizing over the weight $\mathbf{w}$:

$$\mathbb{P}(\mathbf{t}|\boldsymbol{\alpha},\beta) = (\frac{\beta}{2\pi})^{\frac{N}{2}}(\frac{\alpha}{2\pi})^{\frac{M}{2}}\int e^{-E(\mathbf{w})}d\mathbf{w} = (\frac{\beta}{2\pi})^{\frac{N}{2}}(\frac{\alpha}{2\pi})^{\frac{M}{2}}e^{-E(\mathbf{w})}(2\pi)^{\frac{M}{2}}|\mathbf{A}|^{-\frac{1}{2}},$$

where M is the dimensionality of $\mathbf{w}$, and regularized error is defined as $E(\mathbf{w}) = \frac{\beta}{2}\|\mathbf{t} - \mathbf{\Phi}\mathbf{w}\|^2 + \frac{\alpha}{2}\mathbf{w}^{\top}\mathbf{w}$ [41]. For computational efficiency, the logarithm of the marginal likelihood is used:

$$L(\boldsymbol{\alpha},\beta) = \log\mathbb{P}(\mathbf{t}|\boldsymbol{\alpha},\beta) = \frac{M}{2}\ln\boldsymbol{\alpha} + \frac{N}{2}\ln\boldsymbol{\beta} - E(\mathbf{w}) - \frac{1}{2}\ln|\mathbf{A}| - \frac{N}{2}\ln 2\pi. \tag{2.13}$$

Maximizing $L(\boldsymbol{\alpha},\beta)$ with respect to hyper-parameters $\boldsymbol{\alpha}$ and β gives the following iterative update rule:

$$\begin{aligned}\gamma_i &= 1 - \alpha_i\Sigma_{ii} \\ \alpha_i &\leftarrow \frac{\gamma_i}{\mu_i^2} \\ \beta &\leftarrow \frac{N - \sum_i \gamma_i}{\|\mathbf{t} - \mathbf{\Phi}\boldsymbol{\mu}\|^2}.\end{aligned} \tag{2.14}$$

Here, γ_i is interpreted as a measure of how well-determined the weight w_i is by the data.

When α_* and β_* are found from training, the predictive distribution can be marginalized as Gaussian with the following mean and variance:

$$f(\mathbf{x}_*) = \boldsymbol{\mu}^\top \phi(\mathbf{x}_*) \tag{2.15}$$

$$\sigma^2(f(\mathbf{x}_*)) = {\beta_*}^{-1} + \phi(\mathbf{x}_*)^\top \boldsymbol{\Sigma} \phi(\mathbf{x}_*). \tag{2.16}$$

The trained model contains a large number of very large $\boldsymbol{\alpha}$ that make the weights for corresponding samples zero. Analogous to support vectors in SVM, the data samples with non-zero weights are called *relevance vectors* (RVs). Fast marginal likelihood maximization [40] maintains only the relevance vectors for bases and improves both computational speed, and we implements this version of RVM.

For reinforcement learning problems, we compose the input with state-action pair, $\mathbf{x}_t = (\mathbf{s}_t, \mathbf{a}_t)$, and the Q estimates are computed with the predictive distribution:

$$\begin{aligned} Q(\mathbf{s}_t, \mathbf{a}_t) &\sim \mathcal{N}\left(f(\mathbf{x}_t), \sigma^2(\mathbf{x}_t)\right) \\ &\sim \mathcal{N}\left(\mu_t^\top \phi(\mathbf{s}_t, \mathbf{a}_t), \beta_t^{-1} + \phi(\mathbf{x}_t)^\top \Sigma \phi(\mathbf{x}_t)\right). \end{aligned}$$

2.5.1 Reinforcement Learning with RVMs

Rexakis, et al., [84] have applied relevance vector machines to reinforcement learning. They suggested a directed rollout classfication policy iteration (DRCPI-RVM). This work is motivated by the limitations of their previous work with support vector machines. To avoid the policy simulations, they proposed a directed rollout approach with support vector machines. Although the proposed approach was successful, it had to bear the computational overhead from the growing

number of support vectors. They have shown that RVM-based approaches can overcome the problems in the computational overhead.

Chapter 3

Online RVM-based Reinforcement Learning

In this chapter, we introduce the application of relevance vector machines to online reinforcement learning. In this chapter, relevance vector machines (RVMs) are used as a simple function approximator. For successful application, we modify the traditional RVM to different versions to lower the computation and memory limits so that it can be applied to online reinforcement learning. The modified version with a buffer ensures re-examination of deleted samples for basis candidates. Our multivariate version uses multiple RVMs to estimate multiple output values which potentially share the hyperparameter β. By applying these modified versions to a reinforcement learning problem, we examine the efficiency and limitations of RVM function approximation.

As discussed in Section 2.2, various function approximations have extended the applications of reinforcement learning to large state space problems successfully. However, 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 [85].

In Section 2.4, we discussed Bayesian reinforcement learning models and their advantages of success and efficiency. Since Bayesian approaches to reinforcement learning have shown advantages in the exploration-exploitation tradeoff and in lower sampling costs, by inheriting the Bayesian advantages, we expect further improvement of learning performance.

To improve stability of learning with the previous function approximators and to solve the growing number of support vectors with the support vector machines, we apply the sparse Bayesian learning model, relevance vector machines [39, 40, 86], as a function approximator. Relevance

vector machines 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 SVM, the RVM function approximators are expected to improve the robustness of learning and lessen the memory and computation overhead. Online RVM function approximators are tested to examine their efficacy. The RVM modification details, test experiments and results are discussed in the following sections.

3.1 Online RVM Variations

With an incremental optimization, Tipping, et al., [40], proposed a more computationally efficient algorithm than $O(N^3)$ in their previous work. Starting without an RV, at each iteration, it adds, re-estimates, or removes one RV. When adding an RV, the hyperparameter α_i is adjusted to maximize the marginal likelihood. They decomposed Equation (2.12), $\mathbf{C} = \mathbf{C}_{-i} + \alpha_i^{-1}\phi_i\phi_i^\top$ where $-i$ represents $\mathbf{C}$ without a relevance vector i contribution,

$$\mathbf{C}_{-i} = \beta\mathbf{I} + \sum_{j \neq i} \alpha_j \phi_j \phi_j^\top .$$

The logarithm of the marginal likelihood in Equation (2.13) can be written as

$$L(\boldsymbol{\alpha}) = L(\boldsymbol{\alpha}_{-i}) + l(\alpha_i)$$
$$l(\alpha_i) = \frac{1}{2}\left(\log \alpha_i - \log(\alpha_i + s_i) + \frac{q_i^2}{\alpha_i + s_i}\right)$$

where $s_i = \phi_i^\top \mathbf{C}_{-i}^{-1}\phi_i$ and $q_i = \phi_i^\top \mathbf{C}_{-i}^{-1}\mathbf{t}$. The *sparsity factor*, s_i measures the extent of overlaps of ϕ_i with the existing other bases. The *quality factor*, $q_i = \sigma^{-2}\phi_i^\top(\mathbf{t} - \mathbf{y}_{-i})$, measures the alignment error of ϕ_i when the i-th output is excluded.

From the derivation in [86], there is a single maximum at $\alpha_i = \frac{s_i^2}{q_i^2 - s_i}$ if the denominator $q_i^2 > s_i$. If the denominator is zero or negative, α_i is not valid, so we set it to ∞. Thus, the algorithm adds an RV or re-estimates the RV that is indexed by i when $q_i^2 > s_i$; otherwise it removes the RV.

To apply this to a reinforcement learning function approximation, RVM must learn a policy incrementally, so it needs to be memory-efficient. However, the existing algorithm needs to store the kernel matrix $\mathbf{\Phi}$, and the memory requirement is $O(N^2)$ where N is total number of interactions with an environment. For this, we adopt the incremental/decremental steps in online support vector regressions that add or remove a vector to a kernel matrix at a time [75, 87]. This results in the online RVM (oRVM) with the following incremental and decremental steps to manage the kernel matrix $\mathbf{\Phi}$. In this approach, oRVMs forget insignificant samples and remember only the relevance vectors that can cover all the Q value space.

Incremental step: When a new relevance vector is added, the proposed algorithm adds a vector ϕ_i to $\mathbf{\Phi}$ as follows:

$$\mathbf{\Phi}^{new} = \begin{bmatrix} & & & \phi_{1i} \\ & \mathbf{\Phi} & & \vdots \\ & & & \phi_{ni} \\ \phi_{1i} & \cdots & \phi_{ni} & \phi_{ii} \end{bmatrix}$$

where $\phi_{ji} = k(x_j, x_i)$ and i represents an index for a new input.

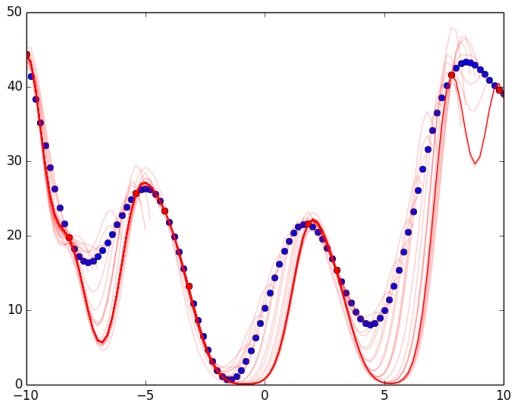
Decremental step: When $q_i^2 \leq s_i$, oRVM sets $\alpha = \infty$ and decrements the kernel matrix, by removing the row i and column i.

$$\mathbf{\Phi}^{new} = \begin{bmatrix} \mathbf{\Phi}_{(1:i-1,1:i-1)} & \mathbf{\Phi}_{(1:i-1,i+1:N)} \\ \mathbf{\Phi}_{(i+1:N,1:i-1)} & \mathbf{\Phi}_{(i+1:N,i+1:N)} \end{bmatrix}$$

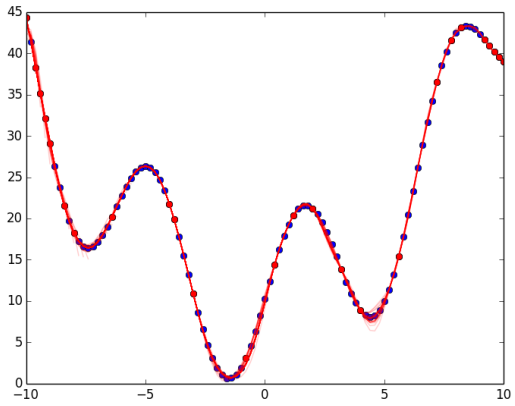
Figure 3.1a shows oRVM applied to the regression problem of approximating a quadratic sine curve when the data input is received in the order of low to high x values. Since it discards samples at the moment of data input, the fit is a bit away from the true value. To compensate for this estimation, we propose the buffered online RVM in the next section.

3.1.1 Buffered Online RVM

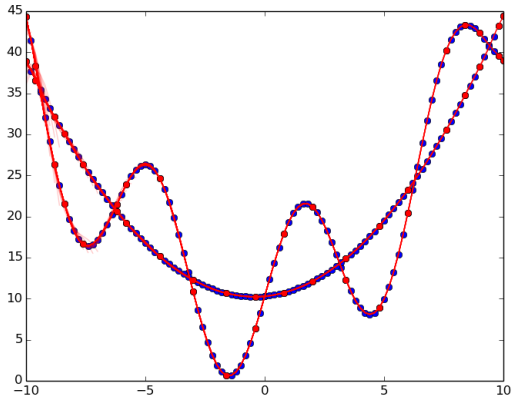
To improve the estimation accuracy, we propose a buffered online RVM algorithm that stores l non-RVs. In addition to RVs, l base candidates are maintained for better RV selection. Now, $\mathbf{\Phi}$ includes the feature vector ϕ's for the candidate samples. For buffer management, when the buffer is not full, new data can be simply added to the buffer. However, if it is full, we need to discard



(a) Online RVM



(b) Buffered online RVM



(c) Multivariate online RVM

Figure 3.1: oRVM, boRVM, and moRVM on quadratic sine curve. Blue dots are input data in left-to-right sequence. Red dots represent relevance vectors, and light red curves are run-time estimation. Final estimation after all 100 data inputs are exposed is shown in the red line. The numbers of RVs are 8, 24, and 41 for (a), (b), and (c) respectively. The accuracy gain or multivariate output requires more RVs to store.

one of the candidates. Using the kernel function, we remove the closest candidate to existing RVs. Thus, the index to delete from buffer is determined by:

$$i^{del} = \arg\min_{j} k\left(\mathbf{x_j}, \mathbf{X}^{(\mathbf{RV})}\right)$$

where kernel values are computed between the candidate $\mathbf{x_j}$ and existing RV-related sample inputs $\mathbf{X}^{(\mathbf{RV})}$. When an RV is removed, it can be stored as a candidate, determined by repeating the above procedure. With buffered online RVM (boRVM), Figure 3.1b shows the improved regression curve using $l = 25$. However, adding a buffer results in less sparse results that store unnecessary relevance vectors from the series of inputs.

3.1.2 Multivariate Online RVM

As described in the next section, we need a multivariate-output form of oRVM, moRVM, for reinforcement learning. Thayananthan, et al., [88], simply extend RVM to a multivariate setting by using a set of RVMs. Similarly, we propose a set of boRVM's for moRVM. For computational simplicity, β can be computed with average of errors:

$$\beta_{\text{shared}} = K \frac{N - \sum_i \gamma_i}{\sum_{k=1}^{K} \|\mathbf{t}_k - \mathbf{\Phi}\boldsymbol{\mu}_k\|^2}$$

where K is the number of output dimensions. Figure 3.1c shows the online fit on two dimensional output using β_{shared}. Multivariate extension of RVM also requires more number of relevance vectors to fit on multiple target outputs.

3.2 oRVM-RL

For reinforcement learning problems, we consider the approximation of the function, $Q(s, a)$. Now, the input $\mathbf{x_t} = (\mathbf{s_t}, \mathbf{a_t})$, and the Q estimates are computed with the predictive distribution from Equation (2.15) and Equation (2.16):

$$\begin{aligned} Q(\mathbf{s_t}, \mathbf{a_t}) &\sim \mathcal{N}\left(f(\mathbf{x_t}), \sigma^2(\mathbf{x_t})\right) \\ &\sim \mathcal{N}\left(\boldsymbol{\mu}_t^\top \phi(\mathbf{s_t}, \mathbf{a_t}), \beta_t^{-1} + \phi(\mathbf{s_t}, \mathbf{a_t})^\top \boldsymbol{\Sigma} \phi(\mathbf{s_t}, \mathbf{a_t})\right). \end{aligned}$$

When both state and action are used as an input of a function approximation, independence between each output y_i is hard to be guaranteed. Separating outputs for generating target y_i can improve the stability of learning [89]. With moRVM, the predictive distribution for each action is computed as:

$$\begin{aligned} Q^{(\mathbf{a})}(\mathbf{s_t}) &\sim \mathcal{N}\left(f^{(\mathbf{a})}(\mathbf{s_t}), {\sigma^{(\mathbf{a})}}^2(\mathbf{s_t})\right) \\ &\sim \mathcal{N}\left({\boldsymbol{\mu}_t^{(\mathbf{a})}}^\top \phi(\mathbf{s_t}), {\beta_t^{(\mathbf{a})}}^{-1} + \phi(\mathbf{s_t})^\top \Sigma^{(\mathbf{a})} \phi(\mathbf{s_t})\right). \end{aligned} \tag{3.1}$$

With β_{shared}, Equation (3.1) for all actions can be computed at once:

$$Q^{(\mathbf{a})}(\mathbf{s_t}) \sim \mathcal{N}\left({\boldsymbol{\mu}_t^{(\mathbf{a})}}^\top \phi(\mathbf{s_t}), {(\beta_{shared})_t}^{-1} + \phi(\mathbf{s_t})^\top \Sigma^{(\mathbf{a})} \phi(\mathbf{s_t})\right).$$

The β_{shared} will be successfully applicable to dynamic programming; however, it is hard to use in on-policy temporal difference learning, SARSA. Since SARSA explores only one action at a time, there are no other outputs in other dimension, for which the β_{shared} is a poor estimation.

To minimize the temporal difference error, we can define the oRVM target as $y = r_t + \gamma Q(\mathbf{s_{t+1}}, \mathbf{a_{t+1}})$ for SARSA or $y = r_t + \gamma \max_{\mathbf{a}} Q(\mathbf{s_{t+1}}, \mathbf{a})$ for Q-learning. In this chapter, we examine only SARSA, so Algorithm 1 is based on SARSA update. Q-learning can be implemented by simply changing y. Reinforcement learning agents develop the Q function from an initially poor estimate of future reinforcement values to a more accurate one as learning progresses, so maintaining Q estimates as targets for existing RV's can deter learning. Thus, before RVM training, updating RV targets is critical for the learning performance. The online learning algorithm in Algorithm 1 include such updates.

Algorithm 1 oRVM-RL / boRVM-RL / moRVM-RL

Initialization: empty RV for **oRVM**.
Choose discounting factor $\gamma \in (0, 1]$, $\epsilon \in (0, 1]$ and $c_\epsilon \in (0, 1]$.
for i = 0...n episodes **do**
  Select action $\mathbf{a_t}$ given state $\mathbf{s_t}$ by ϵ-greedy or softmax action selection.
  Apply $\mathbf{a_t}$ to arrive at $\mathbf{s_{t+1}}$.
  Observe sample, $(\mathbf{s_t}, \mathbf{a_t}, r_t, \mathbf{s_{t+1}})$ at time step t.
  for each observed sample and $Q_{\mathbf{w},\boldsymbol{\alpha},\beta}(s, a) = \mathbf{w}^\top \phi(s, a)$ **do**
    Set $y = r_t + \gamma Q(\mathbf{s_{t+1}}, \mathbf{a_{t+1}})$.
    Updates RV targets with current estimations $\mathbf{t^{RV}} = Q_{\mathbf{w},\boldsymbol{\alpha},\beta}(\mathbf{X^{RV}})$
    Performs **oRVM**, **boRBM**, or **moRVM** training for $Q_{\mathbf{w},\boldsymbol{\alpha},\beta}(\mathbf{s_t}, \mathbf{a_t})$.
    Decreases $\epsilon = \epsilon \times c_\epsilon$
  end for
end for

3.3 Experiments

In this section, experiments are described in which the oRVM-RL algorithms are applied to reinforcement learning problems of different complexities. For the experiments, we use Gaussian kernel to localize the state or state-action space with Gaussian pdf. Discounting factor $\gamma = 0.99$

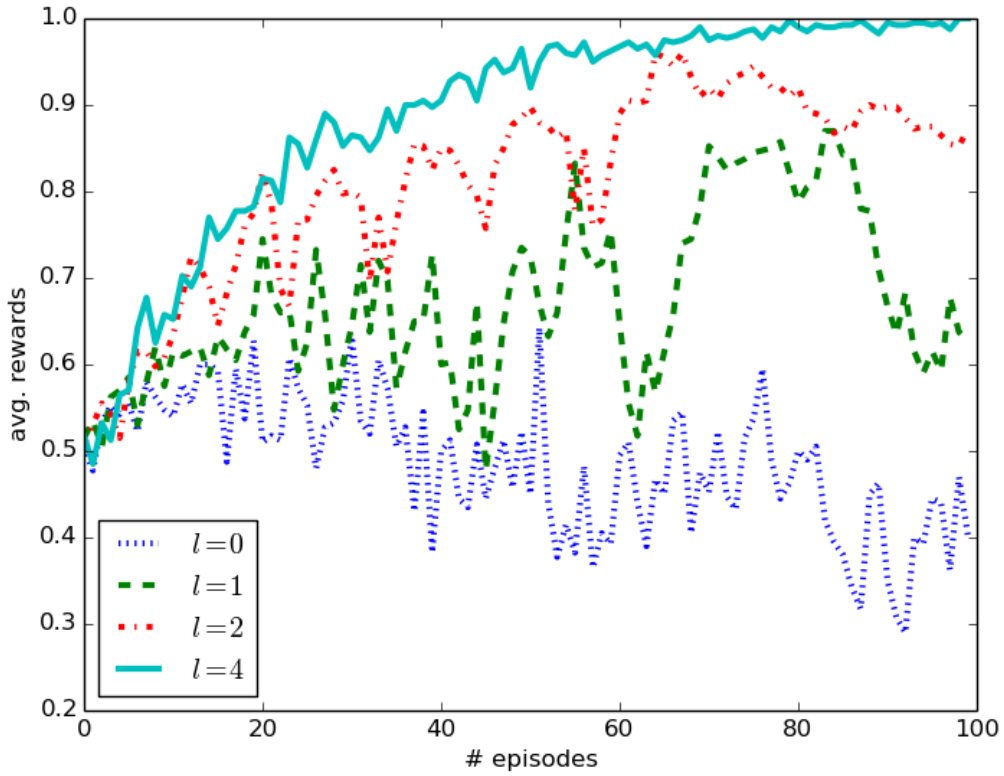


Figure 3.2: boRVM-RL in two state problem. boRVM-RL stabilizes learning curves as increasing the size of a buffer. The curves are means of 10 runs. Stability of learning curve increases as l, the size of the buffer increases.

was the best parameter value for all of the following tests. To examine the consistency of the performance, each experiment was repeated 10 times.

