Policy Search in Kernel Hilbert Space

J. Andrew Bagnell and Jeff Schneider
Robotics Institute
Carnegie-Mellon University
Pittsburgh, PA 15213
{*dbagnell,schneide*}*@ri.cmu.edu*

Abstract

Much recent work in reinforcement learning and stochastic optimal control has focused on algorithms that search directly through a space of policies rather than building approximate value functions. Policy search has numerous advantages: it does not rely on the Markov assumption, domain knowledge may be encoded in a policy, the policy may require less representational power than a value-function approximation, and stable and convergent algorithms are well-understood. In contrast with value-function methods, however, existing approaches to policy search have heretofore focused entirely on parametric approaches. This places fundamental limits on the kind of policies that can be represented. In this work, we show how policy search (with or without the additional guidance of value-functions) in a Reproducing Kernel Hilbert Space gives a simple and rigorous extension of the technique to non-parametric settings.

In particular, we investigate a new class of algorithms which generalize REINFORCE-style likelihood ratio methods to yield both online and batch techniques that perform gradient search in a function space of policies. Further, we describe the computational tools that allow efficient implementation. Finally, we apply our new techniques towards interesting reinforcement learning problems.

Introduction

Control under uncertainty and reinforcement learning are central problems in artificial intelligence and robotics. The customary approach has been to solve such problems using Bellman recursion methods. Recently, however, methods that instead search directly through a space of parametrized policies have gained currency. Despite the potential disadvantages of policy search, e.g. inability to bootstrap, on-policy restrictions, local minima, and potentially high variance in estimates, the approach has some practical appeal.

We review some the advantages of the direct search approach: While aliasing is potentially very detrimental to both the performance and stability properties of value-function based methods, policy search methods extend easily to accommodate partial observability. The convergence properties are fairly well understood, in contrast with value methods. It also often seems to be the case that a very effective policy may be more easily represented in policy form than as a value-function. Direct policy representation are also at times more appropriate for real-time applications where computing policy outputs must be rapid. Finally, domain knowledge about the problem solution is often easily encodable in the policy form.

Further, frameworks that leverage the best of both value-function and policy search methods are available. A simple, learned, approximate value function, for example, may not be sufficient for control purposes, but may prove quite valuable in reducing the variance of a gradient based policy search algorithm. A number of algorithms merge the properties of the two, including Actor-Critic techniques, VAPS [2], and approaches utilizing reward-shaping [11].

Value-function methods have had at least one significant advantage that has not been overcome. It has long been common practice to use very powerful non-parametric function approximation techniques, including locally-weighted regression and support-vector machines to accurately approximate the value-function. Recently, theoretical results have been obtained as well for these techniques. [12] In contrast, policy search has been phrased in terms of an explicit search over a parametric policy class. This presents a fundamental limitation on the applicability of policy search techniques; although one would hope that in the presence of more data, more refined and complicated policies can be learned, the parametric approach obstructs this.

In this work we show how techniques from the *kernel machine* [7] approach to learning can be extended to the policy search problem. The machinery of Reproducing Kernel Hilbert Spaces allows us to develop a natural generalization of policy search to search through a space of functions. From this we recover a fully non-parametric policy search. Further, we investigate the regularization of policy learning, a notion concomitant to learning in function space, but one that has seen little investigation in the reinforcement learning literature.

We briefly review the aims of reinforcement learning and the essentials of Reproducing Kernel Hilbert Space (RKHS) theory. We then derive kernelized versions of William's REINFORCE algorithm [3] for policy search. We consider both online and batch approaches. We then consider techniques that render the approach computationally tractable, including least-squares approximation in Hilbert space, and data-structures tricks from computational geometry. Finally, we consider the application of the techniques to two domains: the well-known dynamics problem *Mountain Car*, and a financial portfolio optimization considered in [12].

1 Preliminaries

1.1 Reinforcement Learning

A stochastic control problem consists of paths ξ (also called system trajectories) in a space Ξ, a distribution over path-space, $p(\xi)$, that is a function of a sequence of controls $< u_t >_{t\in\{0,..,T\}}$, indexed by time, from a space $\mathcal{U}$. It is useful to limit discussion to Partially Observed Markov Decision Problems [8], in which there are state-variables x_t of the system that compose a path ξ and render the past and future independent. In a POMDP, $p(\xi)$ is defined by an initial state and next state transition probabilities $p(x_t|x_{t-1}, u_{t-1})$. We also typically have a sequence of outputs (observations) $< y_t >_{t\in\{0,..,T\}}$ each of which is measurable with respect to x_t. A controller, μ, usually parameterized by θ, is a feedback-type strategy that maps the history of observations $< y_t >_{0,..,\tau}$ to a distribution over controls u_τ.

