
Reinforcement learning with kernels and Gaussian processes

Reinforcement Learning, Gaussian Processes, Temporal Difference Learning, Kernel Methods

Yaakov Engel

YAKI@ALICE.NC.HUJI.AC.IL

Interdisciplinary Center for Neural Computation, The Hebrew University of Jerusalem, Jerusalem 91904, Israel

Shie Mannor

SHIE@ECE.MCGILL.CA

Dept. of Electrical and Computer Engineering, McGill University, Montreal, Canada

Ron Meir

RMEIR@EE.TECHNION.AC.IL

Dept. of Electrical Engineering, Technion Institute of Technology, Haifa 32000, Israel

Abstract

Kernel methods have become popular in many sub-fields of machine learning with the exception of reinforcement learning; they facilitate rich representations, and enable machine learning techniques to work in diverse input spaces. We describe a principled approach to the policy evaluation problem of reinforcement learning. We present a temporal difference (TD) learning using kernel functions. Our approach allows the TD algorithm to work in arbitrary spaces as long as a kernel function is defined in this space. This kernel function is used to measure similarity between states. The value function is described as a Gaussian process and we obtain a Bayesian solution by solving a generative model. A SARSA based extension of the kernel-based TD algorithm is also mentioned.

1. Introduction

In Engel et al. (2003) the use of Gaussian Processes (GPs) for solving the Reinforcement Learning (RL) problem of value estimation was introduced. Since GPs belong to the family of kernel machines, they bring into RL the high, and quickly growing representational flexibility of kernel based representations, allowing them to deal with almost any conceivable object of interest, from text documents and DNA sequence data to probability distributions, trees and graphs, to

mention just a few (see Schölkopf & Smola, 2002, and references therein). Moreover, the use of Bayesian reasoning with GPs allows one to obtain not only value estimates, but also estimates of the *uncertainty* in the value, and this in large and even infinite MDPs.

In this extended abstract we present our approach concerning learning with Gaussian processes and kernel methods. We show how to use GPs and kernels to perform TD algorithms on any input space where a kernel function can be defined. The results reported here are based Engel et al. (2003); Engel et al. (2005); Engel and Mannor (2005) as well as on some ongoing work.

We start by providing a model to the value function based on the discounted return in Section 2. We then describe an online implementation in Section 3. A SARSA based algorithm is briefly mentioned in Section 4. A short summary follows in Section 5.

2. Modeling the Value Via the Discounted Return

A fundamental entity that is of interest in RL is the *discounted return*. Much of the RL literature is concerned with the expected value of this random process, known as the *value function*. This is mainly due to the simplicity of the Bellman equations which govern the behavior of the value function, and because of two provably convergent algorithms (of which many variations exist) that arise from Bellman's equations – value iteration and policy iteration. However, some valuable insights may be gained by considering the discounted return directly and its relation with the value.

A Markov Decision Process (MDP) is a tuple

$(\mathcal{X}, \mathcal{U}, R, p)$ where \mathcal{X} and \mathcal{U} are the state and action spaces, respectively; $R : \mathcal{X} \rightarrow \mathbb{R}$ is the immediate reward, which may be random, in which case $q(\cdot|\mathbf{x})$ denotes the distribution of rewards at the state \mathbf{x} ; and $p : \mathcal{X} \times \mathcal{U} \times \mathcal{X} \rightarrow [0, 1]$ is the transition distribution, which we assume is stationary. Note that we do not assume that \mathcal{X} is Euclidean, or that it is even a vector space. Instead, we will assume that a kernel function, k , is defined on \mathcal{X} . A *stationary policy* $\mu : \mathcal{X} \times \mathcal{U} \rightarrow [0, 1]$ is a mapping from states to action selection probabilities. Given a fixed policy μ , the transition probabilities of the MDP are given by the *policy-dependent state transition probability distribution* $p^\mu(\mathbf{x}'|\mathbf{x}) = \int_{\mathcal{U}} d\mathbf{u} p(\mathbf{x}'|\mathbf{u}, \mathbf{x}) \mu(\mathbf{u}|\mathbf{x})$. The *discounted return* $D(\mathbf{x})$ for a state \mathbf{x} is a random process defined by

$$D(\mathbf{x}) = \sum_{i=0}^{\infty} \gamma^i R(\mathbf{x}_i) | \mathbf{x}_0 = \mathbf{x}, \text{ with } \mathbf{x}_{i+1} \sim p^\mu(\cdot|\mathbf{x}_i). \quad (2.1)$$