First, we examine how the buffer size affects the solutions of a simple toy problem with two states, $\{0, 1\}$, two actions, $\{0, 1\}$, and deterministic state transition and reward functions as follows:

$$s_{t+1} = \begin{cases} s_t & \text{if } a_t = 0 \\ 0 & \text{if } s_t = 1 \text{ and } a_t = 1 \\ 1 & \text{if } s_t = 0 \text{ and } a_t = 1 \end{cases}$$

$$R(s,a) = \begin{cases} 0 & \text{if } s = a \\ 1 & \text{otherwise} \end{cases}.$$

Figure 3.2 shows the average learning performance with varying buffer size, l. As we observed in the quadratic sine curve fit example, without a buffer or $l = 0$, oRVM function approximation is not good enough for stable learning. As the candidate buffer size increases to four, the learning curve converges to optimal policy with low variance. This shows there exists some sweet spots that both performance and efficiency can be obtained. For the following tests, we found $l = 10$ produced the best performance.

The proposed algorithm is now tested on a problem with state $s \in \{-b, -b+1, \cdots, b-1, b\}$, action $a \in \{-1, 0, 1\}$, and deterministic state transition given by

$$s_{t+1} = \begin{cases} s_t + a_t & \text{if } -b < s_t < b \\ s_t & \text{otherwise} \end{cases}$$

where b is any arbitrary integer number to represent half of the state size. The reward function is defined as

$$R(s,a) = \begin{cases} 1 & \text{if } -\frac{b}{10} < s < \frac{b}{10} \\ 0 & \text{otherwise} \end{cases}.$$

This problem is a non-episodic task, which means it needs to maintain in the goal region, $\left(-\frac{b}{10}, \frac{b}{10}\right)$, until the simulation time (500 steps) ends.

Storing a maximum of 10 candidates in a buffer ($l = 10$), boRVM successfully finds good policies for these discrete problems. However, as the state space to search grows, it faces a lack of experience. In Figure 3.3, when the size of the state space is up to 40, it learns well and converges

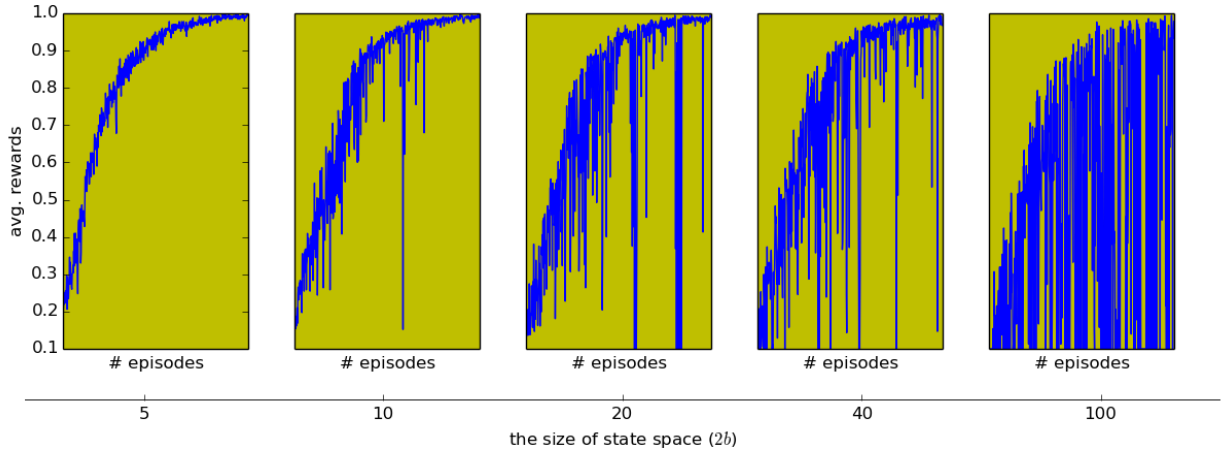


Figure 3.3: boRVM ($l = 10$) function approximation in different state size of discrete-state transition problems. Each run spends 500 episodes with ϵ-greedy by exponentially decreasing ϵ from 1 to 0.01.

to an almost optimal policy, but when it reaches 100, not all states are explored so it oscillates when it starts from unexplored region. More time is needed to explore to converge to an optimal policy with boRVM function approximation. Intuitively, it fails to learn an optimal policy in this discrete problem with more than 100 states. We find this is also true in the following continuous-state problem as well.

The continuous-state problem dynamics and reward function are as follows:

$$s = [x, v], a \in \{-1, 0, 1\}$$

$$x_{t+1} = \begin{cases} x_t + v_t \times \Delta t & \text{if } -5 < s_t < 5 \\ -5 & \text{if } s_t < -5 \\ 5 & \text{if } s_t > 5 \end{cases}$$

$$R(s, a) = \begin{cases} 1 & \text{if } -1 < s < 1 \\ 0 & \text{otherwise} \end{cases}$$

$$v_{t+1} = \begin{cases} v_t + (2a_t - f_r v_t)\Delta t & \text{if } -5 < s_t < 5 \\ 0 & \text{otherwise} \end{cases}$$

where position x is a scalar in $[-5, 5]$, v is velocity, $\Delta t = 0.1$ and friction $f_r = 0.2$. The action space is still discrete with three values, and the maximum time step is set to 500. A state-action input to a Q function approximator increases the search space with larger input dimensions. By changing a Q function approximation to generate multiple outputs for each action for given state-only inputs, we can reduce the search space for faster convergence. For this, we adopt moRVM as a function approximation. We apply moRVM-RL to the continuous-state problem.

While oRVM was not able to learn in 500 episodes because of the large search space, moRVM learns good policies within 200 episodes. Figure 3.4 shows the results with moRVM and GPTD function approximation in continuous-state problem. It shows the difference between ϵ-greedy and softmax. For ϵ-greedy, ϵ decreases exponentially from 1 to 0.1, so it has a full exploration stage in the beginning. For softmax action selection, the temperature decreases from 3 to 0.005. Softmax results shows that moRVM finds decent policy probability distributions quickly. Although it oscil-

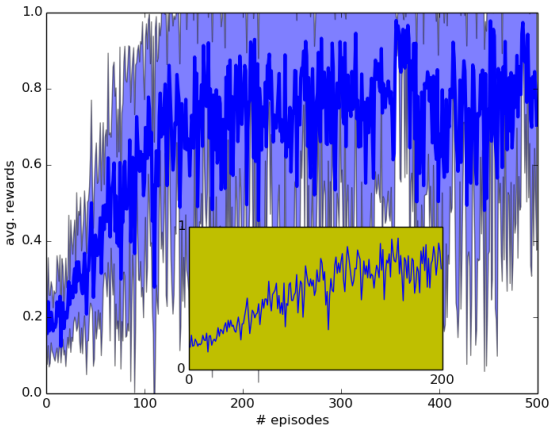
lates in 500 episodes, RVM builds good bases in the very early stage of learning. This can trigger follow-up research that uses the learned relevance vectors to stabilize the learning and to improve the performance. moRVM uses $1-7$ RV bases for each action, so in total, moRVM construct the Q value space with $8-25$ bases. Online GPTD with accuracy threshold 0.01 maintains $41-45$ bases for the poor approximation. With the small amount of samples, moRVM function approximator learns more efficiently which results in sparse solutions that can well-estimate the Q values.

With a Gaussian kernel, the learning resembles radial basis functions (RBFs) [35]. Unlike RBFs, Gaussian kernel-based RVM makes the center move by picking the relevant RVs to bases, so the number of centers either increases or decreases as learning goes on. Thus, it is comparable to Platt's resource allocation network (RAN) [90] or Barreto, et al.'s restricted gradient descent (RGD) [49]. Unlike the RGD, Gaussian RVM uses a fixed width for each radial basis function. Although the Gaussian RVM is less flexible than the RGD, by replacing the kernel function, an RVM can be more flexble. Thus, the RVM function approximator is capable to solve various problems including non-parametric problems.

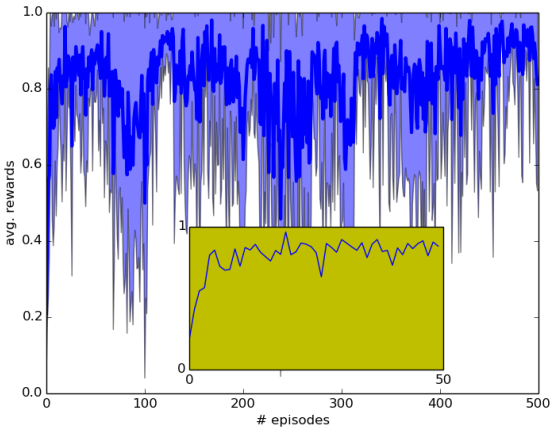
Here, note that the instability of online RVM as the problem states grow. As learning goes on, addition and deletion of relevance vectors vastly affects the Q estimation until it finds a good policy, which requires more samples to converge. By using the minibatch training with knowledge augmentation in next chapter, we achieve more stable learning.

3.4 Conclusions

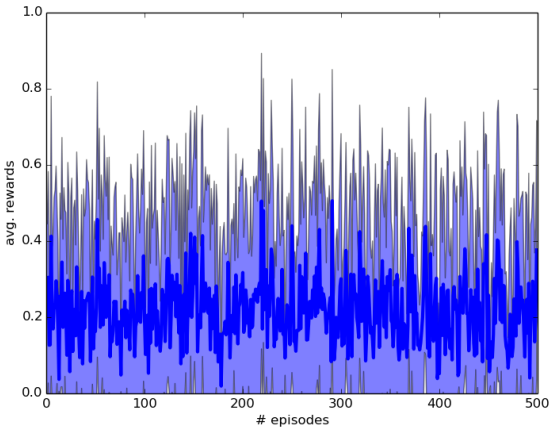
We have described modified versions of online RVMs to apply to reinforcement learning problems. Based on online RVM with incremental and decremental steps, we introduce buffered online RVM and multivariate online RVM. The buffered online RVM increases the accuracy by recon-



(a) ϵ-greedy



(b) softmax



(c) online GPTD

Figure 3.4: moRVM-RL ($l = 10$) and online GPTD in continuous-state problem. Blue line represents the average over 10 runs and lighter area shows the variance. (a) and (b) depicts moRVM-RL with different action selection method, ϵ-greedy and softmax. ϵ-greedy uses the average of 19.6 (min. 14, max. 25) RVs for the training and softmax uses the average of 17.5 (min. 8, max. 28) RVs. (c) shows the unsuccessful learning over 10 runs with online GPTD, which need more number of bases ranging between 41 and 45 bases.

sidering non-RV samples as a basis candidate. The multivariate online RVM makes it possible to evaluate multiple Q values for all the actions for reinforcement learning problems. We use all these versions to test the efficacy of online RVM function approximation.

The main contribution of this chapter is the extensions of the original RVM to fully online update versions. These modifications is not merely for reinforcement learning. With online update, improved accuracy with a buffer, and multiple outputs possibility, we provide computationally and memory-wise efficient online algorithm. Our preliminary tests discover that it is impossible to use the traditional RVM in reinforcement learning because of limitations in computation and memory. Thus, these modifications will help other applications that requires extensive computational power and memory requirements.

Without storing or augmenting significant experience, however, it suffers from instability of learning. Especially with online update, the bias from each sample is maximized enough to intervene the learning process. This leads us to develop the knowledge augmentation approach that will be introduced in next chapter.

Chapter 4

Mini-batch RVM-based Reinforcement Learning

This chapter introduces the reinforcement learning framework that systemically stores most significant memory, which is named as *snapshot memory*. The relevance vector machines first update the hyperparameters that represent the covariance of weights and target estimation. RVM training consists of two alternative stages. First, it estimates the hyperparameters α and β in Equation (2.14). Based on the current estimation of hyperparameters, it approximates the mean and variance of the weights as in Equation (2.11). The training process repeats these alternating steps until it converges. The alternating training steps discard unnecessary bases based on the hyperparameter values. This process makes the solution sparse by maintaining a small number of relevance vectors. These relevance vectors are the key information that is related to the input data samples. With some heuristics, we filter and store the relevance vectors to keep the most significant snapshot experiences. In this chapter, we examine how reinforcement learning stabilizes the performance by augmenting knowledge with successive transfer.

In Chapter 3, we examined the online RVM function approximator and observed the instability of learning when we do not store the obtained knowledge, especially when the task is complex. The simple replacement of the function approximator with an RVM still has the same problem of unstable learning. Changes in the samples state distributions during training can cause the issue, thus it requires many samples for good estimations of value functions.

For improved stability of learning, we propose a novel reinforcement learning framework that is built upon Riedmiller's fitted-Q minibatch learning [91] and incrementally accumulates snapshot memory from experiences, which uses the relevance vector machines as a function approxima-

tor. From RVM's sparse learning, the framework's gradual knowledge construction process with snapshots increases the stability and robustness of reinforcement learning by preventing possible forgetting. An additional benefit of the proposed approach is that RVM's low sampling costs improve the learning speed. This approach can be compared to directed rollout classification policy iteration [84] 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 average 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) or snapshots capture the significant mass of the agent's optimal behavior. We examine this approach with the popular benchmark problems of pole-balancing and mountain car.

4.1 Framework for Knowledge Augmentation

The reinforcement learning framework with RVM function approximator is depicted in Figure 4.1. The approach extends Fitted-Q minibatch learning [91] 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 section is that RVM regression on the Bellman target can obtain significant RV bases for good Q estimation. By transferring and shaping the knowledge from RVM training, we expect to achieve good Q function approximation.

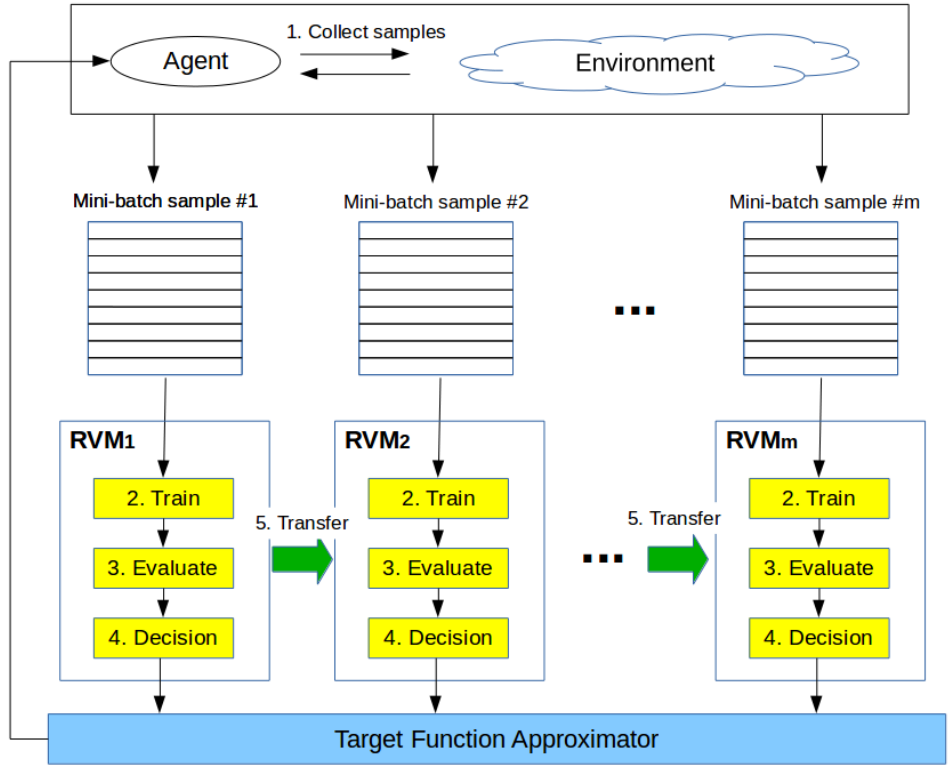


Figure 4.1: The RVM-RL learning framework

Algorithm 2 RVM-RL

Initialization: empty features for $\mathbf{RVM}_0$ and set target function approximation $\mathbf{FA} = \mathbf{RVM}_0$.
Choose discounting factor $\gamma \in (0, 1]$, $\epsilon \in (0, 1]$, learning rate c, threshold τ and $c_\epsilon \in (0, 1]$.
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 Equation (2.11) to Equation (2.14).
  Evaluate $\mathbf{RVM}_t$ with train RMSE
  if RMSE $< \tau$ **then**
    $\mathtt{RV}(\mathbf{FA}) = \mathtt{RV}(\mathbf{FA}) \cup \mathtt{RV}(\mathbf{RVM}_{t+1})$
    $\mathtt{W}(\mathbf{FA}) = (1 - c)\mathtt{W}(\mathbf{FA}) + c\mathtt{W}(\mathbf{RVM}_{t+1})$
    Store $\mathtt{RV}(\mathbf{RVM}_{t+1})$ for transfer.
  end if
  Decreases $\epsilon = \epsilon \times c_\epsilon$
end for

Figure 4.1 shows the ordered steps from collecting samples to transfer between minibatch samples, and Algorithm 2 summarizes the steps. From an interaction with a simulated or real world, an RL agent collects samples $(s_t, a_t, s_{t+1}, r_{t+1}, a_{t+1})$. For mini-batch training, the agent collects N samples and updates the RVM model via a Q-learning update (Step 2. Train). The sparse solution of RVM training embedded the RVs that can refer to the corresponding input samples. For RV transfer and feature computation, 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. Evaluate). Many different heuristics possibly exist for evaluating RVM training. In this chapter, we chose the regression 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. Decision). 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:

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

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

where $\texttt{RV}(\cdot)$ retrieves the relevance vectors of the RVM and $\texttt{W}(\cdot)$ 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 RVM weights and uses the new RVM weights only. $c = 0$ 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 4.2.

For the next sample training, instead of starting from scratch, we transfer the relevance vectors. The new, next RVM ($\mathbf{RVM}_t$) initializes the initial features with the transferred RVs. That is, the stored snapshot memories are reintroduced as new input data with union operation, and they are used as initial bases for training the new RVM. At time t, the new input samples $\mathbf{X}_t$ are augmented with the transferred samples and the transferred samples become the initial bases as follows.

$$\mathbf{X}_t = \mathbf{X}_{transfer} \cup \mathbf{X}_t$$

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

where $\texttt{RV}(\mathbf{RVM}_t) = \phi$. Here, $\mathbf{X}_{transfer}$ are the accumulated snapshot samples to be used as an initial bases for next minibatch training. 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.

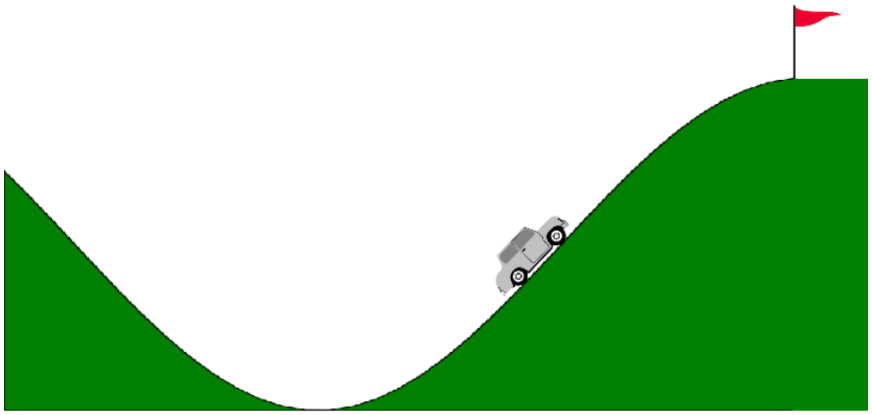


Figure 4.2: The mountain car problem

4.2 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 the 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) [61] and GPTD [64].