The goal in a control problem is to maximize the expected reinforcement $J_\theta = \sum_\Xi p_\theta(\xi)r(\xi)$ with respect to θ. The reward function on paths in a POMDP is additive in time (or in the infinite time case, discounted or averaged), and a function $R(x)$ of state, although unlike value-function methods, the algorithms discussed here are not necessarily predicated on that assumption.

Reinforcement learning (RL) is adaptive stochastic control, where an agent attempts to solve for an optimal controller without an apriori model of the dynamics. Rather, the reinforcement learner optimizes its control strategy using sample trajectories from the dynamics. (Possibly using many sophisticated techniques including internal approximate models.) In this work, our algorithms will be adaptable to both situations.

1.2 Reproducing Kernel Hilbert Spaces

Kernel methods, in their various incarnations (e.g. Gaussian Processes, Support Vector machines) have recently become a preferred approach to non-parametric machine learning and statistics. They enjoy this status because kernel methods form a kind of point-free linear-algebra, where inner product operations are performed by the kernel. As such, natural operations, like least-squares approximation in functions space, can be described by simple algebraic methods that naturally generalize parametric linear counterparts.

By kernel here we mean a symmetric function $k(\cdot, \cdot)$ of two arguments that produces a real scalar. Further, we must assume that the kernel is positive-definite in the sense that when applied to any two lists of inputs $\{a\}_i$ and $\{b\}_j$ the matrix $K_{i,j} = k(a_i, b_j)$ is a positive definite one. (i.e. has all positive eigenvalues) Kernel methods can be thought of as implementing a generalization of the notion of similarity defined by the standard inner-product, where the kernel computes the similarity between its inputs. Kernels as described can also be thought of as computing the standard inner product in a linear space formed by the kernel functions. For any object a in the domain of the kernel, the function $k(a, \cdot)$ can be thought of as a vector in a linear space.

We turn this linear space into a Hilbert space as follows. First we equip it with an inner product: for two vectors in the linear space, $f = \sum_i \alpha_i k(a_i, \cdot)$ and $g = \sum_i \beta_i k(b_i, \cdot)$ the inner product $< f, g >_{k(\cdot,\cdot)}$is $\sum_{i,j} \alpha_i\beta_j k(a_i, b_j)$. That this is a proper inner product may be easily checked. Finally, technical issues from analysis require us to complete the space by adding the limits of all Cauchy sequences. We denote the space $H_{k(\cdot,\cdot)}$. We note a very useful property of the Hilbert space we have constructed is the *reproducing* property. That is for any $f \in H_{k(\cdot,\cdot)}$, the kernel reproduces the function: $< f, k(x, \cdot) >_{k(\cdot,\cdot)}= f(x)$.

2 Search in RKHS

We will consider stochastic policies $\mu_f(u|y)$ that are defined by a function $f \in H_{k(\cdot,\cdot)}$ where the kernel $k(\cdot, \cdot)$ is defined on the observation space. That is, our policies will take an observation y and compute a probability distribution over controls. The elegant representer theorem, given in sufficiently general form by [13], tells that to maximize the empirical reinforcement for sample paths ξ_i:

$$J_{emp}(f) = \frac{1}{n}\sum_i r(\xi_i)$$

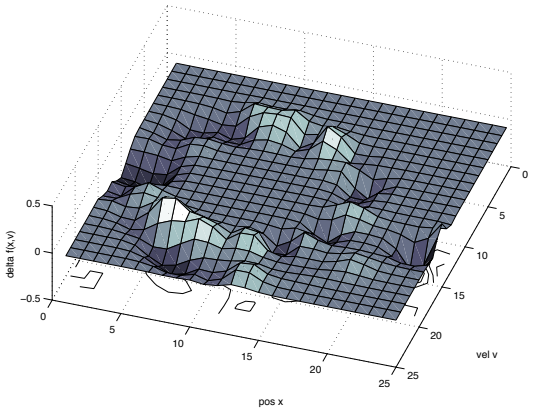

Figure 1: The figure shows a stochastic gradient update calculated from a single sample path in the *Mountain-Car* problem. Notice the gradient is a function on the state-space composed of kernels (here Gaussians) at points along the sample trajectory.

we need only consider functions in the span of kernels at states that actually occurred in the sample trajectories. This result is maintained (actually strengthened to uniqueness) by minimizing the risk regularized by the norm of the function $< f, f >_{k(\cdot,\cdot)}$. This result is very powerful, and provides the rigorous basis for us to extend policy search to non-parametric policies.