Here, $\gamma \in (0, 1)$ is a discount factor that determines the exponential devaluation rate of delayed rewards. Note that the randomness in $D(\mathbf{x}_0)$ for any given state \mathbf{x}_0 is due both to the stochasticity of the sequence of states that follow \mathbf{x}_0 , and to the randomness in the rewards $R(\mathbf{x}_0), R(\mathbf{x}_1), R(\mathbf{x}_2) \dots$. We refer to this as the *intrinsic* randomness of the MDP. Using the stationarity of the MDP we may write

$$D(\mathbf{x}) = R(\mathbf{x}) + \gamma D(\mathbf{x}'), \text{ with } \mathbf{x}' \sim p^\mu(\cdot|\mathbf{x}). \quad (2.2)$$

The equality here marks an equality in the distributions of the two sides of the equation. Let us define the expectation operator \mathbf{E}_μ as the expectation over all possible trajectories and all possible rewards collected in them. This allows us to define the *value function* $V(\mathbf{x})$ as the result of applying this expectation operator to the discounted return $D(\mathbf{x})$. Let $\mathbf{E}_{\mathbf{x}'} V(\mathbf{x}') = \int_{\mathcal{X}} d\mathbf{x}' p^\mu(\mathbf{x}'|\mathbf{x}) V(\mathbf{x}')$, and $\bar{r}(\mathbf{x}) = \int_{\mathbb{R}} dr q(r|\mathbf{x}) r$ be the expected reward at state \mathbf{x} . The value function satisfies the fixed-policy version of the Bellman Equation:

$$V(\mathbf{x}) = \bar{r}(\mathbf{x}) + \gamma \mathbf{E}_{\mathbf{x}'} V(\mathbf{x}') \quad \forall \mathbf{x} \in \mathcal{X}. \quad (2.3)$$

2.1. The Value Model

The recursive definition of the discounted return (2.2) is the basis for our statistical generative model connecting values and rewards. Let us decompose the discounted return D into its mean V and a random, zero-mean residual ΔV ,

$$D(\mathbf{x}) = V(\mathbf{x}) + \Delta V(\mathbf{x}), \quad (2.4)$$

where $V(\mathbf{x}) = \mathbf{E}_\mu D(\mathbf{x})$. In the classic frequentist approach $V(\cdot)$ is no longer random, since it is the true

value function induced by the policy μ . Adopting the Bayesian methodology, we may still view the value $V(\cdot)$ as a random entity by assigning it additional randomness that is due to our subjective uncertainty regarding the MDP's model (p, q) . We do not know what the true functions p and q are, which means that we are also uncertain about the true value function. We choose to model this additional *extrinsic* uncertainty by defining $V(\mathbf{x})$ as a random process indexed by the state variable \mathbf{x} . This decomposition is useful, since it separates the two sources of uncertainty inherent in the discounted return process D : For a known MDP model, V becomes deterministic and the randomness in D is fully attributed to the intrinsic randomness in the state-reward trajectory, modelled by ΔV . On the other hand, in a MDP in which both transitions and rewards are deterministic but otherwise unknown, ΔV becomes deterministic (i.e., identically zero), and the randomness in D is due solely to the extrinsic uncertainty, modelled by V . For a more thorough discussion of intrinsic and extrinsic uncertainties see Mannor et al. (2004).

Substituting Eq. (2.4) into Eq. (2.2) and rearranging we get

$$R(\mathbf{x}) = V(\mathbf{x}) - \gamma V(\mathbf{x}') + N(\mathbf{x}, \mathbf{x}'), \quad \mathbf{x}' \sim p^\mu(\cdot|\mathbf{x}), \quad (2.5)$$

where $N(\mathbf{x}, \mathbf{x}') \stackrel{\text{def}}{=} \Delta V(\mathbf{x}) - \gamma \Delta V(\mathbf{x}')$. Suppose we are provided with a trajectory $\mathbf{x}_0, \mathbf{x}_1, \dots, \mathbf{x}_t$, sampled from the MDP under a policy μ , i.e., from $p_0(\mathbf{x}_0) \prod_{i=1}^t p^\mu(\mathbf{x}_i|\mathbf{x}_{i-1})$, where p_0 is an arbitrary probability distribution for the first state. Let us write our model (2.5) with respect to these samples:

$$R(\mathbf{x}_i) = V(\mathbf{x}_i) - \gamma V(\mathbf{x}_{i+1}) + N(\mathbf{x}_i, \mathbf{x}_{i+1}), \quad i = 0, \dots, t-1. \quad (2.6)$$