In our experiments, oSVR-RL fails to learn when a relatively small number of samples (200 episodes) are given. 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 [62], which generates fewer SVs than oSVR-RL.

4.2.1 Mountain Car

The mountain car (Figure 4.2) 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 [42].

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

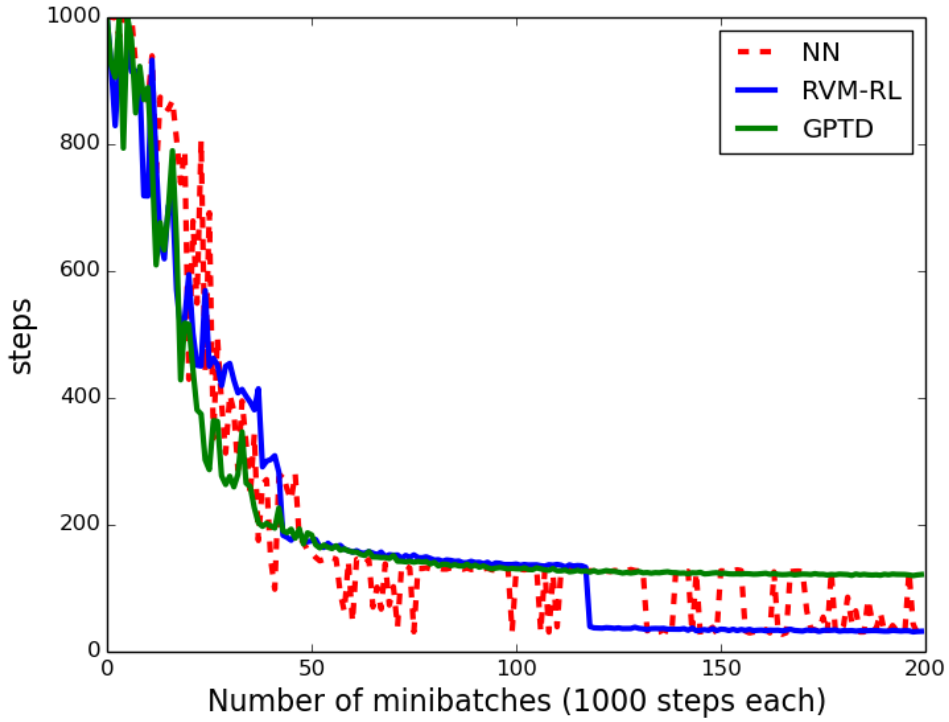


Figure 4.3: Average of steps to reach the goal in mountain car problem.

layer (one output). For the gradient update in backpropagation, we used Moller's scaled conjugate gradient update (SCG) [92] 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 $\gamma_{\mathrm{k}}^{(\mathbf{GPTD})} = 0.01$. All these parameters are chosen from our pilot tests that pick the best performing set.

To compare the performance of RVM-RL, GPTD, and neural networks, we repeated the experiment 10 times. Figure 4.3 shows the average number of steps to reach the goal with each 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

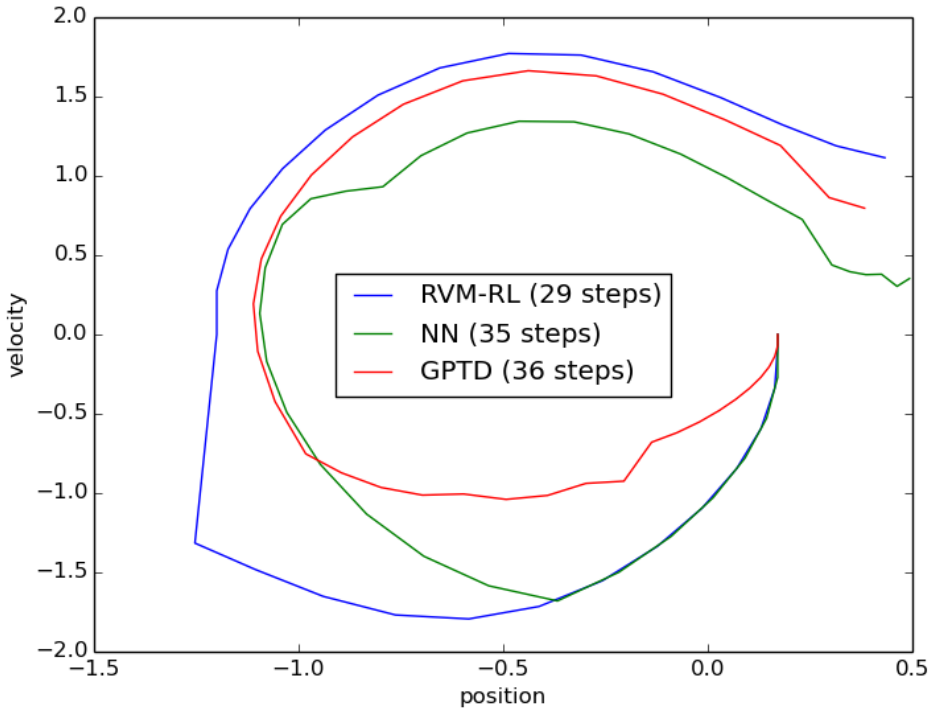


Figure 4.4: Trajectory of successful control of mountain car by trained RVM from position -0.3.

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 4.4. 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.

4.2.2 Pole Balancing

Adding a pole to a cart that swings in two dimensions, Barto, et al., [93] first introduced the benchmark pole-balancing problem (Figure 4.5). 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, its velocity is set to zero. In this instance, the pole is allowed to swing the full $360°$.

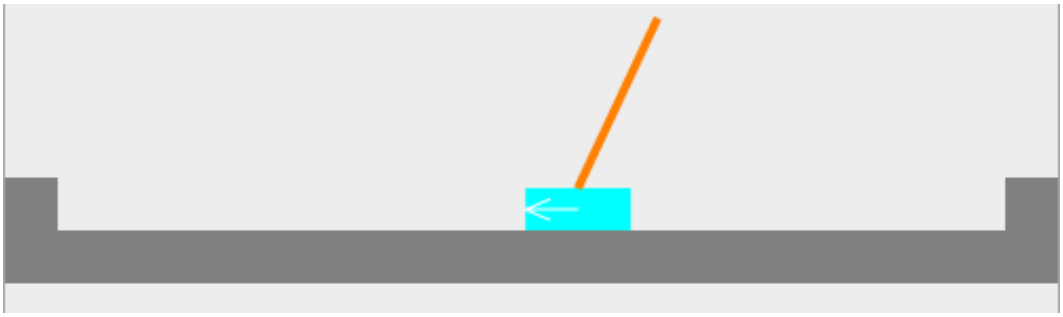


Figure 4.5: The pole balancing task

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 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.

The following parameters are selected from pilot testing. 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

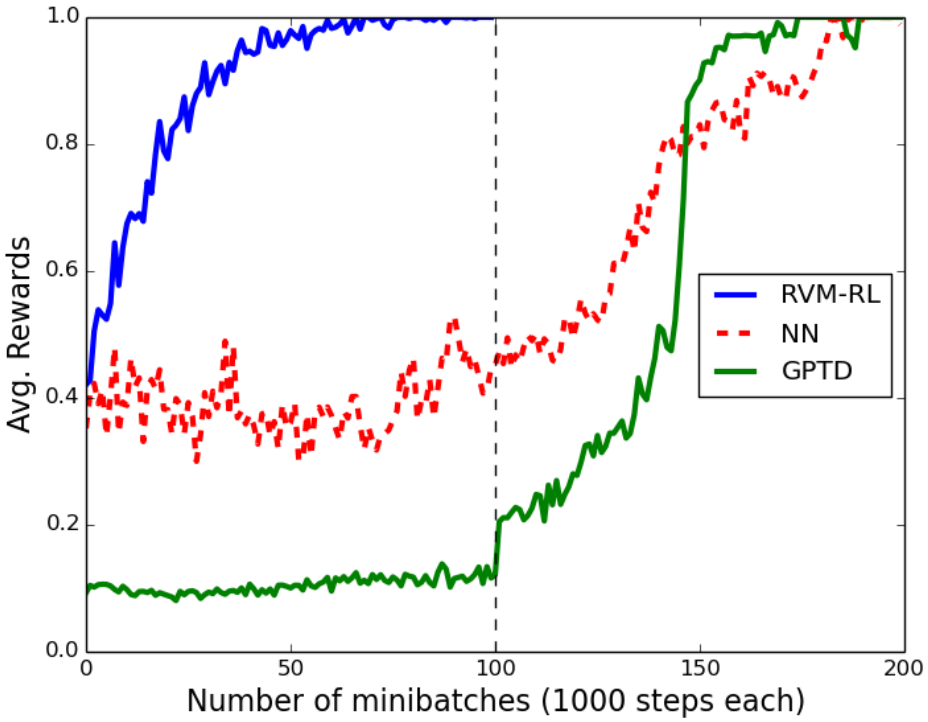


Figure 4.6: Average of rewards of an episode in pole balancing.

each) was the best structure. SCG max iteration was set to 80. GPTD required a finer accuracy threshold $v = 1 \times 10^{-5}$ 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 4.6 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 within 100 mini-batches. They need twice as many samples to learn the policy, compared to RVM-RL. Figure 4.7 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.

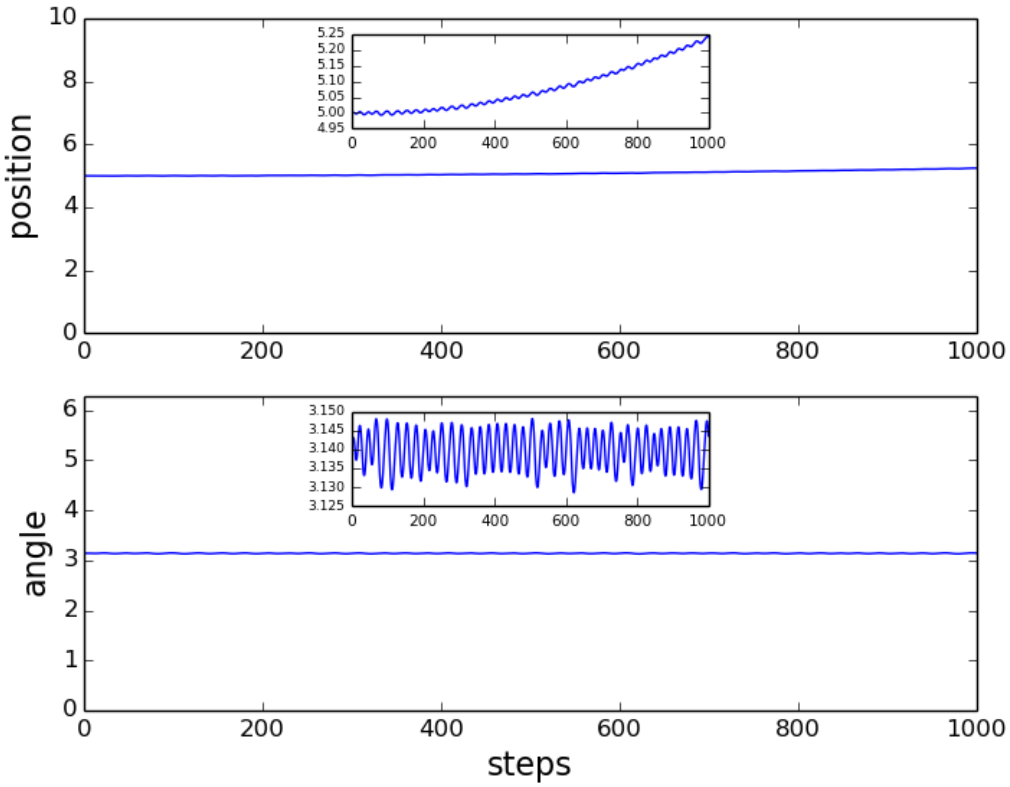


Figure 4.7: Trajectory of successful control for pole balancing. With positive samples, RVM quickly adjusts learning. The cart and pole stay center and upright. Magnifying inner plots show slight wiggle and movement to right.

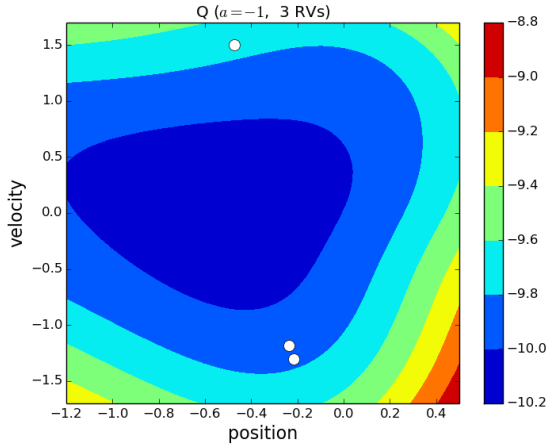
4.3 Discussion

As Tipping, et al., [39] 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 4.1 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., [62]. The mean and median of the 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. This suggests that the RVM-RL may be the more practical approach for complex RL problems such as high-dimensional or continuous-state tasks. In Figure 4.8, we examine the selected RVs for the mountain car problem

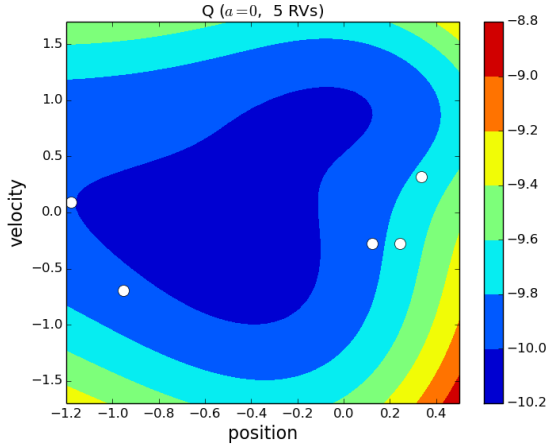
	Mean	Median	Min	Max
SVM	286	128.5	19	852
RVM	11.5	10.5	8	45

Table 4.1: The number of support vectors and relevance vectors in mountain car problem. The numbers are collected from 10 runs for each FA.

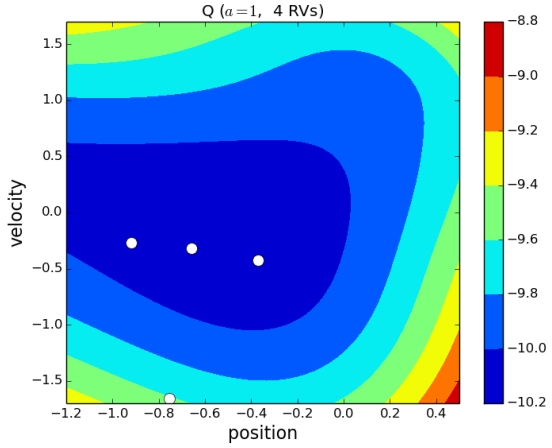
over the contour plot of Q values for each action. From the samples experienced during RL training, RVM-RL discovered the key samples that can be used as a basis and they are plotted as white dots. A total of 12 RVs were chosen as snapshot bases; 3, 5, and 4 RVs for actions of -1, 0, and $+1$, respectively. It appears that for action -1 (Figure 4.8a), two snapshots 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 snapshots for action 0 (Figure 4.8b) are placed in low velocity areas and near the top of the both hills. Snapshots for action $+1$ (Figure 4.8c) allow the selection of action 1 when the car has moved far enough up the left hill. Figure 4.9 shows the policy that



(a) left



(b) no action



(c) right

Figure 4.8: The RVM-RL trained on mountain-car problem. Relevance vectors shown as white dots over the Q contour plot.

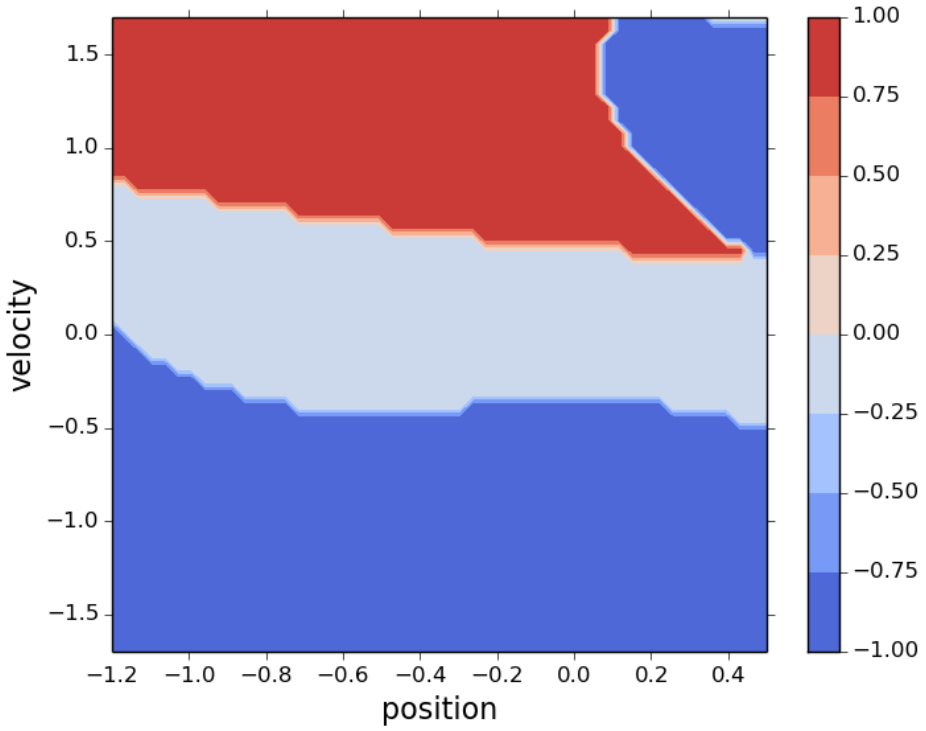


Figure 4.9: Greedy action policy of the RVM-RL trained on mountain-car problem.

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 snapshots can be considered as high level knowledge about the dynamics and goals of the task. They are abstractions of the experienced state-action pairs that are most relevant to the RL problem. The sharing of snapshot memory, 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 [13, 94] and advising [14, 15].

4.4 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, the RVM-RL maintains the learned knowledge, snapshot memory, which are considered as important experiences. By first evaluating the new RV bases 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. The RVM is fully considered and examined as a function approximator for reinforcement learning. Furthermore, with sparse solutions, it can provide computational benefits by reducing the required number of samples to explore. Most significantly, snapshot 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. Policies learned by the RVM-RL can lead to a useful analysis of the state-action space structure by examining the snapshots after training. This analysis can also be utilized for a transfer of knowledge for imitation or advising. Finally, empirically we observe the improvement of learning speed, which is caused by augmented snapshot memory.

Chapter 5

Continuous Action Spaces

In this chapter, we examine how the acquired snapshot memory from the knowledge acquisition and retention framework can be used to improve learning performance, especially in continuous action control problems. First, we introduce the improved framework in Chapter 4 without successive relevance vector transfer. This improvement is made since one RVM function approximation is sufficient with an additional weight update rule and the novel relevant experience replay. In the framework, an agent memorizes the most significant snapshots and stores them in the memory as before. With the snapshots, we suggest the novel real-valued action sampling method by using snapshot memory.

Reinforcement learning (RL) problems are often modeled as finite Markov decision processes (MDP) [42] with solutions that lie in discrete spaces. Real world problems, however, require multidimensional, even continuous state representation. Thus, the curse of dimensionality gets worse when continuous actions are needed. In the infinite state-action spaces, fine discretization can lead to successful control, but this can result in the need for a prohibitive amount of experience [95].

In practical reinforcement learning studies, function approximation estimates the state-action (Q) values to *generalize* based on limited experience to overcome the lack of experience. Parameterized Q function approximators have been successfully applied to various problems [51–55]. Some investigated transfer learning approaches to improve learning efficacy over the function approximations [13, 15, 94, 96]. Also, some studies have demonstrated convergence in practical ap-

plications involving continuous domains [57, 97, 98]. However, these approaches are still slow for practical applications and require numerous samples to learn convergent, near-optimal policies.