2.1 Stochastic Gradient Methods

One general policy-search approach in reinforcement learning is to perform stochastic gradient ascent on J using sampled paths. The REINFORCE class of algorithms use the likelihood ratio method to approximate the gradient using the identity:

$$\nabla J = \sum_{\Xi} \nabla p(\xi) r(\xi) = \sum_{\Xi} p(\xi) \frac{\nabla p(\xi)}{p(\xi)} r(\xi)$$

Here we must consider stochastic gradient descent in the function space $H_{k(\cdot,\cdot)}$. Functional gradient methods have been considered for classification and regression in [9] and [7]. In reinforcement learning the distribution of samples changes with the choice of function defining the policy, so we must consider the derivative of this distribution, not simply the derivative of the empirical cost function as in [7]. This reflects a fundamental difference in the formulation of reinforcement learning as compared to classification or regression. Although the functional gradient may in fact be very complicated, as with the maximizer of J_{emp}, our best approximation by sampling will only depend on kernels centered at points in our sampled trajectories.

First, we define the gradient $\nabla_f F(f)$ of a functional F on the Hilbert space $H_{k(\cdot,\cdot)}$ as the linear deviation of the functional for a perturbation (ϵg) :

$$F[f + \epsilon g] = F[f] + \epsilon < \nabla_f F, g > +O(\epsilon^2)$$

We note the important result that the reproducing property immediately implies that $\nabla_f h(y_i) = k(y_i, \cdot)$. That is, the functional gradient of a function in the Hilbert space applied at a point is simply the kernel itself at that point.

To perform stochastic gradient descent, we "roll-out" a trajectory using the current policy (defined by f) and then compute:

$$(\nabla_f J_f)_{emp} = \frac{\nabla_f p_f(\xi_i)}{p_f(\xi_i)} r(\xi_i)$$

Gradient ascent takes a step in this direction. Fortunately, it proves easy to compute $\frac{\nabla_f p_f(\xi_i)}{p_f(\xi_i)}$. In particular, the derivative depends only on the controller probabilities and can be written as:

$$\sum_t \frac{\nabla_f \mu_f(u_t|y_t)}{\mu_f(u_t|y_t)}$$

This is simply a consequence of the equivalent of the product-rule for functional derivatives. All terms, for instance $p(x_{t+1}|x_t, u_t)$, that are independent of the parameterizing function cancel out.

Let us consider the simplest example of a functional policy where μ is a distribution over two actions (u in {-1,1}), f is a one dimensional function and we compute $\mu(u = 1|y) = 1/(1 + e^{f(y)})$ for our stochastic policy. These policies provides simple to visualize examples (and is the form of policy we use in our experimental section). Further, the sigmoidal policy easily generalizes to non-binary actions as $\mu(u|y) = e^{f_u(y)}/(\sum_i e^{f_i(y)})$ where we provide a different function f_i for each action.

We need one more result from functional analysis to complete the computation of the gradient. It is the functional equivalent of the chain rule:

$$\nabla_f g(F[h]) = \partial_x g(x)|_{F(h)} \nabla_f F|_h$$

Given this, and the preceding notes about the reproducing property, one can see the important result that:

$$\nabla_f \mu_f(u_t|y_t)/\mu_f(u_t|y_t) = -sign(u_t)(1-\mu_f(u_t|y_t))k(y_t, \cdot) \quad (1)$$

That is, for each step along the trajectory we get a kernel basis center on that point with a coefficient related to the action chosen, and the appropriate probability of that action. To increase the reinforcement functional, we simply

add to the current f the product of a learning rate and the computed stochastic gradient.