Defining the finite-dimensional processes R_t, V_t, N_t and the $t \times (t+1)$ matrix \mathbf{H}_t

$$\begin{aligned} R_t &= (R(\mathbf{x}_0), \dots, R(\mathbf{x}_t))^\top, \\ V_t &= (V(\mathbf{x}_0), \dots, V(\mathbf{x}_t))^\top, \\ N_t &= (N(\mathbf{x}_0, \mathbf{x}_1), \dots, N(\mathbf{x}_{t-1}, \mathbf{x}_t))^\top, \\ \mathbf{H}_t &= \begin{bmatrix} 1 & -\gamma & 0 & \dots & 0 \\ 0 & 1 & -\gamma & \dots & 0 \\ \vdots & & & & \vdots \\ 0 & 0 & \dots & 1 & -\gamma \end{bmatrix}, \end{aligned} \quad (2.7)$$

we may write the equation set (2.6) more concisely as

$$R_{t-1} = \mathbf{H}_t V_t + N_t. \quad (2.8)$$

2.2. The prior

In order to specify a complete probabilistic generative model connecting values and rewards, we need

to define a prior distribution for the value process V and the distribution of the “noise” process N . We impose a Gaussian prior over value functions, i.e., $V \sim \mathcal{N}(0, k(\cdot, \cdot))$, meaning that V is a Gaussian Process (GP) for which, a priori, $\mathbf{E}(V(\mathbf{x})) = 0$ and $\mathbf{E}(V(\mathbf{x})V(\mathbf{x}')) = k(\mathbf{x}, \mathbf{x}')$ for all $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$, where k is a positive-definite kernel function. Therefore, $V_t \sim \mathcal{N}(\mathbf{0}, \mathbf{K}_t)$, where $\mathbf{0}$ is a vector of zeros and $[\mathbf{K}_t]_{i,j} = k(\mathbf{x}_i, \mathbf{x}_j)$. Our choice of kernel function k should reflect our prior beliefs concerning the correlations between the values of states in the domain at hand.

2.3. The posterior

In order to maintain the analytical tractability of the posterior value distribution, we model the residuals $\Delta V_t = (\Delta V(\mathbf{x}_0), \dots, \Delta V(\mathbf{x}_t))^\top$ as a Gaussian process. This means that the distribution of the vector ΔV_t is completely specified by its mean and covariance. Another assumption we make is that each of the residuals $\Delta V(\mathbf{x}_i)$ is generated independently of all the others. This means that, for any $i \neq j$, the random variables $\Delta V(\mathbf{x}_i)$ and $\Delta V(\mathbf{x}_j)$ correspond to two distinct experiments, in which two random trajectories starting from the states \mathbf{x}_i and \mathbf{x}_j , respectively, are generated independently of each other. We are now ready to proceed with the derivation of the distribution of the noise process N_t .

By definition (Eq. 2.4), $\mathbf{E}_\mu[\Delta V(\mathbf{x})] = 0$ for all \mathbf{x} , so we have $\mathbf{E}_\mu[N(\mathbf{x}_i, \mathbf{x}_{i+1})] = 0$. Turning to the covariance, we have $\mathbf{E}_\mu[N(\mathbf{x}_i, \mathbf{x}_{i+1})N(\mathbf{x}_j, \mathbf{x}_{j+1})] = \mathbf{E}_\mu[(\Delta V(\mathbf{x}_i) - \gamma\Delta V(\mathbf{x}_{i+1}))(\Delta V(\mathbf{x}_j) - \gamma\Delta V(\mathbf{x}_{j+1}))]$. According to our assumption regarding the independence of the residuals, for $i \neq j$, $\mathbf{E}_\mu[\Delta V(\mathbf{x}_i)\Delta V(\mathbf{x}_j)] = 0$. In contrast, $\mathbf{E}_\mu[\Delta V(\mathbf{x}_i)^2] = \mathbf{Var}_\mu[D(\mathbf{x}_i)]$ is generally larger than zero, unless both transitions and rewards are deterministic. Denoting $\sigma_i^2 = \mathbf{Var}[D(\mathbf{x}_i)]$, these observations may be summarized into the distribution of ΔV_t : $\Delta V_t \sim \mathcal{N}(\mathbf{0}, \text{diag}(\boldsymbol{\sigma}_t))$ where $\boldsymbol{\sigma}_t = [\sigma_0^2, \sigma_1^2, \dots, \sigma_t^2]^\top$, and $\text{diag}(\cdot)$ denotes a diagonal matrix whose diagonal entries are the components of the argument vector. In order to simplify the subsequent analysis let us assume that, for all $i \in \{1, \dots, t\}$, $\sigma_i = \sigma$, and therefore $\text{diag}(\boldsymbol{\sigma}_t) = \sigma^2 \mathbf{I}$. Since $N_t = \mathbf{H}_t \Delta V_t$, we have $N_t \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}_t)$ with,