Perkins and Pendrith [37] showed that when the environment is partially observable, TD learning (SARSA or Q-learning) needs to follow the policy with real-valued continuous actions. Also, Lee and Anderson [57] showed that continuous actions take finer control to improve performance, and they can balance exploration and exploitation by directly searching an action over function approximation regardless of maturity. When the function approximation is not mature, searched action can be equivalent to guided exploration. When fine control is essential as in the applications such as medical surgery, the continuous space action control can be beneficial.

Several studies have shown that continuous actions allow the solution of problems that are impossible to solve with coarse discretization of action space [98–100]. The following studies considered alternatives to discretization. From a finite set of actions, some researchers obtain real-valued actions by interpolating discrete actions based on the value functions. Millan, et al., [101] sampled real-valued actions from neighbors incrementally based on the approximated value function. Hasselt, et al., [99] select actions that have the highest Q value from the interpolator. Lazaric, et al., [98] use Sequential Monte Carlo methods, which resample real continuous actions according to an importance sampling. However, these approaches require additional computations to search continuous actions.

We showed that our knowledge retention framework with relevance vector machine function approximation can be successfully adopted and the gradual snapshot augmentation improves the robustness of learning. However, the RVM-RL with augmentation suffers from a growing search space when a problem requires continuous action control. The sparse nature of RVMs captures the significant masses of an agent's experience, and it is observed that relevance vectors tend to be

located in the modes of the Q estimation surface. This happens especially when the Q estimation converges to the true Q values. Thus, we can reuse the actions in a relevance vector set since one of them is likely to be an optimal action.

In the following sections, we introduce a gradient descent approach with backpropagation and the proposed RV-sampling approach that is followed by the experiments on the simulated octopus arm control problem and analysis.

5.1 Gradient Descent Approaches for Continuous Action Search

In this section, we examine the gradient descent approaches for the problems in continuous action spaces. Both neural network and relevance vector machine function approximators are derived to search optimal, real-valued actions.

5.1.1 Action Search for Neural Network Function Approximation

The best action leads to the maximum estimated Q value on each step. At time t, with trained weights $\mathbf{v}_t$ and $\mathbf{w}_t$ (for hidden units and output units respectively), the estimated optimal action $\mathbf{a}_t^*$ is determined by this maximization step [95]:

$$\mathbf{a}_t^* = \underset{\mathbf{a}}{\operatorname{argmax}} Q_{\mathbf{v}_t,\mathbf{w}_t}(\mathbf{s}_t, \mathbf{a}).$$

For neural networks, Lee and Anderson [57] have demonstrated continuous action search with back-propagation in neural networks. They use back-propagation to calculate the derivative of Q with respect to the continuous action input. We can find the best action $\mathbf{a}^*$ that maximizes $Q(\mathbf{s}_t, \mathbf{a})$ by using gradient ascent of $Q(\mathbf{s}_t, \mathbf{a})$ with respect to $\mathbf{a}$. Since the feed-forward output is Q, the

gradient step is derived as below:

$$\frac{\partial Q(\mathbf{s}_t, \mathbf{a})}{\partial \mathbf{a}} = \mathbf{w}_t^\top \odot (1 - \mathbf{z}^2) \mathbf{v}_t^{(\mathbf{a})^\top},$$

where $\odot$ represents a component-wise multiplication, $\mathbf{v}_t^{(\mathbf{a})}$ are the weights applied to action input ($\mathbf{v} = [\mathbf{v}^{(\mathbf{s})}, \mathbf{v}^{(\mathbf{a})}]$) and the hidden unit outputs $\mathbf{z} = h(\mathbf{X}\mathbf{v})$ with the activation function $h(\cdot) = \tanh(\cdot)$. In the beginning, the network approximates the Q function poorly, and the found action is not likely to be the optimal. This will lead to further exploration in early stages. However, as the training goes on, or as ϵ decreases to exploit the learned policy, the accuracy of approximation grows, and back-propagation search will be close to the optimal. For faster search, we use Moller's SCG [92] that uses gradients with approximate second order derivatives.

5.1.2 Action Search for RVM-RL

We examine a continuous gradient descent action search over the RVM-RL framework with Gaussian kernel. When function approximation is mature, the best action leads to the maximum estimated Q value on each step. At time t, with trained weights $\mathbf{w_t}$, the estimated optimal action $\mathbf{a_t^*}$ is determined by the following step [95]:

$$\mathbf{a}_t^* = \underset{\mathbf{a}}{\operatorname{argmax}}\, Q_{\mathbf{w}_t}(\mathbf{s}_t, \mathbf{a}).$$

With Gaussian kernel $k(\mathbf{x}, \mathbf{x}') = \exp(-\gamma \|\mathbf{x} - \mathbf{x}'\|^2)$, from the mean estimate

$$Q = f(\mathbf{x}) = \sum_{n=1}^{N} w_n k(\mathbf{x}, \mathbf{x_n}) + b = \mathbf{\Phi}(\mathbf{x})\mathbf{w} + b,$$

we derive the gradient for input $\mathbf{x}$

$$\frac{\partial Q}{\partial \mathbf{x}} = -2\gamma(\mathbf{x} - \mathbf{x}')^\top(\mathbf{w} \odot \Phi(\mathbf{x})),$$

where $\odot$ is an element-wise multiplication operator and $\mathbf{x}$ is a N-by-M matrix composed by N M-dimensional relevance vectors. Since $\mathbf{x} = [\mathbf{s}^\top \mathbf{a}^\top]$, the last $\dim(\mathbf{a})$ columns are gradient for action search. Using scaled conjugate gradient (SCG) [92], action search will utilize real-valued continuous actions for finer control.

Along with the continuous action search, automatic action selection based on RVM distribution output on Q estimation can be adopted. Using Maximum Expected Improvement (MEI) [102], the improvement measure $I(Q, \theta) \in [0, 1]$ can be used as ϵ for ϵ-greedy action selection.

5.2 RVM-RL with One Function Approximation

The main goal of relevance vector sampling is minimizing the computation costs for continuous action search. By using the sparse solutions of relevance vector machines, we can easily construct a continuous action set that has the best action with a maximum Q value. To tackle the large search space problem with real-valued continuous actions, it is required to improve the RVM-RL as follows.

5.2.1 Relevant Experience Replay

The RVM-RL [103] can be slow to learn when the problem is extremely complex. When the number of input dimensions grow and the state and action spaces are continuous, the RVM-RL spends much time on maintaining multiple RVMs and updating relevant parameters. Having only

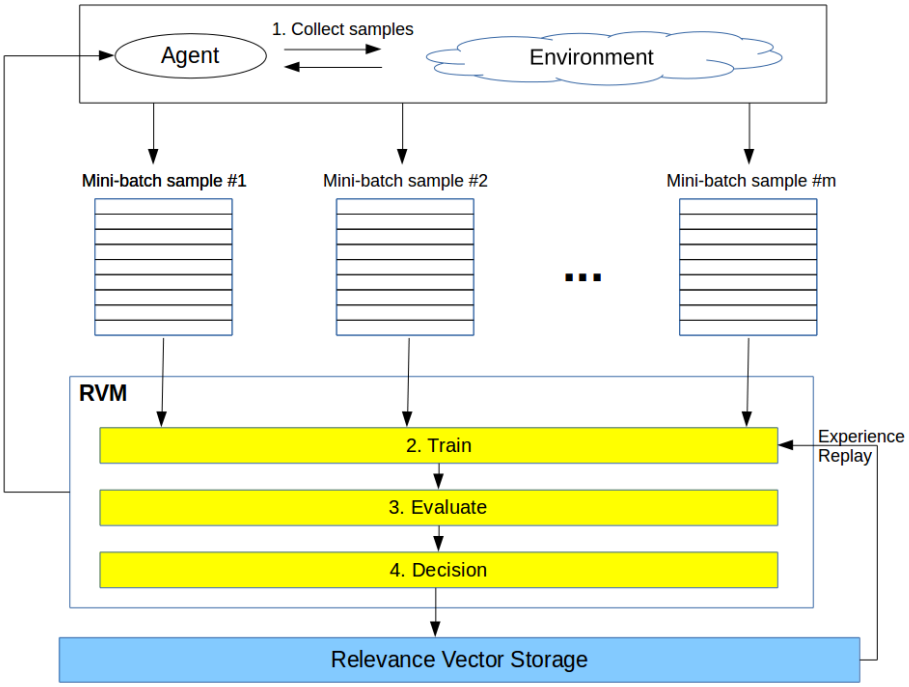


Figure 5.1: The modified RVM-RL learning framework. Instead of maintaining multiple RVMs, it shapes a single RVM with relevant experience replay.

one RVM can reduce the required computation or transfer processes. However, a single RVM can be unstable with overfitting on each batch of training data. As a solution, we suggest a relevance vector storage, snapshot memory, for experience replay–we named this as *relevant experience replay*. Thus, the modified RVM-RL, after each batch, stores relevance vectors and replays them as a training sample that is combined with the new minibatch sample. From pilot tests, it is observed that the relevant experience replay reduces the possibility of losing sparsity with augmentation. Figure 5.1 depicts the modified RVM-RL with the relevant experience replay.

5.2.2 RV Sampling for Continuous Actions

The previous section described the derivative of a radial basis function kernel for the gradient continuous action search over RVM function approximation. Focusing on the action $\mathbf{a}$, we can

rewrite the previous equation with respect to the action gradient:

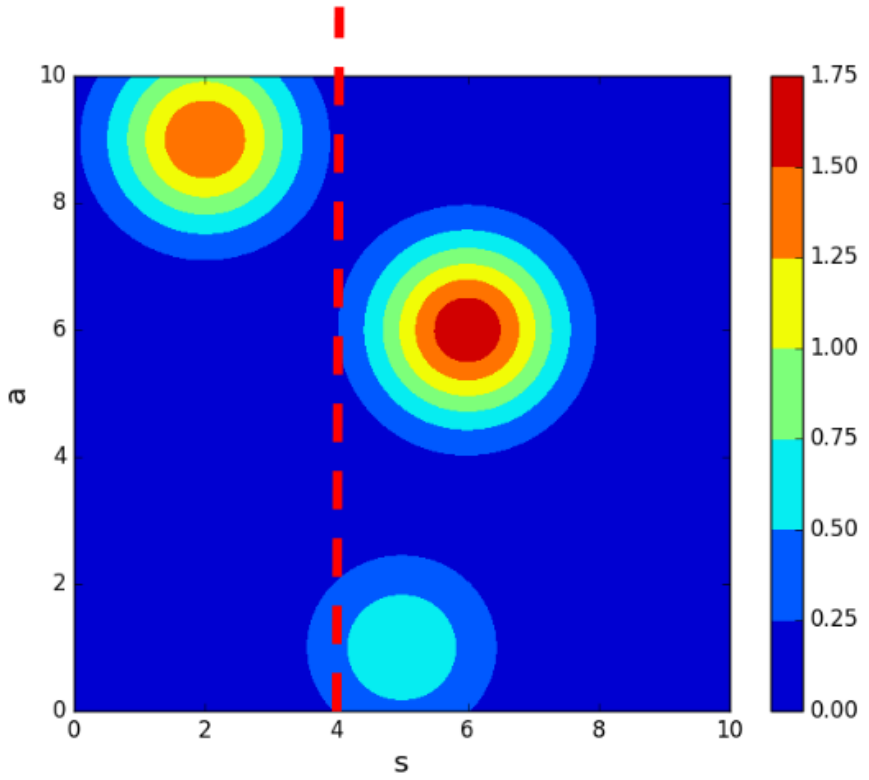
$$\frac{\partial Q(\mathbf{s_t}, \mathbf{a})}{\partial \mathbf{a}} = -2\gamma(\mathbf{w_t} \odot \boldsymbol{\phi})^\top (\mathbf{a} - \mathbf{a}'), \tag{5.1}$$

where $\mathbf{a}'$ is actions in a set of relevance vectors. These gradient search approaches, however, require additional computational costs for optimization. Furthermore, there is no guarantee for global maximum solution with the additional computation since the Q approximation can have multiple local maxima.

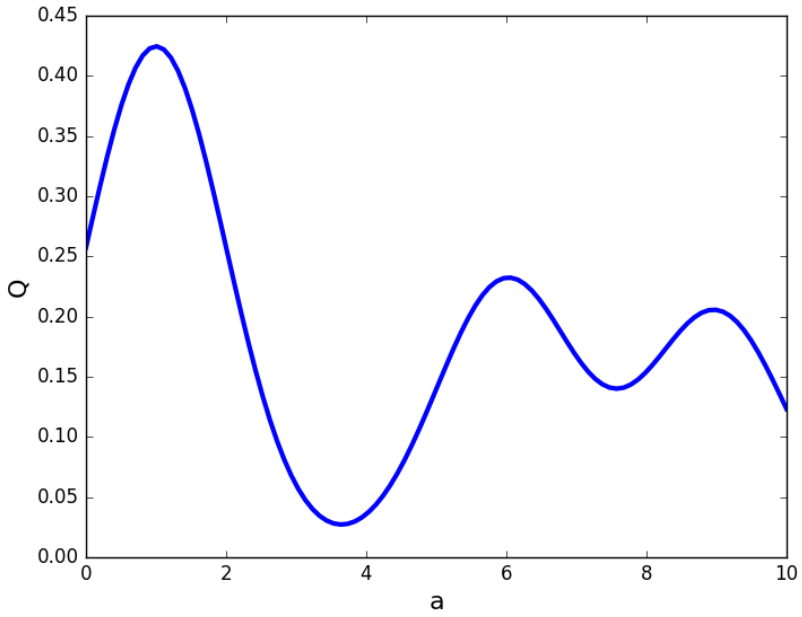
Figure 5.2 shows that with multiple RVs, depending on the selection of kernel parameters, it is likely to have more than one local maximum, which makes it difficult to find a global maximum with a gradient descent approach. Along with the multimodality, it is observed that the placements of relevance vectors are located at the modes of the Q estimation surface.

Now, we suggest a way to use an RVM as an action sampler based on the observation. With an assumption that the current Q estimation is valid, the action sets that are stored in relevance vectors lead to local maxima. For greedy action selection, the RV action with highest Q value in the action set can be chosen. From state $\mathbf{s}_t$, when current relevance vectors are stored in $\mathbf{X}^{(RVM)} = [\mathbf{s}^{(RVM)}, \mathbf{a}^{(RVM)}]$, the candidate actions are $\tilde{\mathbf{a}}^{(RVM)}$ after removing duplicate elements. From the candidates, we select action as follows:

$$\mathbf{a}_t^* = \underset{\mathbf{a} \in \tilde{\mathbf{a}}^{(RVM)}}{\operatorname{argmax}} Q_{\mathbf{w}_t}(\mathbf{s}_t, \mathbf{a}). \tag{5.2}$$



(a) Q plots



(b) Q values for all actions at the State 4

Figure 5.2: Multi-modal Q function approximation with RVM-RL. As learning converges, relevance vectors are place the modes of Q function approximation. Restricting the search of continuous actions to relevance vectors lead the proposed RV sampling.

By using the RVM-RL framework, the relevance vector sampling simplifies the continuous action set construction greatly. Additional computation is only required to remove the duplicate actions in relevance vectors that are taken in different states.

5.2.3 Kernel for Bases

For the given MDP model, RVM function approximation maps state and action inputs to the estimated Q values, ie. $Q : S \times A \rightarrow \mathbb{R}$ for state set S and action set A. Since the positive definite kernels are closed under multiplication, Engel, et al., [104] separately compute the correlation between different state values and the correlation between different action values with the product kernel.

$$k(\mathbf{s}, \mathbf{a}, \mathbf{s}', \mathbf{a}') = k_s(\mathbf{s}, \mathbf{s}')k_a(\mathbf{a}, \mathbf{a}')$$

Interestingly, the kernel approach enables us to handle both parametric and nonparametric problems. In this chapter, our target problem includes finding a general solution for different goals, so we include goal position in the input, ie. $Q : S \times A \times G \rightarrow \mathbb{R}$ for the goal position set G. For this, we extend the product kernel with an additional kernel for the goal inputs.

$$k(\mathbf{s}, \mathbf{a}, \mathbf{g}, \mathbf{s}', \mathbf{a}', \mathbf{g}') = k_s(\mathbf{s}, \mathbf{s}')k_a(\mathbf{a}, \mathbf{a}')k_g(\mathbf{g}, \mathbf{g}')$$

For the continuous action space, this product kernel can increase the search space. Thus, the proposed search of actions with relevance vectors can simplify the problem.

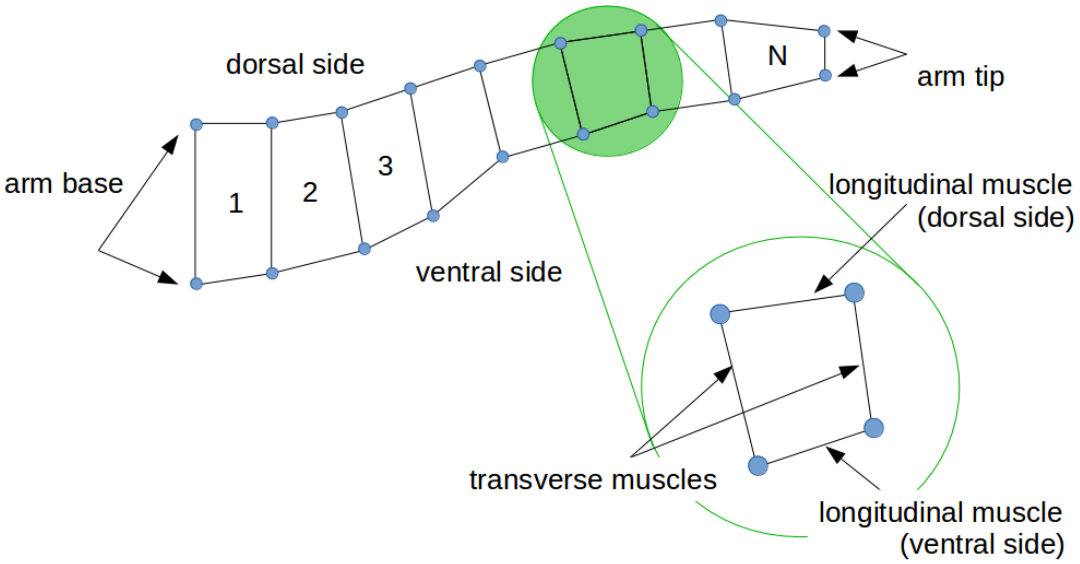
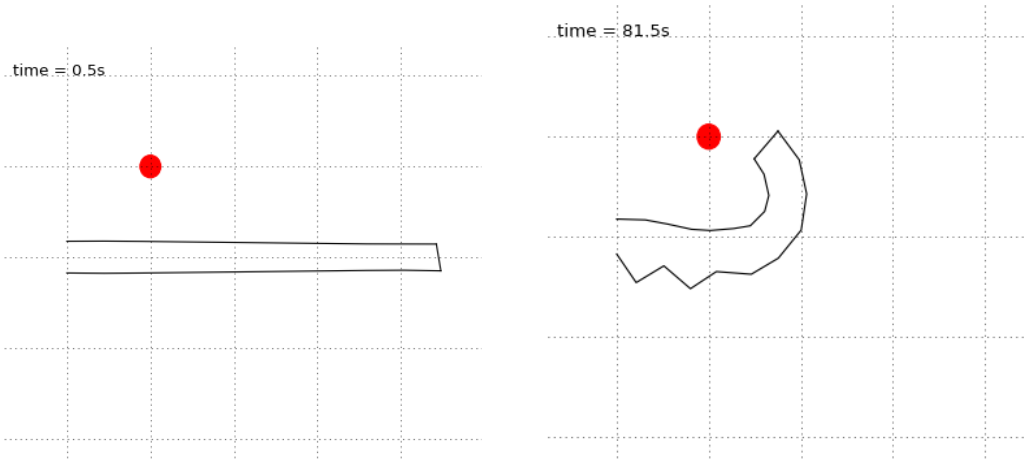


Figure 5.3: 2-dimensional octopus arm model with N compartments [2]. Spring-like muscles, 2 longitudinal (dorsal and ventral) and 2 transverse, surround a constant area C, and $2N + 2$ masses connect the muscles.