Adding a regularization term that penalizes functions by their norm $< f, f >_{k(\cdot,\cdot)}$ amounts to, in the stochastic gradient framework, shrinking the coefficients of all previous kernels by multiplying them by some constant $c \in (0, 1)$. We can see this by noting that adding $-\frac{\lambda}{2} < f, f >$ to our reward function and then taking the functional gradient gives us back $-\lambda f$. The update is then

$$f_{t+1} := f_t + \alpha(-\lambda f_t + r(\xi)\frac{\nabla_f \mu_f(u_t|y_t)}{\mu_f(u_t|y_t)}) \qquad (2)$$

where $r(\xi)$ is the total reinforcement for the current trajectory.

2.2 Metric Computation

It is interesting to observe that the functional gradient differs (and is simpler!) from that which one would compute [7] by simply adding kernels at each point along the trajectory and computing partial derivatives with respect to the coefficients on these kernels. Such an algorithm has no obvious convergence properties as it is continuously adding basis functions. Our algorithm, in contrast, has a clear interpretation as searching in a compact space of functions.

Further, this naive algorithm of adding kernels and then computing partial derivatives ignores the interaction the different kernel basis functions have with each other, which the function space approach takes explicitly into account. It can also be seen that this naive algorithm of simply adding kernels and computing the gradient has an $O(n^3)$ scaling as well, as it requires computing the inverse of the matrix $K_{ij} = k(y_i, y_j)$. This is because the kernel defines a different Riemannian metric than the Euclidean one on the space of coefficients. This metric can be considered a kind of prior on policies– implying policies should be similar where the kernel function is large. The regularization term enforces just this similarity.

The notion of metric begs the question of whether there is, in some sense, a canonical metric. Recent work [1] [6] has consider what such a metric should be, concluding that the natural metric should be related to the distribution of paths induced by the controller μ. A particularly well-justified choice is a metric derived from the Fisher information matrix on the distribution of paths. Future work will investigate the benefits of approximating the inverse of the Fisher operator $E_{p(\xi)}[Outer(\frac{\nabla_f p_f(\xi)}{p_f(\xi)}, \frac{\nabla_f p_f(\xi)}{p_f(\xi)})]$ (where Outer designates the outer product) in RKHS to compute this natural gradient direction.

2.3 On-line implementation

For time additive cost functions (including average or discounted reward), one can make another addition to make the procedure fully online. This is the observation that controls at a given time don't influence the past of that trajectory. It lowers the variance of the estimate to include only the future reward of the trajectory. Further, by allowing the policy to shift during the trajectory we can make our algorithm a fully online one like REINFORCE. We present the non-parametric version of REINFORCE (rather GPOMDP since we give an average reward infinite-trajectory length formulation) below.

Algorithm 1 (Online Non-parametric Policy Search)

- *t=0*
- *Begin with an empty list of kernels k.*
- *Begin with an empty list of eligibility coefficients e.*
- *Begin with an empty list of gradient coefficients* Δ.
- *Repeat until done:*
  - *Choose an action according to* $\mu_f(u|y_{t+1})$.
  - *Observe reward* R_t
  - *Shrink each eligibility constant by* γ*– (a discount factor or bias/variance tradeoff parameter)*
  - *Add a new kernel* $k(y_t, \cdot)$ *to the list*
  - *Add the corresponding coefficient of* $k(y_t, \cdot)$ *from* $\frac{\nabla_f \mu_f(u_t|y_t)}{\mu_f(u_t|y_t)}$ *computed as in equation (1) to the eligibility list as* e_i
  - *Set all* Δ_i *coefficients to* $\Delta_i(1-1/(t+1))+R_t e_i/(t+1)$
- *Return kernel list and matching coefficient list* Δ

3 Multi-sample version

In practice it may be valuable to get more accurate gradient estimates before changing our policy. This may potentially be more stable, allow us to apply more sophisticated step length heuristics, and will lower the computational burden. To do so, we consider batch processing a functional gradient computed from a reasonably large number of sample trajectories.

3.1 Sparse Representation

There are difficulties in manipulating functional gradients consisting of large numbers of kernel basis. To make matters more tractable, we consider approximations that compress the functional gradient to a more parsimonious representation. Such an approach is composed of two pieces. The first chooses a basis set, and the second computes the closest approximation given this set. (Algorithms may interleave these steps as well.) Although surely an area of interesting research, we have found that quite trivial strategies may suffice for basis selection. The simplest of all,

random subset, seems to be adequate for at least some problems. Given the basis, we simply compute the least squares approximation to the gradient g in the RKHS:

$$\min_{\tilde{g}} ||\tilde{g} - g||_{H_k(\cdot,\cdot)}$$

where $\tilde{g}$ lies in the span of our reduced basis. The RKHS approach makes such computations very simple.