$$\begin{aligned} \boldsymbol{\Sigma}_t &= \sigma^2 \mathbf{H}_t \mathbf{H}_t^\top \\ &= \sigma^2 \begin{bmatrix} 1 + \gamma^2 & -\gamma & 0 & \dots & 0 \\ -\gamma & 1 + \gamma^2 & -\gamma & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & -\gamma & 1 + \gamma^2 \end{bmatrix}. \end{aligned}$$

Since both the value prior and the noise are Gaussian, by the Gauss-Markov theorem (Scharf, 1991), so is the posterior distribution of the value conditioned on an observed sequence of rewards $\mathbf{r}_{t-1} = (r_0, \dots, r_{t-1})^\top$. The posterior mean and variance of the value at some point \mathbf{x} are given, respectively, by

$$\begin{aligned} \hat{v}_t(\mathbf{x}) &= \mathbf{k}_t(\mathbf{x})^\top \boldsymbol{\alpha}_t, \\ p_t(\mathbf{x}) &= k(\mathbf{x}, \mathbf{x}) - \mathbf{k}_t(\mathbf{x})^\top \mathbf{C}_t \mathbf{k}_t(\mathbf{x}), \end{aligned} \quad (2.9)$$

$$\text{where } \boldsymbol{\alpha}_t = \mathbf{H}_t^\top (\mathbf{H}_t \mathbf{K}_t \mathbf{H}_t^\top + \boldsymbol{\Sigma}_t)^{-1} \mathbf{r}_{t-1},$$

$$\mathbf{C}_t = \mathbf{H}_t^\top (\mathbf{H}_t \mathbf{K}_t \mathbf{H}_t^\top + \boldsymbol{\Sigma}_t)^{-1} \mathbf{H}_t, \quad (2.10)$$

and $\mathbf{k}_t(\mathbf{x})$ is a vector of size t whose elements are $k(\mathbf{x}_i, \mathbf{x})$.

3. An On-Line Algorithm

Computing the parameters $\boldsymbol{\alpha}_t$ and \mathbf{C}_t of the posterior moments (2.10) is computationally expensive for large samples, due to the need to store and invert a matrix of size $t \times t$. Even when this has been performed, computing the posterior moments for every new query point requires that we multiply two $t \times 1$ vectors for the mean, and compute a $t \times t$ quadratic form for the variance. These computational requirements are prohibitive if we are to compute value estimates on-line, as is usually required of RL algorithms. Engel et al. (2003) used an on-line kernel sparsification algorithm that is based on a view of the kernel as an inner-product in some high dimensional feature space to which raw state vectors are mapped. This sparsification method incrementally constructs a dictionary $\mathcal{D} = \{\tilde{\mathbf{x}}_1, \dots, \tilde{\mathbf{x}}_{|\mathcal{D}|}\}$ of representative states. Upon observing \mathbf{x}_t , the distance between the feature-space image of \mathbf{x}_t and the span of the images of current dictionary members is computed. If the squared distance exceeds some positive threshold ν , \mathbf{x}_t is added to the dictionary, otherwise, it is left out. Determining this squared distance, δ_t , involves solving a simple least-squares problem, whose solution is a $|\mathcal{D}| \times 1$ vector \mathbf{a}_t of optimal approximation coefficients, satisfying

$$\mathbf{a}_t = \tilde{\mathbf{K}}_{t-1}^{-1} \tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t), \quad \delta_t = k_{tt} - \mathbf{a}_t^\top \tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t), \quad (3.11)$$

where $\tilde{\mathbf{k}}_t(\mathbf{x}) = (k(\tilde{\mathbf{x}}_1, \mathbf{x}), \dots, k(\tilde{\mathbf{x}}_{|\mathcal{D}|}, \mathbf{x}))^\top$ is a $|\mathcal{D}_t| \times 1$ vector, and $\tilde{\mathbf{K}}_t = [\tilde{\mathbf{k}}_t(\tilde{\mathbf{x}}_1), \dots, \tilde{\mathbf{k}}_t(\tilde{\mathbf{x}}_{|\mathcal{D}_t}|)]$ a square $|\mathcal{D}_t| \times |\mathcal{D}_t|$, symmetric, positive-definite matrix.