(a) Initial State **(b)** Rereaching the goal

Figure 5.4: The octopus arm control task (10 compartments)

5.3 Octopus Arm Control

In this section, we investigate the efficiency of our RV sampling approach for continuous action for reinforcement learning with the simulated octopus arm control problem. A real octopus arm is a complex organ with many degrees of freedom. Here, the Yekutieli's two-dimensional model [2] is used (Figure 5.3). The model is composed of spring muscles for each compartment. The basis of the model is that muscular hydrostats maintain a constant volume [105], so forces are transferred among the segments. Gravity, buoyancy, fluid friction, internal particle repulsion and pressure, and muscle contractions are computed each time step. For our experiments, we implemented a Python version of the octopus arm model defined in the RL-competition [106] (Figure 5.4), which simulates the simplified physics in Yekutieli, et al., [2]. The problem contains 10 compartments, and the ventral, dorsal, and transverse muscles are controlled by independent activation variables. For the experiments, we fix the base and do not consider rotation about the base. To make the problem simple, Engel, et al., [65], used 6 predefined discrete actions. Here we do not predefine actions. Without reducing the action space manually, we train the arm to learn from the full action space defined by 30-dimensional real-valued actions. For the 10-compartment example, the state space is defined by 88 continuous values: x-y coordinate positions of each joint, and their velocities.

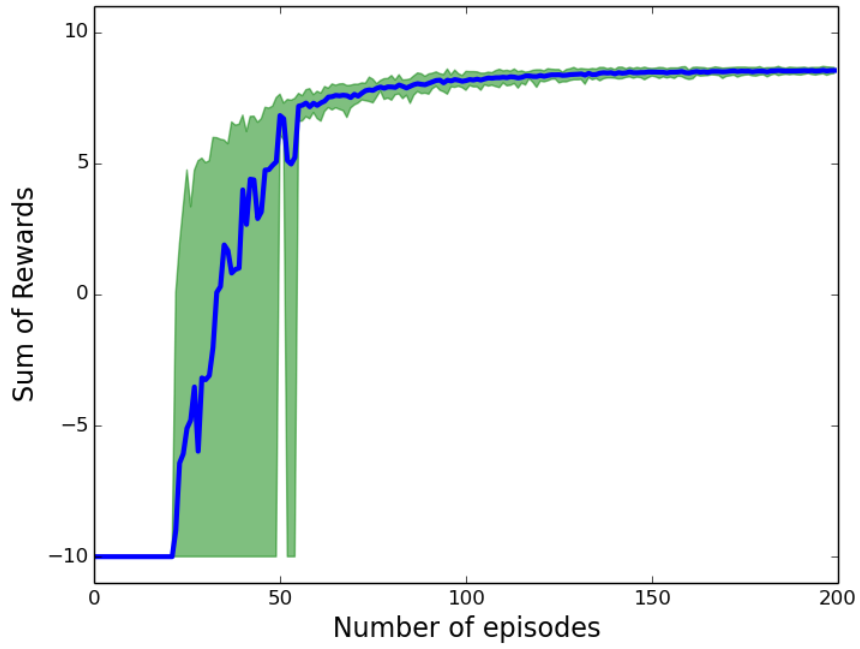
We place the goal at (4, 2) as in Figure 5.4. Initially the base of the arm is placed close to (0, 0) and the arm is straight toward the right. The target task is touching the goal with any part of the arm. On each time step, the arm receives -0.01 as a penalty, and if it reaches the goal, it gets 10 as a reward. The maximum number of steps per each episode is limited to 1000 steps. Thus, if the arm touches the goal at the last moment, the total reward will be 0. If it fails to reach the goal, the total will be -10. Positive, larger rewards are obtained when the goal is reached sooner.

The episodes are repeated 200 times with exponentially decreasing ϵ value from 1 to 0.01. Each experiment was run 10 times.

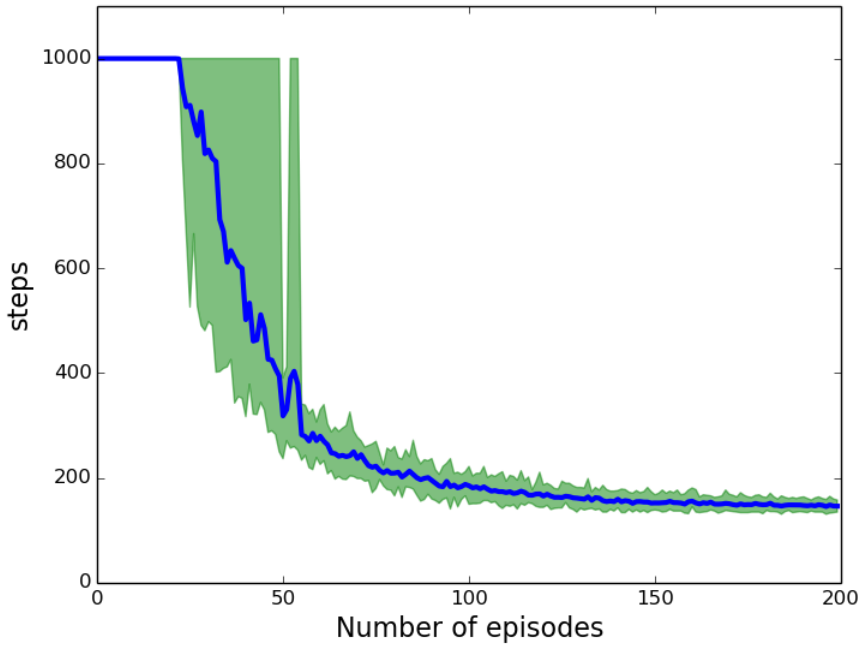
Figure 5.5 shows the average learning performance with RV-based continuous action selection in the RVM-RL. For the test, the best performing parameters from pilot tests were chosen. We use the kernel parameter $\gamma_k = 10.0$ and the learning rate $c = 0.6$. The maximum number of RVM iterations was set to 100 and tolerance threshold was set to 1×10^{-5}.

In the beginning, since the RVM-RL agent starts learning with no RV basis, the agent explores the world randomly or revisits one of a few actions in a RV set when ϵ decreases. This pure exploration stage does not have any success record, but it accumulates the RV samples with continuous actions. After about 25 episodes, as the agent applies RV actions more often, instability of learning happens because of poor Q approximation. As learning continues, after 50 episodes of experience the transient curve quickly converges in both reward and step curves. Eventually, it finds a good solution to reach the goal in 137 steps. These results are comparable to our previous continuous action search with neural networks [57] but with very low cost for search. With the similar performance, RV sampling benefits from its sparsity again. By evaluating the small number of actions, it can quickly find a greedy action while neural network back-propagation search spends more time with gradient descent updates. Furthermore, the RV sampling has room for improvement by adopting efficient exploration strategies such as importance sampling [98] and Bayesian exploration-exploitation control [63, 107].

Figure 5.6 depicts the continuous actions in the chosen relevance vectors after training. After training, 21 RVs are achieved, and they repeat the two actions, one full contraction on dorsal muscles (action2) and a more complex s-shaped muscle contraction (action1). These two actions are alternated to curl the octopus arm to reach the goal. We can observe that sparse (only two)



(a) Initial State



(b) Approaching the goal

Figure 5.5: Successful learning with RVM-based continuous actions. Blue line represents mean steps and rewards over 10 experiments, and green region shows the min and max values.

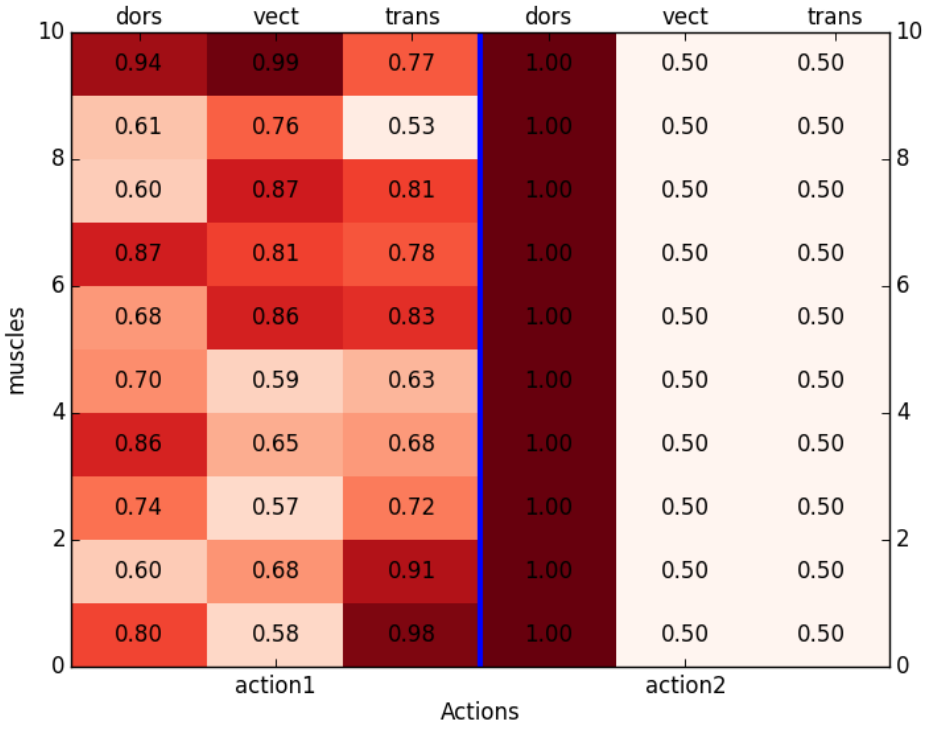


Figure 5.6: Two core continuous actions found in 21 RVs after training. As the annotated number in each box represents the contraction force. 1 means full contraction force and 0 means releasing action without any force on a muscle.

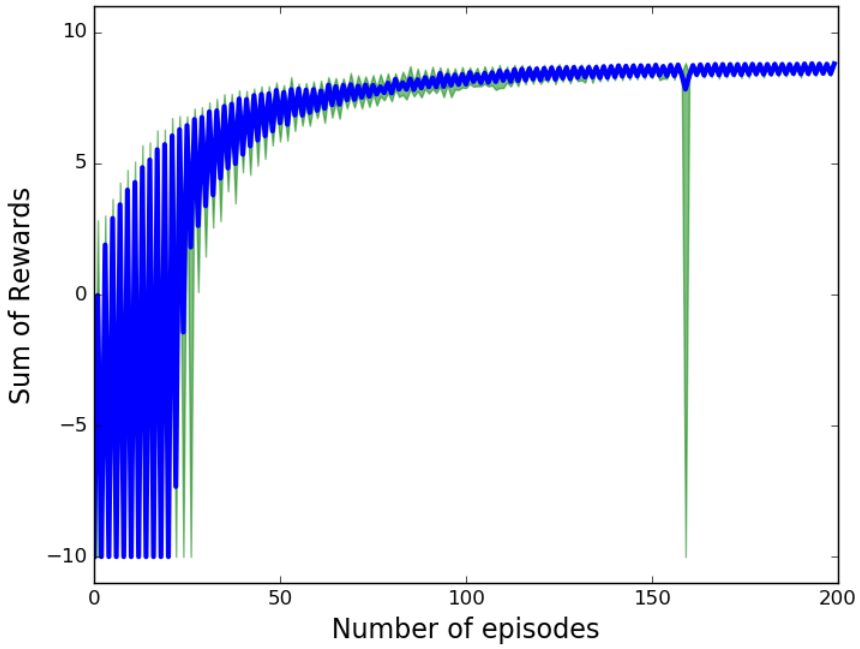


Figure 5.7: Successful transfer learning from two separate tasks to a moving goal task on each episode. In the beginning, it oscillates without noticing the changes of the goal, but as it proceeds, it discovers a good policy that can handle both goals. Blue line represents the average over rewards over 20 experiments, and green region represents the minimum and maximum values.

action options are left. This supports the argument about the benefits of RV sampling over slower neural network gradient descent action search. The solutions are sparser, so it is faster to evaluate the actions. However, we need to be careful about extremely sparse solutions that are likely to miss important samples, which can result in a poor policy. We expect this problem could be solved with better exploration strategies.

Now, we generalize the problem to have multiple goals. In this experiment, we change the goal positions to (4, 2) and (4, -2) every other episode. Alternating the goal locations can disturb what is learned, especially when it overfits to one goal. Thus, this problem ends up with oscillating performance. To simplify this problem, we adopt transfer learning. We train for two goals separately, one with (4, -2) and the other with (4, 2). After finding near-optimal policies from two tasks, an

agent learns two curling actions toward the different goal directions. After that, we transfer the learned relevance vectors to tackle the changing goal problem. In this problem, we added goal position in the state input along with state and action. Figure 5.7 shows the successful learning curve from 20 experiments. With the transferred relevance vectors, after 20 episodes of oscillation with random exploration (large ϵ), it quickly discovers good continuous actions and a near-optimal policy that reaches the goals quickly. For this experiment, we used the same parameters that we used for the previous single goal experiment.

Videos showing the arm controlling at different learning stages are available at http://www.cs.colostate.edu/~lemin/octopus.php.

5.4 Conclusion

In this chapter, we proposed a sparse Bayesian reinforcement learning algorithm with novel relevance vector sampling. The RVM-RL framework has been improved to handle problems with large search spaces by using experience replay with relevance vector samples. The proposed approaches are successfully applied to the high dimensional, continuous octopus arm control problem, even with alternating goals.

The major contributions of this chapter are the modification of the RVM-RL and low-cost, continuous action sampling. To overcome the limitations of the RVM-RL for continuous action domain problems, we replace the costly target function approximation shaping with significant (or relevant) sample storage for relevant experience replay. In addition, by reusing already discovered relevance vectors, our approach lowers the continuous action search cost to constant time. The approach proposed here to achieve the action sets from relevance vectors resembles importance sampling in sequential Monte Carlo learning [98], so its computation load is less intense than the

direct line search as in [57]. Since the RVM-RL stores the sparse relevance vectors from the learning process, there is no additional sampling cost required. In addition and most importantly, the relevant experience replay suggest a novel approach to improve learning stability by reintroducing only the necessary inputs to prevent sampling bias. Also, by using the snapshots as bases, an agent can build new knowledge upon the existing ones so that it makes learning more efficient and robust.

The following steps will be taken to further improve the approach described here. Since the proposed approach assumes the eventual convergence of Q approximation, it can guarantee fast or stable learning only when it reaches the near-optimal point. Thus, we can combine Equation (5.1) and Equation (5.2) by using the RV actions as starting position for gradient search for improved learning performance in early stage. Instead of random exploration, however, if we sample actions efficiently based on the current RVs and Q estimation, the RVM-RL is expected to place RVs on the peaks of new Q estimation and the correct action with the highest Q value will be selected with greedy strategy.

Chapter 6

Practice in Reinforcement Learning

This chapter introduces our *practice* approach [108] to improve the learning performance in target tasks. We define practice as a kind of transfer learning but with a no-reinforcement or no-goal related source task. The idea of practice comes from human development. Occasionally, babies wave their hands and arms without any purpose. From successive incidents, they realize the effects of the random movements, and they use this experience to better perform in a specific task such as grasping a toy. Similarly, athletics spend a lot of time on practice so that they can out-perform in real games.

Natural intelligence is developed from acquired experience and adaptation to new environments, resulting in the learning of new knowledge. The repetition of this cycle constructs new knowledge and sharpens acumen. Machine learning studies have tried to mimic this behavior and have shown successful transfer of knowledge in various domains. The benefits from transfer learning in supervised learning have been shown in textual data classification [26], natural language processing [109], image classification [110], WIFI localization [27], spam filtering and intrusion detection [28].

A reinforcement learning (RL) agent needs a fair amount of experience to find a near-optimal policy. Transfer learning has been investigated as a means to reduce the amount of experience required. Transfer learning can boost reinforcement learning (RL) in many tasks, such as video games [59], and robot soccer [111]. In reinforcement learning, since the samples for training contain an agent's behavior, schemes in addition to simple parameter value transfer can be adopted, such as smooth landing imitation [13, 94] and advising [14, 15]. However, these approaches are

limited to knowledge exchange between two reinforcement learning tasks. When the target task or the goal is not unveiled, it is impossible to collect knowledge for transfer. Transfer learning that requires another similar reinforcement learning task as a transfer source can also be costly in the amount of experience required.

In this chapter, we examine the possible “practice" approach that transfers knowledge from a non-RL task to a target RL task to avoid the expensive data sampling. By using snapshot memory, we analyze how practice captures the distributions of state and action spaces in an environment. To better understand the benefits of practice, we focus on the obtained snapshots from the knowledge retention framework. The analytical power of the framework through a Bayesian interpretation can help us observe how practiced knowledge assists successive learning. Because of the sparse nature of RVMs that capture the significant mass of an agent's experience, an RVM-RL helps us understand how state transition prediction in Anderson, et al., [96] improves the performance of reinforcement learning. For this, we use a modified version of the RVM-RL as a learning framework to collect the experiences in practice to explain what knowledge an agent obtains from practice and how it can be applied as a feature space for reinforcement learning. We demonstrate how practiced knowledge contributes to reinforcement learning by fixing the RV bases with the suggested framework.

This chapter introduces a novel learning approach that acquires important snapshot samples from practice and then applies them to a target RL task without changing learned bases. Results show an improved learning efficiency through practice in classical benchmark problems and limitations in OpenAI Gym problems.

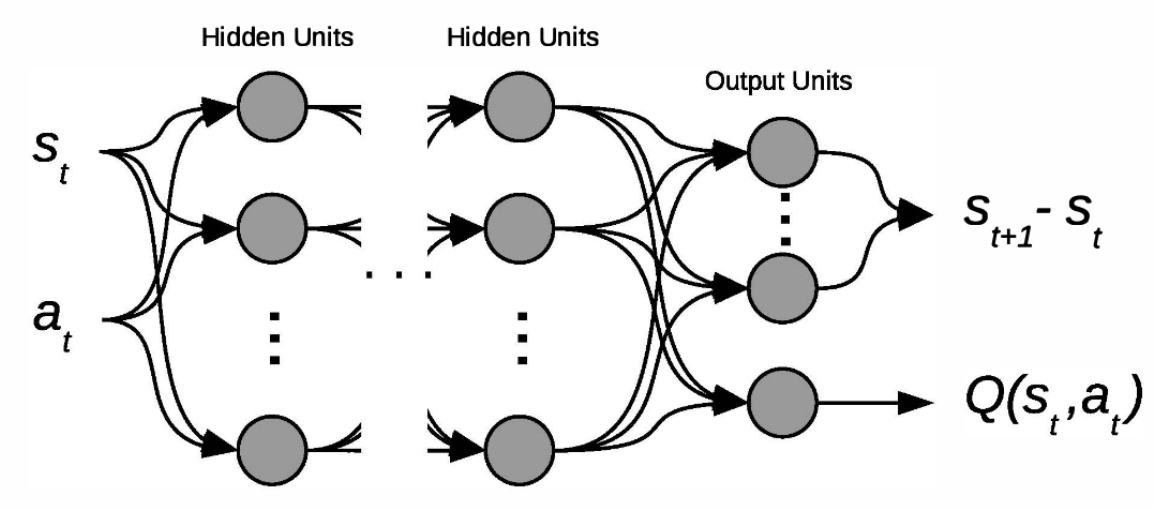


Figure 6.1: Neural networks for state change prediction and Q estimation. First, neural networks are trained to predict state changes. Then the role of the final layer is changed from predicting state change to predicting future reinforcement as the value of a Q function.

6.1 Pretraining Deep Networks for Reinforcement Learning

Anderson, et al., [96], demonstrated how learning the dynamics of an environment can facilitate the learning of a Q function in multilayered neural networks. For complex problems in practice, pretraining neural networks can greatly reduce the number of interactions with an environment that are required to achieve good performance.

Figure 6.1 explains how network weights are pretrained and transferred for learning a Q function. To *pretrain* the neural networks, state transition samples of state s_t, action a_t, and next state s_{t+1} are collected. From the samples, s_t and a_t are used as input and $s_{t+1} - s_t$ forms the target output to fit. The scaled conjugate gradient (SCG) algorithm [92] was used to train the network to minimize the mean squared error in the outputs. For this pretraining stage, any reinforcement learning related information such as rewards, goals or objectives is not provided.