In coefficient form, the functions are $\tilde{g} = \sum_i \alpha_i k(\tilde{y}_i, \cdot)$ with the true gradient lying in the span of a different set of kernels $g = \sum_j \beta_j k(y_j, \cdot)$. This isn't necessary for the general presentation, but suffices for our use. Then

$$< \tilde{g}-g, \tilde{g}-g >_{k(\cdot,\cdot)} = \|\tilde{g}\|_{k(\cdot,\cdot)} - 2 < \tilde{g}, g >_{k(\cdot,\cdot)} + \|g\|_{k(\cdot,\cdot)}$$

The final term is irrelevant to maximize with respect to $\tilde{g}$ so we have in terms of the kernel representation:

$$< \sum_i \alpha_i k(\tilde{y}_i, \cdot), \sum_i \alpha_i k(\tilde{y}_i, \cdot) >_{k(\cdot,\cdot)} - 2 < \sum_i \alpha_i k(\tilde{y}_i, \cdot), \sum_j \beta_j k(y_j, \cdot) >_{k(\cdot,\cdot)}$$

To maximize this with respect to the α parameters, we compute the gradient and set it to zero. Using this, and the definition of the kernel inner product, we get in vector notation:

$$\alpha^T K^{\alpha,\alpha} - \beta^T K^{\beta,\alpha} = 0$$

where $K^{\alpha,\alpha}$ is the kernel matrix for the $\tilde{y}_i$ and $K^{\beta,\alpha}$ the kernel matrices for the $\tilde{y}_i$ with the y_j. (i.e. $K^{\beta,\alpha}_{j,i} = k(y_j, y_i)$). By inspection, the second term is simply the function g applied at the points in our approximation. Thus the final answer is simply:$\alpha_i = (K^{\alpha,\alpha})^{-1} g(y_i)$

The matrix inverse is only the size of our reduced basis, so this may be computationally tractable. The evaluation of the kernel function at each of the points may remain quite non-trivial.

3.2 Efficient computation

The central computational difficulty of the approach we outline in this work is the evaluation of the functional gradient. Although the compression method above helps lighten this burden, we are still forced to do work linear in the number of observations seen per batch at least once when we evaluate $g(x_i)$ for the compression. In general, we would like to keep as faithful a recreation of the kernel gradient as possible, but doing so will generally make the computation more expensive.

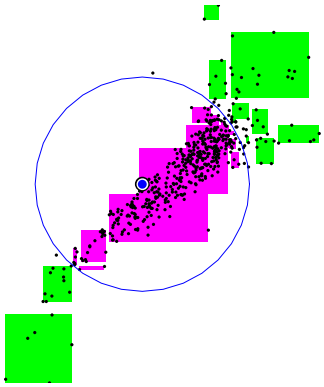

Figure 2: This image depicts a KD-TREE being used to compute the gradient evaluation at a point. (center) Parts of the computation get cutoff (outside the range depicted in the circle) as it can be shown they must have negligible effect on the query. This computational reduction makes the algorithms feasible for very large data-sets.

We may make the gradient evaluations tractable, even when potentially millions of kernel basis spangle our observation space, by noting that the evaluation can be viewed a kind of weighted n-body problem.[5] As such, data-structure and algorithm tricks from physics and computational geometry have the potential to render the approach tractable. For our research, we used the classical *kd-tree*. [10] In effect, we create a tree that divides the samples in the observation space into regions.

Whenever called upon to make an evaluation, we descend into the tree adding samples from regions that are near our evaluation point. Throughout the tree, we cache the weight of all the kernels that fall underneath it. In such a way, we can compute whether at any given query a node in the tree (and all the observations it contains) can have an effect on our query. We compute the maximum and minimum impact, and if the difference between the two is small enough that it will have negligible influence on our prediction, we simply use the average value. Whenever this cannot be ruled out, we recurse further into the tree.

Geometric data-structures have seen wide-spread use in certain areas of machine learning– like density estimation and locally-weighted learning. [10] We believe that this approach has great potential in alleviating the computational burden of kernel methods in more general contexts than the reinforcement learning application we discuss here. For example, kernel logistic regression and kernel least squares classification both may be solved as special cases of the algorithm we present here and may utilize the same data structures.