By construction, the dictionary has the property that the feature-space images of all states encountered during learning may be approximated to within a squared error ν by the images of the dictionary members. The threshold ν may be tuned to control the sparsity of the

solution. Sparsification allows kernel expansions, such as those appearing in Eq. 2.10, to be approximated by kernel expansions involving only dictionary members, by using

$$\mathbf{k}_t(\mathbf{x}) \approx \mathbf{A}_t \tilde{\mathbf{k}}_t(\mathbf{x}), \quad \mathbf{K}_t \approx \mathbf{A}_t \tilde{\mathbf{K}}_t \mathbf{A}_t^\top. \quad (3.12)$$

The $t \times |\mathcal{D}_t|$ matrix \mathbf{A}_t contains in its rows the approximation coefficients computed by the sparsification algorithm, i.e., $\mathbf{A}_t = [\mathbf{a}_1, \dots, \mathbf{a}_t]^\top$, with padding zeros placed where necessary, see Engel et al. (2003).

The end result of the sparsification procedure is that the posterior value mean \hat{v}_t and variance p_t may be compactly approximated as follows (compare to Eq. 2.9, 2.10)

$$\begin{aligned} \hat{v}_t(\mathbf{x}) &= \tilde{\mathbf{k}}_t(\mathbf{x})^\top \tilde{\boldsymbol{\alpha}}_t, \\ p_t(\mathbf{x}) &= k(\mathbf{x}, \mathbf{x}) - \tilde{\mathbf{k}}_t(\mathbf{x})^\top \tilde{\mathbf{C}}_t \tilde{\mathbf{k}}_t(\mathbf{x}), \end{aligned} \quad (3.13)$$

$$\text{where } \tilde{\boldsymbol{\alpha}}_t = \tilde{\mathbf{H}}_t^\top \left(\tilde{\mathbf{H}}_t \tilde{\mathbf{K}}_t \tilde{\mathbf{H}}_t^\top + \boldsymbol{\Sigma}_t \right)^{-1} \mathbf{r}_{t-1}$$

$$\tilde{\mathbf{C}}_t = \tilde{\mathbf{H}}_t^\top \left(\tilde{\mathbf{H}}_t \tilde{\mathbf{K}}_t \tilde{\mathbf{H}}_t^\top + \boldsymbol{\Sigma}_t \right)^{-1} \tilde{\mathbf{H}}_t, \quad (3.14)$$

and $\tilde{\mathbf{H}}_t = \mathbf{H}_t \mathbf{A}_t$.

The parameters that the algorithm is required to store and update in order to evaluate the posterior mean and variance are now $\tilde{\boldsymbol{\alpha}}_t$ and $\tilde{\mathbf{C}}_t$, whose dimensions are $|\mathcal{D}_t| \times 1$ and $|\mathcal{D}_t| \times |\mathcal{D}_t|$, respectively. In many cases this results in significant computational savings, both in terms of memory and time, when compared with the exact non-sparse solution.

We omit the lengthy technical derivation of the algorithm as it appears in (Engel et al., 2003; Engel et al., 2005). The main idea is that if the matrix H_t admits a favorable shape (tridiagonal or diagonal), an efficient algorithm for recursive computation of the posterior can be derived. We note that the algorithm is expressed only in terms of kernel function activations, thus the input space \mathcal{X} may be completely arbitrary.

4. Policy Improvement with GPSARSA

SARSA is a fairly straightforward extension of the TD algorithm (Sutton & Barto, 1998), in which state-action values are estimated, thus allowing policy improvement steps to be performed without requiring any additional knowledge on the MDP model. The idea is to use the stationary policy μ being followed in order to define a new, augmented process, the state space of which is $\mathcal{X}' = \mathcal{X} \times \mathcal{U}$, (i.e., the original state space augmented by the action space), maintaining the same reward model. This augmented process is Markovian with transition probabilities $p'(\mathbf{x}', \mathbf{u}' | \mathbf{x}, \mathbf{u}) =$