After pretraining the networks, a reinforcement learning agent collects mini-batches of s_t, a_t, reward r_{t+1}, s_{t+1}, and a_{t+1} samples. With these mini-batches, the pretrained neural networks are

further trained to estimate Q values by minimizing the Bellman error with SARSA [35] update. Again, the SCG algorithm is applied. To lessen the chance of overfitting with each mini-batch, the number of iterations of the SCG algorithm is limited to a small number.

6.2 RVM Practice for Reinforcement Learning

Here, we replace the term *pretraining* with the more general term *practice*. We focus on discovering knowledge that summarizes the dynamics of an environment. We adopt the relevance vector machines to discover such knowledge. We develop fixed-basis RVMs to examine the obtained knowledge efficacy for efficient reinforcement learning. For this, we first summarize the slight modifications to the RVM-RL framework with a fixed-basis and describe how to incorporate practice in the learning of an agent and how to apply this to actual problems.

6.2.1 Fixed-basis RVM (fRVM)

For efficient learning, the fast marginal likelihood maximization algorithm for RVMs [40] adds or removes bases, as defined by the set of RVs, to find the best fit that maximizes the log-likelihood. However, when we already know the best bases or alternative ones, it is not necessary to go through the RV addition or removal process. Assuming that $\mathbf{X}^{\mathbf{RVM}}$ are relevance vectors, we can define the feature vector $\boldsymbol{\phi}$ with kernel $k(\cdot)$ as follows:

$$\boldsymbol{\phi}(\mathbf{x}) = k\left(\mathbf{x}, \mathbf{X}^{\mathbf{RVM}}\right).$$

Now, we can alternatively compute the following weight mean and covariance and prior distributions without modifying RV bases. First, compute the mean and covariance of the weights. Here,

$\mathbf{\Phi}$, representing $\boldsymbol{\phi}(\mathbf{x})$, is the similarity of $\mathbf{x}$ to the preset relevance vectors:

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

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

From the weight estimation, the hyper-parameters are computed:

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

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

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

Here, $\boldsymbol{\alpha}$ and β represent the hyper-parameters for the prior distribution of weights and target. γ_i is interpreted as a measure of how well-determined the weight w_i is by the data. $\mathbf{\Phi}$ can be defined as a matrix composed of training vectors transformed by the basis function. $\mathbf{t}$ is the Q-learning target.

Figure 6.2 shows the application of the fRVM to a classic quadratic sine curve fit problem from Tipping, et al., [39]. Manually setting the seven fixed RVs, we can obtain good predictions with the radial basis function (RBF) kernel parameter $\gamma_{\mathrm{k}} = 0.06$ and tolerance 1×10^{-3}.

6.2.2 RVM-based Practice for RL

In the practice stage, a regular RVM predicts state changes and discovers the relevance vectors for reinforcement learning tasks. Our hypothesis is that learning the dynamics of the world can result in the discovery of knowledge that can be applicable to reinforcement learning tasks. This was empirically examined with deep networks [96]. We expect this will be the same with an

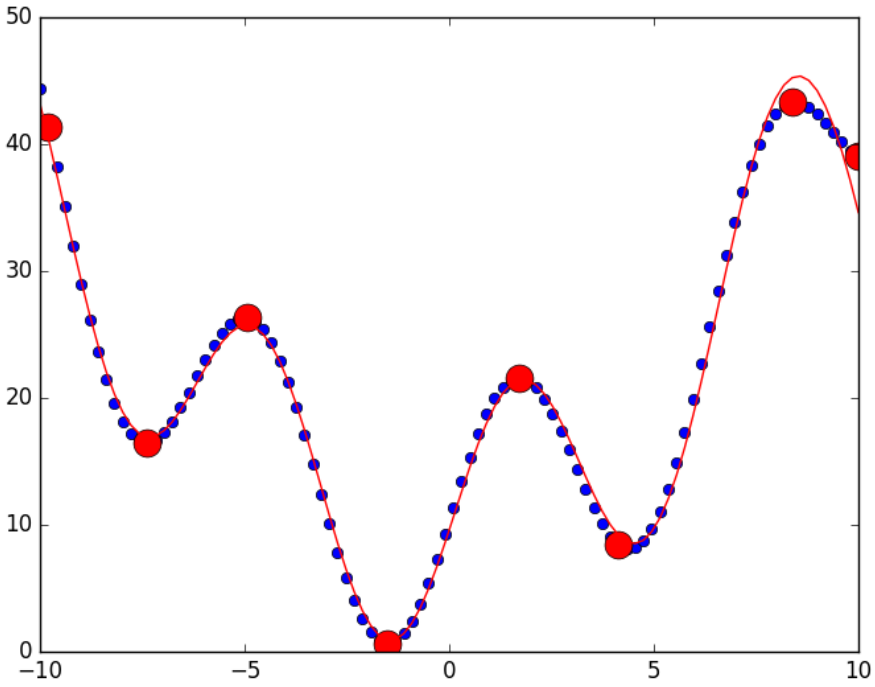


Figure 6.2: fRVM with preset RVs (red dots). Blue dots represent the training samples and red line shows the prediction curve fit.

RVM function approximation. RVMs relate the learned experience to input samples, so we expect to interpret what was learned in the practice stage more easily than from the pretrained neural network structures.

To improve the learning speed and examine the practice contribution, we adopt fRVM-based reinforcement learning that do not change the RV bases. In the previous section, we observed fRVMs can fit well when the bases are known. From the randomly explored or collected samples, we can train an RVM to predict the next state or the difference between current and next state. The learned RVM produces relevance vectors that capture key dynamics of an environment. Assuming these RVs are known bases, we can build a fixed-basis RVM for reinforcement learning (fRVM-RL). Now the fRVM-RL adjusts weights, so it estimates Q vales based on similarity kernel features to the learned RVs.

In a reinforcement learning framework, we use fRVM as a function approximator that estimates Q-values. Unlike the RVM-RL, it does not need to maintain multiple RVMs and does not need to transfer RVs in each step. It simply updates weights following the RVM update rules. Thus, after practice results in good bases for reinforcement learning, fRVM-RL can learn a policy very efficiently. Algorithm 3 describes the learning algorithm for practice and fRVM-based reinforcement learning.

How to collect practice samples to improve the target learning performance is a significant issue. Various practice approaches can be investigated but in this chapter, we focus on previously examined state-transition dynamics samples and random sample collection. First, we can use a simulation of a dynamic system to generate samples. In this case, we can have two different options for the regression target, the next state or the difference between the next state and current state. When using dynamics simulation, the next state is close to the previous state and the samples

Algorithm 3 RVM-Practice and fRVM-RL

Collect L samples of tuple (s, a, s') using environment dynamics.
Set regression target $z = s'$ or the state changes $z = s' - s$.
Train RVM and discover basis $\mathbf{RV}_{practice}$ and weights $\mathbf{w}_{practice}$.
Initialization: the basis sample $\mathbf{X}^{\mathbf{RVM}}$ and weights $\mathbf{w}$ of **fRVM** with practiced $\mathbf{RV}_{practice}$ and $\mathbf{w}_{practice}$.
Choose discounting factor $\gamma \in (0, 1]$, $\epsilon \in (0, 1]$, learning rate c and $c_\epsilon \in (0, 1]$.
for each mini-batch **do**
  Select action $\mathbf{a_t}$ given state $\mathbf{s_t}$ by ϵ-greedy action selection. Apply $\mathbf{a_t}$ to arrive at $\mathbf{s_{t+1}}$.
  Observe N samples, $(\mathbf{s_t}, \mathbf{a_t}, r_{t+1}, \mathbf{s_{t+1}})$ at consecutive time steps t.
  Set target $y = r_{t+1} + \gamma \max_a Q_{\mathbf{w},\boldsymbol{\alpha},\beta}(s_{t+1}, a)$
  Train fRVM
  $\mathbf{w} = (1 - c)\mathbf{w}_t + c\mathbf{w}_{t+1}$
  Decreases $\epsilon = \epsilon \times c_\epsilon$
end for

are more likely dependent on each other, which can require more samples for the practice stage. With the state difference target, we can reduce the required number of samples and increase the independence of samples. Random sampling can be used when there is no simulation model for dynamic sampling. Without knowing the dynamics of the world, it randomly generates samples with a certain distribution. Also, it can reduce the possible biased sampling from simulated dynamics. However, it is difficult to understand what it learned from this randomly sampled practice by disconnecting correlation between s_t and s_{t+1} since s_{t+1} is not dependent on s_t and a_t. Further strategies should be investigated for better practice models and efficient reinforcement learning. We will discuss more about this issue in following sections.

6.3 Experiments

We investigate the efficacy of RV bases that are built by RVM-based practice with two classic reinforcement learning benchmark tasks. The first task is the mountain-car problem consisting of an under-powered car that must climb up a hill is tested. For the second task, we test the pole

balancing problem in which a pole must be balanced by pushing on the cart to which it is attached. We compare RVM-based practice methods on these tasks with the following RL algorithms. Neural fitted-Q learning (NN) is a well-known successful learning framework with a similar mini-batch learning structure. The Gaussian process temporal difference (GPTD) algorithm has a Bayesian structure similar to RVMs. Finally, the RVM-RL is included to allow the direct examination of the efficiency of practice.

To examine the cases with large search space for practice, we apply the proposed fixed RVM-RL to two Box2D problems in OpenAI Gym (http://gym.openai.com) [112]. The first task is the lunar lander that controls a spaceship that fires main and side engines to smoothly reach on the landing pad. The second task, car racing, is to learn how to control a car from the top-down racing track image pixels. Both problems require large amounts of exploration to obtain a good policy. Although solutions are found for lunar lander, yet no one found solutions for the car racing task. With these two examples, we discuss the limitations of fixed bases learning approach.

6.3.1 Mountain Car

We are revisiting the mountain car problem in Chapter 4. that controls an under-powered car that cannot climb directly up the right hill, but must first be pushed up the left hill to gain enough momentum to reach the goal at the top of the right hill. Available actions are push forward (+1), push backward (-1), and no acceleration (+0).

For the practice stage, we randomly generate samples $p = (x_t, \dot{x}_t)$ in the range of $x_t \in [-1.2, 0.5]$ and $\dot{x}_t \in [-2.0, 2.0]$. 10 repetitions of the generation of 1000 practice samples result in different numbers of RVs, ranging from 12 to 18. During this practice stage, the RBF kernel parameter γ_{k} is set to 1.0, and the maximum number of iterations is limited to 100.

The reinforcement learning discount factor γ is set to 0.9. To test with a small number of exploration actions, we exponentially decrease ϵ from 1 to 0.1 with a factor 0.9885. With the decreasing ϵ, actions are chosen by ϵ-greedy algorithm. This is repeated 1000 times. The mini-batch of 1000 steps is used to update the fRVM weights for Q function estimation. This is repeated for 200 mini-batches. For fRVM-RL training, we preset the fRVM with the $RV_{practice}$ achieved from the practice stage. The RBF kernel parameter γ_{k} is not changed from 1.0 to accommodate the achieved knowledge. The learning rate $c = 0.2$ was best-performing in our pilot tests and used for the mountain car task tests. The fRVM maximum number of iterations is set to 10. Neural networks with two hidden layers, each of 20 units, are chosen for comparison. Moller's scaled conjugate gradient optimization algorithm [92] was limited to 20 iterations to avoid overfitting. For GPTD, the RBF kernel parameter $\gamma_{\mathrm{k}} = 0.01$, the accuracy threshold $v = 0.1$, the convergence threshold $\eta = 1 \times 10^{-4}$, and initial standard deviation $\sigma_0 = 0.1$ result in the best performance.

fRVM updates only the weights in the middle of training, and as we can see in Figure 6.3, fRVM-RL quickly finds the best policy. Comparing the convergence point, fRVM-RL with practice converges to good performance with 100 fewer mini-batches than the previously best performing algorithm, RVM-RL. Since we start $\epsilon = 1$, we observe that the starting points of the curves are not different, and the transferred weights are not utilized for jumpstart test. However, by not adding or removing bases in the middle of training, the training is simplified with linear weight updates that reduces learning time considerably. Most of all, this exemplifies the RV bases obtained from practice well capture the distributions of main factors for correct Q estimation.

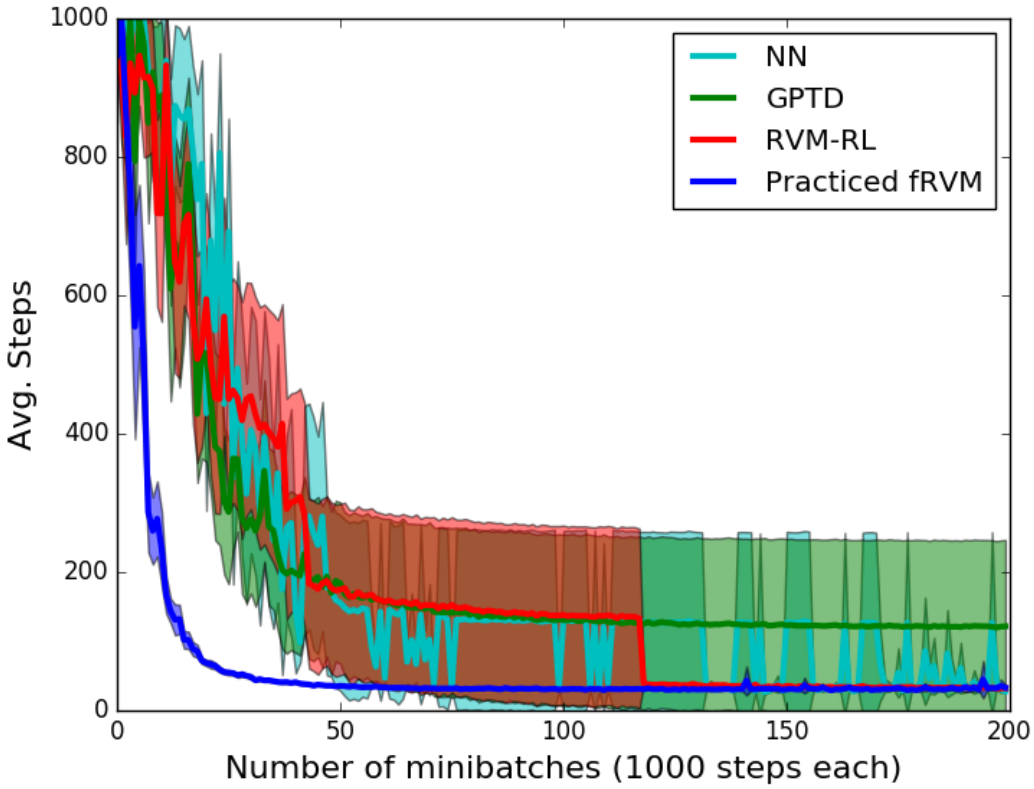


Figure 6.3: Average of steps to reach the goal in mountain car problem. The average curve line is computed from 10 experiments. Practice reduces the required number of samples greatly. 1000 samples (the number of steps in one minibatch) are used for practice. The shaded areas represent 95 % confidence interval.

.

6.3.2 Pole Balancing

To examine the effects of practice, we test the pole balancing problem in Chapter 4. The objective is to apply forces to a cart in a given track and to keep the pole from falling over. Three actions to control the cart are defined: move left, move right, and apply zero force. The goal is to maintain the pole upright as long as possible.

For the practice stage in the pole balancing environment, we use dynamic simulation and train an RVM to predict state changes. 100 practice samples are good enough to produce the necessary bases for fRVM-RL training. In 30 practice stages, RVMs produced from 7 to 20 of RVs. For

practice, the RBF kernel parameter $\gamma_{\mathrm{k}} = 20$, and the maximum number of iterations is limited to 100.

All tests share the discount factor $\gamma = 0.99$ and decrease ϵ from 1.0 to 0.1 exponentially. For training, 100 mini-batches with 1000 samples each are collected. fRVM-RL uses the RBF kernel parameter $\gamma_{\mathrm{k}} = 20$ and the learning rate $c = 0.2$. The best parameters for the neural networks and GPTD were found from pilot tests. Neural networks with 10 units in each of two hidden layers was the best performing structure with the maximum number of iterations for SCG was set to 80. GPTD performed best when $v = 1 \times 10^{-5}$ and $\eta = 0.1$ Initial standard deviation $\sigma_0 = 10$ and the RBF kernel parameter γ_{k} is set to 1×10^{-5}. As we can see in following results, even with the best parameters, we cannot make the two function approximators work in 100 minibatches. They required twice as many samples to find an optimal policy.

Similar to the mountain car task, we can observe that practice greatly contribute establishing good basis for reinforcement learning function approximation. The fRVM-RL quickly reaches the optimal point and steadily converges. Comparing to the RVM-RL, fRVM-RL can save more than 40 mini-batches that contain more than 40,000 samples (Figure 6.4). For the pole balancing task, we found that adding 100 samples for practice results in learning good performance quickly, reducing the number of samples needed to approximate Q function correctly by 40,000 samples.

6.3.3 Racing Car and Lunar Lander

To examine the efficacy of practice in complex problems, we applied the fRVM-RL to Box2D problems in OpenAI Gym such as CarRacing-v0 and LunarLander-v2. In the LunarLander-v2, an agent chooses one of four actions: nothing, fire left orientation engine, fire main engine, and fire right orientation engine. The goal is landing the craft on the landing zone smoothly. Thus, the

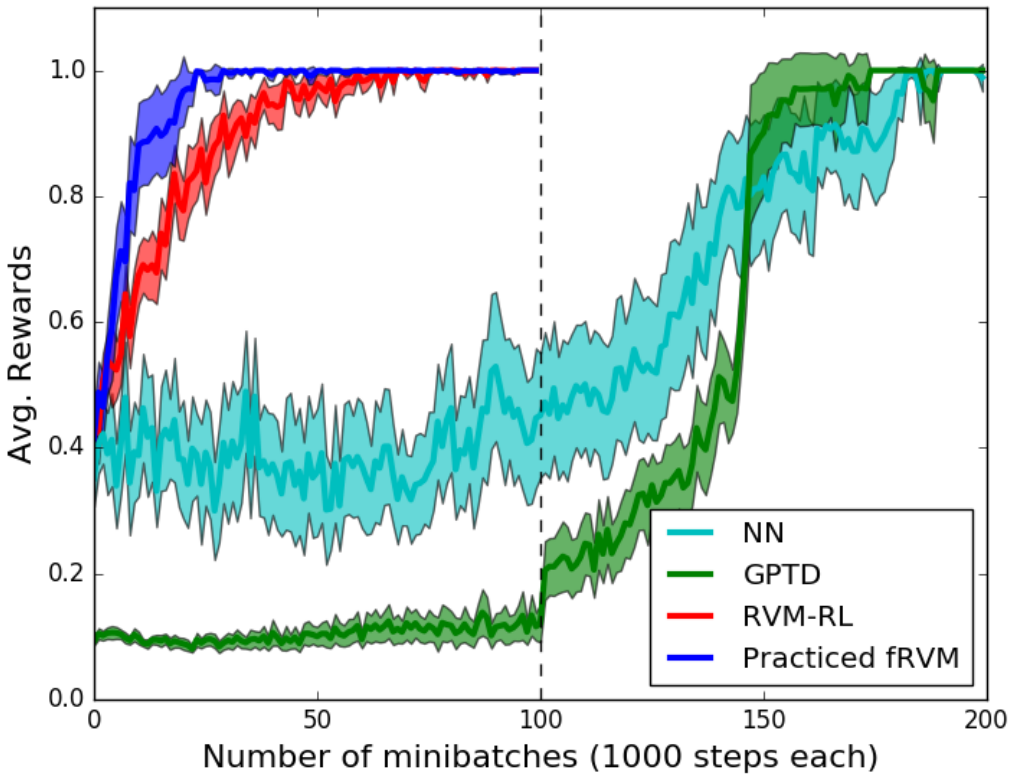


Figure 6.4: Average of rewards for each episode in cart-pole balancing. Again, practice helps to converge quickly at the optimal policy. 100 samples (10 % of the number of steps in one minibatch) are used for practice. The shaded areas represent 95 % confidence interval.

reward between 100 and 140 is given when it lands near zero speed. When it lands in the goal and rests, it gets additional 100 while it gets -100 when it crashes on the surface. Firing main engine cost -0.3, and each leg contact to ground gives 10.

For this problem, most samples that are collected during practice are only the crashing on the surface. Thus, without strategic practice sampling, it gathers samples without any positive bases around the high rewarding states, and resulting bases make feature values to near-zero, preventing a good estimation of Q values.

This problem gets worse in CarRacing-v0. CarRacing is a problem that controls a racing car from the top-down image of the racing environment. An agent controls steering, acceleration and deceleration. The states are represented by 96 by 96 image pixels, and each frame costs -0.1 reward value. Visiting each track tile is worth $1000/N$ when the number of track tiles are N.

Similar to LunarLander-v2, CarRacing practice does not sample enough. Mostly it gathers samples around the starting position and makes it hard to estimate Q values when a car travels far from the starting region with zero feature values. Thus, more strategic practice approaches are required for complex domain problems. We will discuss this issue in next section in detail.

6.4 Discussion

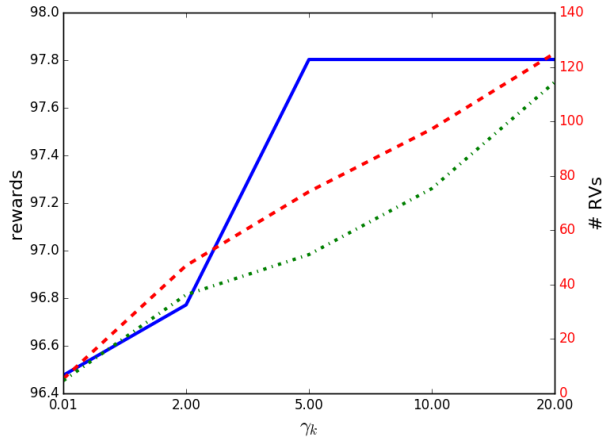
From the results in the previous section, two questions arise. How does the kernel parameter affect learning? After fRVM-RL training, does it select right basis only for the RL task? We discuss these questions in this section.

6.4.1 Analysis of Practice

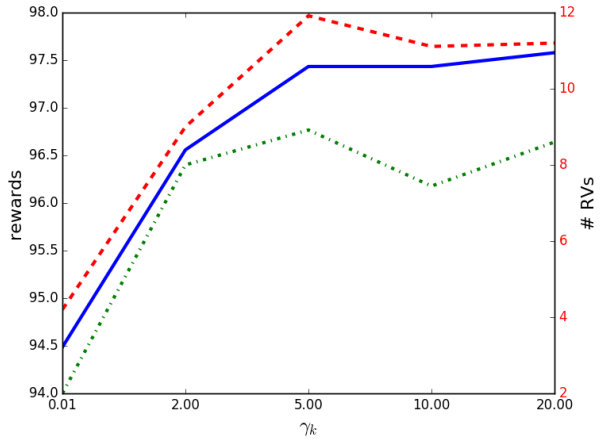
To answer the first question, we plot Figure 6.5. The blue line represents the mean of the area under the mean reward curve. For this plot, 10 test results for each γ_k value are collected. The red line and green line represent the number of RVs after practice and active RVs after RL-train. Here, the number of active RVs are recorded by counting the weights greater than 1.0×10^{-5} in magnitude.

As γ_{k} grows, the base width for RBF gets smaller, which results in more of RVs. However, the number of active RVs decreases because only task-related bases will capture the significant mass [85]. Thus, we can observe that with an RBF kernel, the selection of the kernel parameter greatly affects basis construction and reinforcement learning performance.

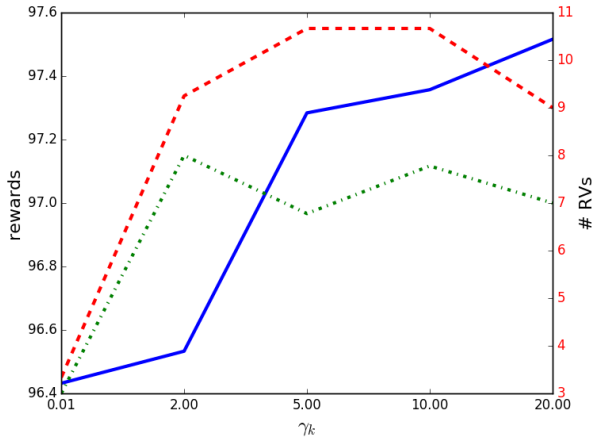
Another observation from Figure 6.5 is the difference between sampling methods. With random sampling, the practice stage generates a larger number of RV bases while dynamic sampling eliminates unnecessary RVs. In the case of next state prediction, the small state change makes the sample similar to existing bases so that it can discard similar RVs. With state change prediction, the RVM can remove samples that do not incur state changes, which can also result in a reduction in the number of RVs. Interestingly, when a large number of RVs are used as a fixed basis, the active RVs are spread over almost all of the basis and the number is not reduced. We can intuitively assume that this is caused by the small likelihood that randomly generated samples coincide with the true basis. This investigation answers the question that we raised in our previous pretraining study with neural networks. Random generation of samples seems to be less likely to generate a good basis that is near-orthogonal.



(a) random sampling



(b) dynamic sampling - next state prediction



(c) dynamic sampling - state change prediction

Figure 6.5: The effects of the RBF kernel γ_k selection with different sampling and target options. The green dashed dot line represents the number RVs after practice, and the red dashed line shows the number RVs with non-zero weights. The blue line depicts the mean of the area under the reward curve. The blue line is scaled on the left reward y-axis and the other two are scaled on the right # RVs y-axis. Only average values are presented for clear reading of plots. The variances of the number of RVs (green and blue) are less than 1 in (b) and (c) and less than 6 in (a). The range of variation in the reward values is between 1.06 and 6.89.

6.4.2 Construction of Bases

A few authors have previously studied basis construction for supervised learning or semi-supervised learning. Raina, et al., [18] posed the self-taught learning approach that requires the learned structure (or basis) from unlabeled data to be applicable to labeled classification tasks. This enables transfer from unsupervised learning to supervised learning tasks by building a basis from unsupervised learning training. Deep learning [113–116] pretrains hidden layers of neural networks in unsupervised ways and learns the network connectivity structures to be applicable to a target supervised learning task. However, none of these considered reinforcement learning tasks that learn from evaluations of an agent's behavior or reinforcements rather than from known output labels. Anderson, et al., [96] first proposed constructing neural network structures from state dynamics prediction for reinforcement learning. However, it is difficult to interpret the learned network structure. By using RVMs, we clearly see the RV bases and how features are generated with a kernel function. This increases the understanding of the learned bases and environment dynamics.

Furthermore, by providing fixed-basis RVM, the approach increases the efficiency of learning. Classical radial basis functions [35], polynomial bases [117], and Fourier bases [118] work well, but how to select a basis is not well understood. Learning in a small source task, proto value functions (PVFs) [119] automatically specify an ortho-normal set of basis functions. This bases can be transferred to tasks with different goals or in a slightly different state space. Practice poses a harder problem and requires samples without reinforcements or objectives. The RVM bases learned through practice can be transferred to a broader range of tasks than PVFs.

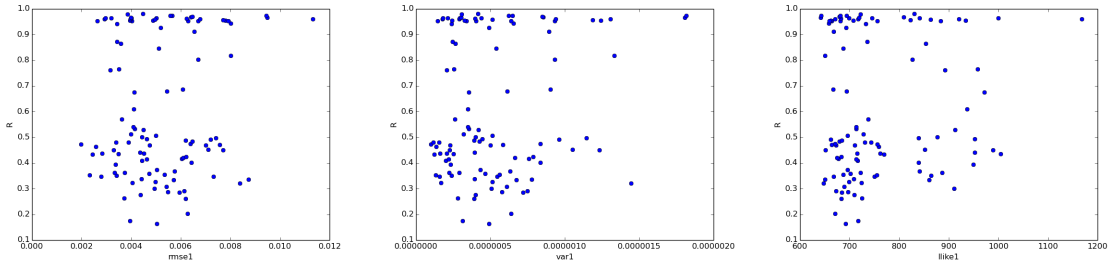
When the RV bases are well-established from practice, they support successful learning. As we discussed earlier, when unstable sampling or learning parameters are chosen, it is possible to learn

poorly on reinforcement learning tasks. For efficient practice, we can investigate 1) kernel methods for reliable basis construction, 2) practice strategy development, 3) search space reduction, 4) cyclic training of practice and learning, and 5) non-fixed, dynamic learning based on practiced knowledge.

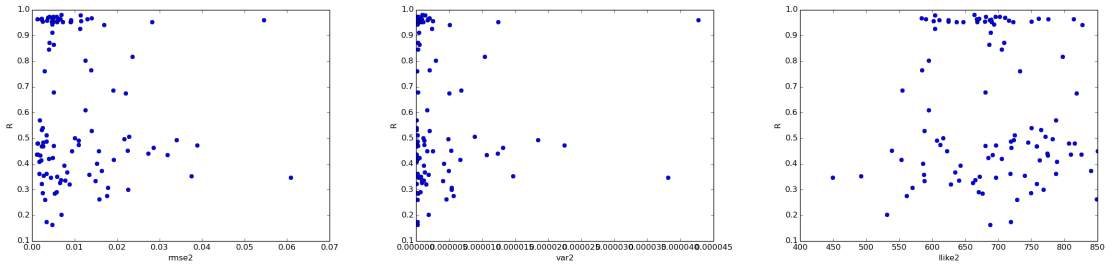
Knowing the effectiveness of bases can automate the basis construction during practice. That is, we can automate the process of finding a good kernel and its parameters. Also, when the found bases are not good enough, we can restart practice or increase the number of samples. Or, with cycles of practice and reinforcement learning, learning can be improved further.

For this, we examine if there is any correlation between the hyperparameters and average rewards. Some preliminary tests were run to evaluate this and collected 100 samples with successful runs ($\sum_t r_t >= 0.9 \times N$ where N is the number steps) and poor ones ($\sum_t r_t < 0.9 \times N$).

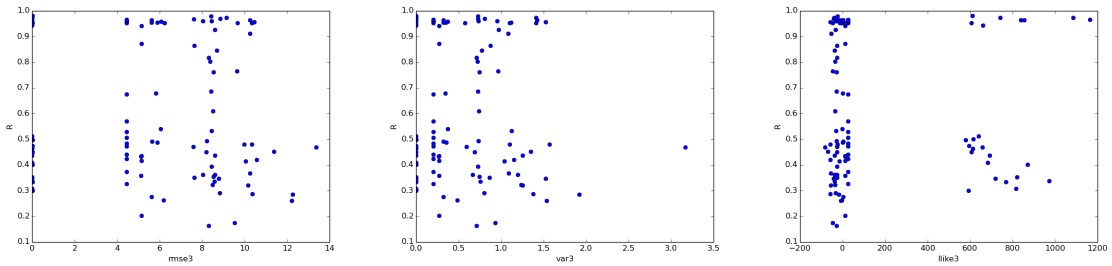
We recorded practice RMSEs, log likelihoods, and variances for each output dimension along with average rewards. We observed that the data is scattered wide and trends are not obvious (Figure 6.6). We tested some classification algorithms, such as LDA, QDA, linear and nonlinear logistic regression, to see if it can be classified. The label is set to true if the mean reward is greater than 0.9 and false otherwise. Table 6.1 shows the classification accuracy with the 12 features. Nonlinear logistic regression seems to clearly separate successful cases by looking at the RMSE, variance and log-likelihood. This tells us that the selected features are strongly related to the subsequent reinforcement learning performance, so can be the bases for an approach to predicting the success of a practice model. We will investigate this further with more samples and other environments to examine if it can be generalized to other tasks.



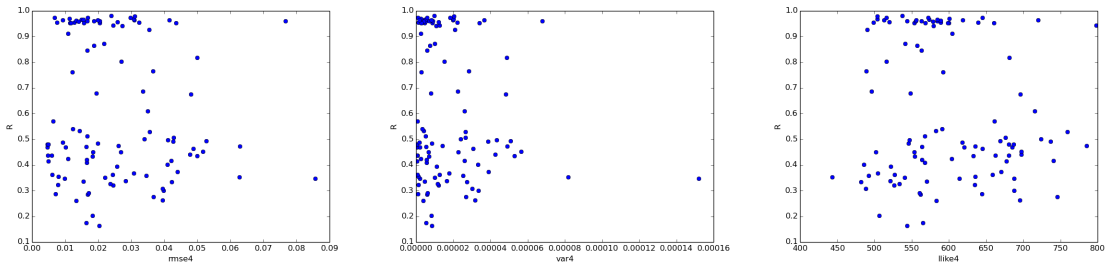
(a) rmse1 **(b)** var1 **(c)** loglike1



(d) rmse2 **(e)** var2 **(f)** loglike3



(g) rmse3 **(h)** var3 **(i)** loglike3



(j) rmse4 **(k)** var4 **(l)** loglike4

Figure 6.6: Scatter plots of features against average rewards. Errors, variances, and log likelihoods for each dimension are selected features.

Table 6.1: Preliminary evaluations for examination of automated practice with 100 practice and fRVM-RL samples

Classifier	Accuracy
LDA	63 / 100
QDA	69 / 100
Linear Logistic Regression	74 / 100
Nonlinear Logistic Regression	100 / 100

6.5 Conclusion

We examined the efficacy of practice for reinforcement learning tasks and observed increased efficiency. By training an RVM and obtaining RV bases from dynamics prediction, we were able to successfully transfer the bases and to show improved learning in classical benchmark problems. With fixed basis learning, we demonstrated how effectively practice establishes bases in these examples. Also, we observed the limitations of practice when the tasks get complex. Discussion leads to plans for investigation of more efficient ways to practice and the measure that evaluates how well bases are constructed.

A major contribution of this chapter is the demonstration of the importance of practice in a machine learning context. Without providing any objective or reinforcement, practice with an RVM extends exploration before starting to solve the actual RL task and generates sparse but helpful bases. As humans practice to develop faster reactions or better performance in real situations, practice in reinforcement learning even without any goal-directives is expected to improve subsequent learning with reinforcement signals. Additional experience with knowledge construction from practice, in the form of sets of relevance vectors (RVs), turns out to be the key reason that makes this work.

From our discussion about results and practice evaluation, we can further investigate stable kernel methods and search space reduction. Practice strategies such as human or agent guided practice ("coaching"), and cyclic repetition of short practice and short learning will be interesting direction for future study. Allowing adaption of the basis learned from practice might be necessary not only to compensate for lack of practice but also to be easily extended to the continuous action tasks [57, 120]. However, investigation of efficient transfer learning should be studied due to possible disturbances from practice. This line of research will be continued through additional experiments with the OpenAI Gym tasks.

Chapter 7

Conclusions

Memory is one of the most important components for successful learning and generalization of knowledge. This dissertation is motivated by cognitive information processing steps of acquisition, retention, and transfer of knowledge. When adopting these steps in intelligent systems, how to retrieve and reuse knowledge is the challenging problem. The proposed framework suggests the principled, systematic knowledge acquisition and retention. Since the snapshots can be representative information about a task, it can be considered as a knowledge representation that can be generalized over tasks. By transferring the knowledge or snapshots, we examined the generalization of them as well. The results summarized here show that the knowledge retention approach through the framework can make learning robust and efficient by preventing forgetting that causes instability of learning. Snapshots are shown its analytical power, improve learning performance in continuous action problems, and generalization over the novel practice application.

7.1 Contributions

Main contributions of this dissertation to reinforcement learning research can be summarized as follows:

- **Impact of the modifications of RVMs:**

  Various versions of the relevance vector machines extend the applications of them. From fully online and low memory learning, oRVM can be easily adopted. By adding a bit more memory with a buffer, boRVM can increase the accuracy of prediction greatly. For multi-

ple target problems, moRVM can applicable. In general, Bayesian learning allows a user's contribution by defining prior distributions. fRVM with fixed bases can reflect users' in-depth contribution by adopting human knowledge about the domain for improved learning performance.

- **Impact of sparse learning:**

  By adopting sparse Bayesian regression model, RVMs, and filtering stages in the framework, the snapshots are managed to keep the low sparsity level. The small number of snapshots lowers the memory requirement for learning to be applicable to complex real-world problems. Learning upon the sparse snapshots minimizes the computational overhead.

- **Bayesian learning:**

  Relevance vector machine, the Bayesian model, provides the advantages of Bayesian learning. Thus, the framework provide a room for human domain knowledge contribution by defining prior distributions. Also, the extended state information including covariance improves the learning algorithms. One of the usages is using the variance information for filtering heuristics. Based on the confidence information about current Q estimates, we can filter out low confident experience.

- **Improved understanding of domain and solutions:**

  As we discussed in the previous chapters, the policies learned by the proposed frameworks can lead to a useful analysis of the state-action space structure by examining the RVs after training. Since the snapshots are raw input samples, we can visualize or examine each of them to see what snapshots are left after the learning process is done. Some snapshots can

be related to the learned solution, and the others can represent the domain or environmental knowledge.

- **Generalization of snapshot memory:**

  Snapshot memory is the knowledge that an agent has obtained from the learning process with the proposed framework. Thus, we can consider the snapshots as the new knowledge representation that allows us to transfer them for training. From the cognitive information processing steps (Figure 1.1), we have discussed that the framework provides the first two stages. We use the snapshots in the generalization step by a simple transfer learning approach. The snapshots can be utilized in imitation or advising for heterogeneous transfer learning. In case of homogeneous transfer, the snapshots can be directly transferred to be used as the initial bases.

- **Improved learning with snapshot memory:**

  Analysis and thorough observations of snapshots discovered that the snapshot samples are located at the modes of Q value estimation curves when Gaussian kernel is used. The fact that this observation is dependent on the choice of the kernel function does not degrade the possibility of the proposed approach. Rather, it promotes more possibilities of new discoveries with various kernels selection. Thus, the analysis of snapshots will lead valuable findings to further improve reinforcement learning algorithms.

- **Impact of practice with snapshot:**

  The practice approach can lower the required interaction with an environment vastly. Especially when the cost of real samples are expensive, practice can lower the burden without using the environment. Furthermore, before tackling goal-directed tasks, we can collect pre-

knowledge about the domain in the form of snapshots and provide the vast opportunity to learn the domain and eventually improve learning performance. Results demonstrate how practiced knowledge contributes to reinforcement learning. The discussion leads to how we can further improve practice strategies to gain efficiency of learning. With the addition of strategic practice, its impacts on various applications are promising.

- **First solution of the full octopus arm control:**

  So far, only Engel, et al., [65] suggested the solution for the octopus arm control problem. However, they limited the number of actions to 6 predefined discrete actions. Without restricting the number of actions, even removing the limitation of discrete control on each muscle, our approach is the first solution to the complex octopus control problem. This envisions the possibility of broad applications of the proposed approaches.

7.2 Future Works

This dissertation examined the advancement of knowledge retention and transfer for reinforcement learning. Although it showed large potential of the proposed approaches, it has still many more to develop. The following lists are future directions to further investigate:

- **Hierarchical continuous action sampling for further efficiency:**

  Although the proposed snapshot sampling approaches are efficient and computational costs are low, unnecessary additional computation can result if we use continuous actions when it is not necessary. Child development research such as Piek, et al., [121] examines different intercorrelations between movement and self-perception. The movement is considered in terms of fine and gross motor skills. Fine motor skills are in charge of delicate task or move-

ment while gross motor skills run large muscle movement. Similarly, we can differentiate the level of control based on hierarchical reinforcement learning [122–127]. The upper level states use the gross control commands, which are discrete values, and the lower level states apply the fine controls with continuous snapshot action sampling. By choosing a right tool for each level, we can improve the efficiency and at the same time, we can save unnecessary computation time.

- **Repetition of practice and learning:**

  In Chapter 6, we proposed two-stage learning model with practice. In the end, we discussed how we can evaluate the practice and how helpful the discovered bases are for the target task. Once we discover better strategy for practice, it can increase the chances to perform better on the target task. Instead of finishing training after one pass, repetition of practice and learn can further improve the performance. In addition, practice can be added in shaping transfer learning approaches, which increase the complexity of problems for transfer learning. Practice in between the transfer can smooth the knowledge transfer process and is expected to learn efficiently.

- **Strategic practice for jumpstart:**

  Jumpstart [24] means the enhanced performance in the beginning of a target task learning after transfer. That is, when a knowledge is transferred from a source to a target task, jumpstart expects high utility of the knowledge from the very early stage of target learning. While practice can speed learning and shorten the required samples to learn, it does not contribute to jumpstarting on the target task. This is an intuitive result since the practice does not have any information to the target task. However, we can ask if there is any practice strategies that

can improve initial performance of learning. This direction of research can be investigated along with the above repetitive approach.