$p^\mu(\mathbf{x}' | \mathbf{x}) \mu(\mathbf{u}' | \mathbf{x}')$. SARSA is simply the TD algorithm applied to this new process. The same reasoning may be applied to derive a SARSA algorithm from the GP based TD algorithm. All we need is to define a covariance kernel function over state-action pairs, i.e., $k : (\mathcal{X} \times \mathcal{U}) \times (\mathcal{X} \times \mathcal{U}) \rightarrow \mathbb{R}$. Since states and actions are different entities it makes sense to decompose k into a state-kernel k_x and an action-kernel k_u : $k(\mathbf{x}, \mathbf{u}, \mathbf{x}', \mathbf{u}') = k_x(\mathbf{x}, \mathbf{x}') k_u(\mathbf{u}, \mathbf{u}')$. If both k_x and k_u are kernels we know that k is also a legitimate kernel (Schölkopf & Smola, 2002), and just as the state-kernel codes our prior beliefs concerning correlations between the values of different states, so should the action-kernel code our prior beliefs on value correlations between different actions.

All that remains now is to run the GP-based TD learning algorithm on the augmented state-reward sequence, using the new state-action kernel function. Action selection may be performed by ε -greedily choosing the highest ranking action, and slowly decreasing ε toward zero. However, we may run into difficulties trying to find the highest ranking action from a large or even infinite number of possible actions. This may be solved by fast iterative maximization method, such as the quasi-Newton method or conjugate gradients. Ideally, we should design the action kernel in such a way as to provide a closed-form expression for the greedy action (as in Engel et al., 2005).

5. Discussion

The Gaussian Processes approach, combined with the kernelization of the TD algorithm facilitates rich representations. Instead of focusing on Euclidean spaces, one can consider employing RL in essentially any space on which a kernel can be defined. Formally, the kernel is used to provide a prior over the covariance of the Gaussian process. Intuitively, the kernel represents how “close” two states (or actions in the SARSA case) are which allows incorporating domain knowledge into the construction of the algorithm.

Finding “good” kernels is a challenge in kernel methods in general. However, there is no need for the kernel to be “optimal” for the algorithm to work well. All that is needed is for the kernel to be reasonable. In the kernel method community there is a significant body of work on how to create kernels. This includes Fisher Kernels (Jaakkola & Haussler, 1998) for probabilistic models (the space \mathcal{X} itself may be a space of probabilistic models), creating kernels from other kernels (Cristianini & Shawe-Taylor, 2000), etc.

The GP-based temporal difference approach itself has

many advantages. For example, it provides confidence bounds on the value function uncertainty. This can be potentially used for balancing exploration and exploitation. In Engel and Mannor (2005) we took advantage of the availability of confidence intervals to extend the GP-based approach to a setup where multiple agents interact with the same environment. We used the uncertainty estimates to weigh the value functions learned by the different agents. We finally mention that the policy improvement mechanism described above is SARSA based. It would be useful to devise a Q-Learning type algorithm for off-policy learning of the optimal policy; this is left for further study.

References

- Bertsekas, D., & Tsitsiklis, J. (1996). *Neuro-dynamic programming*. Athena Scientific.
- Cristianini, N., & Shawe-Taylor, J. (2000). *An introduction to support vector machines and other kernel-based learning methods*. Cambridge, England: Cambridge University Press.
- Engel, Y., & Mannor, S. (2005). Collaborative temporal difference learning with gaussian processes. *Submitted to the Twenty-Second International Conference on Machine Learning*.
- Engel, Y., Mannor, S., & Meir, R. (2003). Bayes meets Bellman: The Gaussian process approach to temporal difference learning. *Proc. of the 20th International Conference on Machine Learning*.
- Engel, Y., Mannor, S., & Meir, R. (2005). Reinforcement learning with Gaussian processes. *Submitted to the Twenty-Second International Conference on Machine Learning*.
- Jaakkola, T., & Haussler, D. (1998). Exploiting generative models in discriminative classifiers. *Advances in Neural Information Processing Systems 11* (pp. 512–519).
- Mannor, S., Simester, D., Sun, P., & Tsitsiklis, J. (2004). Bias and variance in value function estimation. *Proc. of the 21st International Conference on Machine Learning*.
- Scharf, L. (1991). *Statistical signal processing*. Addison-Wesley.
- Schölkopf, B., & Smola, A. (2002). *Learning with Kernels*. Cambridge, MA: MIT Press.
- Sutton, R., & Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. MIT Press.