- **Theoretical examination of snapshot contribution on stability and convergence:**

  So far, we have examined the stability and efficiency of learning empirically. The convergence and stability need to be more investigated theoretically to reexamine the soundness of the approaches. This direction of research will improve our understanding of the approaches and be able to enhance the methods.

- **Multiple snapshots for knowledge base:**

  The snapshot memory can be stored for a task. When the learning process is done, instead of getting rid of the snapshots, storing them for future training can construct multiple snapshot models, eventually to build a knowledge base. The multiple snapshots or a knowledge base can make learning from the existing knowledge not from scratch. The major issue when developing this approach is how to efficiently evaluate new data samples to compute the kernel relations with the large knowledge base. Efficient and fast kernel computation is a key to success for this approach.

- **Snapshots for causality analysis:**

  This dissertation has shown that snapshots can be used for analysis of the problem and the solution. The analytical strength of snapshots can be used to find possible causality relations to a certain results. Recent deep learning research [128,129] starts to peer inside the blackbox model of the deep networks about why or how the networks make a good prediction or the reason for a specific error when it occurs. However, neural networks are not easily interpretable. The sparse Bayesian reinforcement learning framework and snapshots can

provide a plenty of information to analyze the cause of an incident. For example, when an error happens in a manufacturing belt, our snapshot learning model can provide the snapshots to better understand the reason to prevent the same errors happening again.

- **Convergent RVM-RL:**

  Greedy-GQ [56, 130] is proved convergent in the mean squared projected Bellman error (MSPBE). Previous research [57] showed that nonlinear Greedy-GQ algorithm with neural networks function approximation results in stable learning. Instead of mean squared Bellman error (MSBE), the RVM-RL framework can adopt MSPBE as an objective function to minimize. A brief sketch for the convergent RVM-RL follows.

$$\begin{aligned} MSPBE(\theta) &= \|Q_\theta(s,a) - \Pi T Q_\theta(s,a)\|_\mu^2 \\ &= \|\Pi[r_{t+1} + \gamma Q_\theta(s,a)] - Q_\theta(s,a)\|_\mu^2 \end{aligned}$$

  where Π is a projection operator to the linear space and $\mu(s)$ represents state visitation probability. The projection operator takes any value function to the nearest value function approximation, so the projection operator Π is defined as:

$$\Pi = \Phi(\Phi^\top D \Phi)^{-1} \Phi^\top D$$

  where D is a diagonal matrix with $\mu(s)$. The norm $\|Q\|_\mu^2 = \int Q^2(s,a)\mu(ds,da)$.

  For the RVM-RL, let current basis Φ_t, sample state visitation probability D_t, and the relevant project Π_t:

$$\Pi_t = \Phi_t(\Phi_t^\top D_t \Phi_t)^{-1} \Phi_t^\top D_t.$$

Now, using the projection Π_t, the prediction target t' is

$$t' = \Pi_t \left(r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) \right).$$

The log of margianl likelihood for an RVM is

$$L(\boldsymbol{\alpha}) = \log \mathbb{P}(\mathbf{t}|\mathbf{x}, \mathbf{w}, \boldsymbol{\alpha}, \beta) = \frac{M}{2} \ln \boldsymbol{\alpha} + \frac{N}{2} \ln \boldsymbol{\beta} - E(\mathbf{w}) - \frac{1}{2} \ln |A| - \frac{N}{2} \ln 2\pi.$$

where

$$\begin{aligned} E(\mathbf{w}) &= \frac{\beta}{2} \|\mathbf{t} - \boldsymbol{\Phi}\mathbf{w}\|^2 + \frac{\alpha}{2} \mathbf{w}^\top \mathbf{w} \\ &= \frac{\beta}{2} \|\Pi_t(r_{t+1} + \gamma Q(s_{t+1}, a_{t+1})) - \boldsymbol{\Phi}\mathbf{w}\|^2 + \frac{\alpha}{2} \mathbf{w}^\top \mathbf{w}. \end{aligned}$$

Comparing this approach with the MSBE-based RVM-RL, we can examine convergence of the RVM-RL both in theoretical and empirical way.

Bibliography

[1] Bernhard Scholkopf and Alexander J Smola. *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2001.

[2] Yoram Yekutieli, Roni Sagiv-Zohar, Ranit Aharonov, Yaakov Engel, Binyamin Hochner, and Tamar Flash. Dynamic model of the octopus arm. i. biomechanics of the octopus reaching movement. *Journal of Neurophysiology*, 94(2):1443–1458, 2005.

[3] David P Ausubel. *The psychology of meaningful verbal learning*. Grune & Stratton, 1963.

[4] David Paul Ausubel. *The acquisition and retention of knowledge: A cognitive view*. Springer Science & Business Media, 2012.

[5] Thomas K Srull. The role of prior knowledge in the acquisition, retention, and use of new information. *Advances in Consumer Research*, 10:572–576, 1983.

[6] Alice F Healy, James A Kole, and Lyle E Bourne Jr. Training principles to advance expertise. *Frontiers in Psychology*, 5:131, 2014.

[7] L. Mihalkova, T. Huynh, and R.J. Mooney. Mapping and revising markov logic networks for transfer learning. In *Proceedings of the National Conference on Artificial Intelligence*, volume 22, page 608. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press, 2007.

[8] S.J. Pan and Q. Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, October 2010.

[9] Lorien Pratt and Barbara Jennings. *A Survey of Connectionist Network Reuse Through Transfer*, pages 18–43. Kluwer Academic Publishers, Norwell, MA, USA, 1998.

[10] Z. Kira. Inter-robot transfer learning for perceptual classification. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*, pages 13–20. International Foundation for Autonomous Agents and Multiagent Systems, 2010.

[11] G. Konidaris and A. Barto. Autonomous shaping: Knowledge transfer in reinforcement learning. In *Proceedings of the 23rd International Conference on Machine learning*, pages 489–496. ACM, 2006.

[12] G. Kuhlmann and P. Stone. Graph-based domain mapping for transfer learning in general games. *European Conference on Machine Learning (ECML)*, pages 188–200, 2007.

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

[14] 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.

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

[16] L. Torrey, J. Shavlik, T. Walker, and R. Maclin. Relational macros for transfer in reinforcement learning. In *Proceedings of the 17th International Conference on Inductive Logic Programming*, pages 254–268. Springer-Verlag, 2007.

[17] L. Torrey and J. Shavlik. Transfer learning. *Handbook of Research on Machine Learning Applications. IGI Global*, 3:17–35, 2009.

[18] Rajat Raina, Alexis Battle, Honglak Lee, Benjamin Packer, and Andrew Y Ng. Self-taught learning: transfer learning from unlabeled data. In *Proceedings of the 24th International Conference on Machine Learning (ICML)*, pages 759–766, 2007.

[19] K. Ferguson and S. Mahadevan. Proto-transfer learning in markov decision processes using spectral methods. In *Proceedings of the Twenty-third International Conference on Machine Learning Workshop on Structural Knowledge Transfer for Machine Learning*, 2006.

[20] W. Dai, Q. Yang, G.R. Xue, and Y. Yu. Boosting for transfer learning. In *Proceedings of the 24th International Conference on Machine learning*, pages 193–200, 2007.

[21] Matthew E Taylor and Peter Stone. Behavior transfer for value-function-based reinforcement learning. In *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 53–59, 2005.

[22] E. Talvitie and S. Singh. An experts algorithm for transfer learning. In *Proceedings of the twentieth International Joint Conference on Artificial Intelligence*, pages 1065–1070, 2007.

[23] M. Taylor, N. Jong, and P. Stone. Transferring instances for model-based reinforcement learning. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 488–505, 2008.

[24] M.E. Taylor and P. Stone. Transfer learning for reinforcement learning domains: A survey. *The Journal of Machine Learning Research*, 10:1633–1685, 2009.

[25] E. Eaton, M. Desjardins, and T. Lane. Modeling transfer relationships between learning tasks for improved inductive transfer. *Machine Learning and Knowledge Discovery in Databases*, pages 317–332, 2008.

[26] Rajat Raina, Andrew Y Ng, and Daphne Koller. Constructing informative priors using transfer learning. In *Proceedings of the 23rd international Conference on Machine learning (ICML)*, pages 713–720, 2006.

[27] Vincent Wenchen Zheng, Sinno Jialin Pan, Qiang Yang, and Jeffrey Junfeng Pan. Transferring multi-device localization models using latent multi-task learning. In *Proceedings of the 23rd AAAI Conference on Artificial Intelligence (AAAI)*, volume 8, pages 1427–1432, 2008.

[28] Jing Gao, Wei Fan, Jing Jiang, and Jiawei Han. Knowledge transfer via multiple model local structure mapping. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 283–291, 2008.

[29] Michael McCloskey and Neal J Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. *Psychology of learning and motivation*, 24:109–165, 1989.

[30] Robert M French. Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences*, 3(4):128–135, 1999.

[31] Roger Ratcliff. Connectionist models of recognition memory: Constraints imposed by learning and forgetting functions. *Psychological review*, 97(2):285–308, 1990.

[32] James L McClelland, Bruce L McNaughton, and Randall C O'reilly. Why there are complementary learning systems in the hippocampus and neocortex: insights from the successes and failures of connectionist models of learning and memory. *Psychological review*, 102(3):419, 1995.

[33] Ronald J Brachman, Hector J Levesque, and Raymond Reiter. *Knowledge representation*. MIT press, 1992.

[34] Ronald Brachman and Hector Levesque. *Knowledge Representation and Reasoning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.

[35] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT press Cambridge, 1998.

[36] 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.

[37] Theodore J Perkins and Mark D Pendrith. On the existence of fixed points for q-learning and sarsa in partially observable domains. In *Proceedings of International Conference on Machine Learning (ICML)*, pages 490–497, 2002.

[38] Alex J. Smola and Bernhard Schölkopf. A tutorial on support vector regression. *Statistics and computing*, 14(3):199–222, 2004.

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

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

[41] C.M. Bishop. *Pattern Recognition and Machine Learning*. Information Science and Statistics. Springer, 2006.

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

[43] Bellman Richard. Dynamic programming. *Princeton University Press*, 89:92, 1957.

[44] Satinder P. Singh and Richard S. Sutton. Reinforcement learning with replacing eligibility traces. *Machine learning*, 22(1-3):123–158, 1996.

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

[46] Leemon Baird. Residual algorithms: Reinforcement learning with function approximation. In *Proceedings of 12th International Conference on Machine Learning*, pages 30–37, 1995.

[47] J.N. Tsitsiklis and B. Van Roy. An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, 42(5):674–690, 2002.

[48] James Sacra Albus. *Brains, behavior, and robotics*. Byte books. BYTE Books, 1981.

[49] 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.

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

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

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

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

[54] Peter Stone and Richard S. Sutton. Scaling reinforcement learning toward robocup soccer. In *Proceedings of International Conference on Machine Learning (ICML)*, volume 1, pages 537–544, 2001.

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

[56] Hamid Reza Maei. *Gradient temporal-difference learning algorithms*. PhD thesis, University of Alberta, 2011.

[57] 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)*, pages 1–8, 2014.

[58] 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.

[59] 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.

[60] Mario Martin. On-line support vector machines for function approximation. Technical report, Universitat Politècnica de Catalunya, Departament de Llengatges i Sistemes Informàtics, 2002.

[61] 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. IEEE, 2009.

[62] 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. IEEE, 2010.

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

[64] 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.

[65] 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.

[66] 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.

[67] 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.

[68] Dimitri P Bertsekas and John N Tsitsiklis. Neuro-dynamic programming: an overview. In *Proceedings of the 34th IEEE Conference on Decision and Control*, volume 1, pages 560–564, 1995.

[69] Satinder P Singh and Dimitri P Bertsekas. Reinforcement learning for dynamic channel allocation in cellular telephone systems. In *Advances in neural information processing systems*, pages 974–980, 1997.

[70] Honglak Lee, Yirong Shen, Chih-Han Yu, Gurjeet Singh, and Andrew Y Ng. Quadruped robot obstacle negotiation via reinforcement learning. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, pages 3003–3010, 2006.

[71] Richard S Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. *Advances in neural information processing systems*, pages 1038–1044, 1996.

[72] David H Wolpert, William G Macready, et al. No free lunch theorems for search. Technical report, SFI-TR-95-02-010, Santa Fe Institute, 1995.

[73] A. Ben-Hur, C.S. Ong, S. Sonnenburg, B. Schölkopf, and G. Rätsch. Support vector machines and kernels for computational biology. *PLoS Comput Biol*, 4(10):e1000173, 2008.

[74] Vladimir Vapnik. *The nature of statistical learning theory*. Springer Science & Business Media, 2013.

[75] Junshui Ma, James Theiler, and Simon Perkins. Accurate on-line support vector regression. *Neural Computation*, 15(11):2683–2703, 2003.

[76] Mark E Glickman and David A van Dyk. Basic bayesian methods. *Topics in Biostatistics*, pages 319–338, 2007.

[77] Carl Edward Rasmussen. Gaussian processes for machine learning. MIT Press, 2006.

[78] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

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

[80] 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.

[81] Mohammad Ghavamzadeh, Shie Mannor, Joelle Pineau, Aviv Tamar, et al. Bayesian reinforcement learning: A survey. *Foundations and Trends® in Machine Learning*, 8(5-6):359–483, 2015.

[82] Arnab Nilim and Laurent El Ghaoui. Robust control of markov decision processes with uncertain transition matrices. *Operations Research*, 53(5):780–798, 2005.

[83] Michael E Tipping. Bayesian inference: An introduction to principles and practice in machine learning. In *Advanced lectures on machine Learning*, pages 41–62. 2004.

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

[85] David Wipf, Jason Palmer, and Bhaskar Rao. Perspectives on sparse bayesian learning. In *Advances in Neural Information Processing Systems 16*, pages 249–256, 2003.

[86] Anita C Faul and Michael E Tipping. Analysis of sparse bayesian learning. *Advances in neural information processing systems*, 1:383–390, 2002.

[87] Francesco Parrella. Online support vector regression. *Master's Thesis, Department of Information Science, University of Genoa, Italy*, 2007.

[88] Arasanathan Thayananthan, Ramanan Navaratnam, Björn Stenger, Philip Torr, and Roberto Cipolla. Multivariate relevance vector machines for tracking. *Computer Vision–ECCV 2006*, pages 124–138, 2006.

[89] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[90] John Platt. A resource-allocating network for function interpolation. *Neural computation*, 3(2):213–225, 1991.

[91] Martin Riedmiller. Neural fitted q iteration–first experiences with a data efficient neural reinforcement learning method. In *European Conference on Machine Learning (ECML)*, pages 317–328. 2005.

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

[93] 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.

[94] 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. ACM, 2006.

[95] 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.

[96] 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)*, pages 1–7, 2015.

[97] Hamid Reza Maei, Csaba Szepesvári, Shalabh Bhatnagar, Doina Precup, David Silver, and Richard S. Sutton. Convergent temporal-difference learning with arbitrary smooth function approximation. In *Proceedings of the Advances in Neural Information Processing Systems (NIPS)*, pages 1204–1212, 2009.

[98] Alessandro Lazaric Marcello Restelli Andrea Bonarini. Reinforcement learning in continuous action spaces through sequential monte carlo methods. *Proceedings of the Advances in Neural Information Processing Systems (NIPS)*, 20:833–840, 2008.

[99] Hado Van Hasselt and Marco A Wiering. Reinforcement learning in continuous action spaces. In *IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*, pages 272–279, 2007.

[100] Chris Gaskett, David Wettergreen, and Alexander Zelinsky. Q-learning in continuous state and action spaces. In *Australasian Joint Conference on Artificial Intelligence*, pages 417–428, 1999.

[101] José Del R Millán, Daniele Posenato, and Eric Dedieu. Continuous-action q-learning. *Machine Learning*, 49(2-3):247–265, 2002.

[102] Jonas Mockus. Application of bayesian approach to numerical methods of global and stochastic optimization. *Journal of Global Optimization*, 4(4):347–365, 1994.

[103] Minwoo Lee and Charles W. Anderson. Robust reinforcement learning with relevance vector machines. In *Proceedings of the 1st international workshop on robot learning and planning (RLP)*, 2016.

[104] Yaakov Engel, Shie Mannor, and Ron Meir. Bayesian reinforcement learning with gaussian process temporal difference methods. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.205.8454, 2007. (unpublished) Accessed: 2017-05-23.

[105] William M Kier and Kathleen K Smith. Tongues, tentacles and trunks: the biomechanics of movement in muscular-hydrostats. *Zoological Journal of the Linnean Society*, 83(4):307–324, 1985.

[106] RL Community. Rl-competition, 2009. [Online; accessed 2-May-2017].

[107] 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.

[108] Minwoo Lee and Charles W. Anderson. Can a reinforcement learning agent practice before it starts learning? In *International Joint Conference on Neural Networks (IJCNN)*, pages 4006–4013, 2017.

[109] John Blitzer, Ryan McDonald, and Fernando Pereira. Domain adaptation with structural correspondence learning. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, pages 120–128, 2006.

[110] Pengcheng Wu and Thomas G Dietterich. Improving svm accuracy by training on auxiliary data sources. In *Proceedings of the twenty-first international conference on Machine learning*, page 110, 2004.

[111] Matthew E Taylor and Peter Stone. Cross-domain transfer for reinforcement learning. In *Proceedings of the 24th international conference on Machine learning*, pages 879–886, 2007.

[112] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[113] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

[114] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, 2006.

[115] Dumitru Erhan, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, Pascal Vincent, and Samy Bengio. Why does unsupervised pre-training help deep learning? *The Journal of Machine Learning Research (JMLR)*, 11:625–660, 2010.

[116] Yoshua Bengio. Learning deep architectures for ai. *Foundations and trends® in Machine Learning*, 2(1):1–127, 2009.

[117] Michail G Lagoudakis and Ronald Parr. Least-squares policy iteration. *The Journal of Machine Learning Research (JMLR)*, 4:1107–1149, 2003.

[118] G.D. Konidaris, S. Osentoski, and P.S. Thomas. Value function approximation in reinforcement learning using the Fourier basis. In *Proceedings of the 25th Conference on Artificial Intelligence*, pages 380–385, August 2011.

[119] Sridhar Mahadevan and Mauro Maggioni. Proto-value functions: A laplacian framework for learning representation and control in markov decision processes. *The Journal of Machine Learning Research (JMLR)*, 8(2169–2231):16, 2007.

[120] Minwoo Lee and Charles W. Anderson. Relevance vector sampling for reinforcement learning in continuous action space. In *15th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 774–779, 2016.

[121] Jan P Piek, Grant B Baynam, and Nicholas C Barrett. The relationship between fine and gross motor ability, self-perceptions and self-worth in children and adolescents. *Human movement science*, 25(1):65–75, 2006.

[122] Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999.

[123] Doina Precup. *Temporal abstraction in reinforcement learning*. PhD thesis, University of Messachusetts Amherst, 2000.

[124] Ronald Edward Parr. *Hierarchical control and learning for Markov decision processes*. PhD thesis, University of California at Berkeley, 1998.

[125] Thomas G Dietterich. Hierarchical reinforcement learning with the maxq value function decomposition. *Journal of Artifical Intelligent Research (JAIR)*, 13:227–303, 2000.

[126] M. Ghavamzadeh and S. Mahadevan. Learning to communicate and act using hierarchical reinforcement learning. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 3*, pages 1114–1121. IEEE Computer Society, 2004.

[127] Bhaskara Marthi, Stuart J Russell, David Latham, and Carlos Guestrin. Concurrent hierarchical reinforcement learning. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, pages 779–785, 2005.

[128] Grégoire Montavon, Sebastian Lapuschkin, Alexander Binder, Wojciech Samek, and Klaus-Robert Müller. Explaining nonlinear classification decisions with deep taylor decomposition. *Pattern Recognition*, 65:211–222, 2017.

[129] Arnab Paul and Suresh Venkatasubramanian. Why does deep learning work? - A perspective from group theory. *arXiv preprint arXiv:1412.6621*.

[130] Hamid Reza Maei, Csaba Szepesvári, Shalabh Bhatnagar, and Richard S. Sutton. Toward off-policy learning control with function approximation. In *Proceedings of the 27th International Conference on Machine Learning (ICML)*, pages 719–726, 2010.