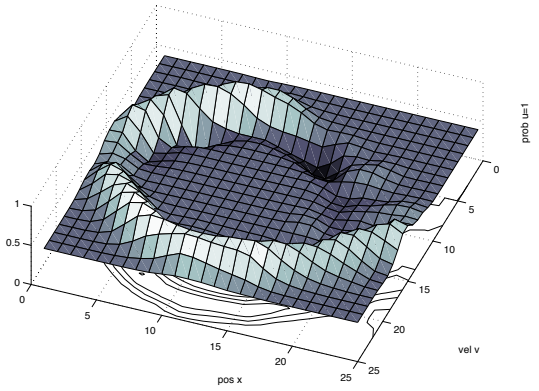

Figure 3: Here we depict a learned policy for the mountain car problem. High points represent high probability of applying forward control and low points represent high probability of applying backward control. Note how our kernel method concentrates representational power where it is required. The regions where the function is flat correspond to regions where f is regularized to 0 as the car spends no time there.

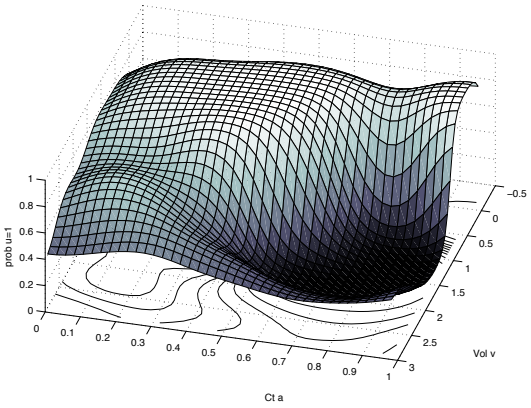

Figure 4: Here we depict a policy learned on the portfolio problem. The metric here is the modified Sharpe ratio metric we describe in the text. High regions indict where the trader will increase its market state, and low regions the reverse. The policy has the property that at high volatilities the trader prefers the riskless-asset.

4 Experiments

4.1 Mountain-Car

We provide here two experiments which we hope can provide further insight into the methods we have developed. First, we applied our method to the well-understood *Mountain Car* domain. [10] This is a simple dynamics problem, where the goal is to get an under-powered car up an incline. The problem is interesting in that it requires "non-minimum phase" behavior– that is, we must retreat from the goal before we can reach it. We use the standard dynamic equations, except that to make the problem somewhat more difficult; we count going out the back of the state-space as a penalty of -1. A time-discounted reward of $+1$ is applied for making it to the top of the hill.

Mountain car is a valuable domain because much intuition can be gained by visualizing solutions. In Figure (1) we show a stochastic gradient computed during the stochastic gradient ascent algorithm. It is noticeably jagged and only partially agrees with the true gradient at that point in policy space.

The learned controller for the *Mountain car* problem achieves near- optimal performance for the standard starting point within a small number of gradient steps. A typical number for our second algorithm is 5 steps. In Figure (3) we illustrate a policy during the process of learning control. This figure demonstrates the kernel approach concentrating its representational power where the policy actually spends time. In this example, the car starts at about (x=17,v=5). It follows the trench of low probability backwards and then begins going forward, from whence it follows the ridge of high probability of the forward action all the way to the goal.

4.2 Portfolio Optimization

Another valuable validation experiment is comparing our work to the motivating application of [12]. This paper is the perhaps the closest in spirit to our own work, as it develops some theory for non-parametric methods, albeit in the value-function framework. As such, the problem they consider, a financial portfolio strategy problem seemed quite appropriate to consider.

In essence, the problem considers an investor trying to decide what fraction of their assets at each time step to devote to a stock market S_t and to a riskless money-market. As in [12] we consider discrete investment levels in the stock $A = \{0.0, 0.1, \ldots 1.0\}$. The market is assumed to move according to an Itô-type stochastic differential equation:

$$\begin{aligned} dS_t &= \mu S_t dt + \sqrt{v_t} S_t dB_t \\ dv_t &= \phi(\hat{v} - v_t) S_t dt + v_t \rho d\tilde{B}_t \end{aligned}$$

where dB_t and $d\tilde{B}_t$ represent independent Brownian motions. The first equation describes the movement in the market price of the stock. This is a standard equation describing a geometric Brownian motion with a varying volatility. The second equation describes the underlying volatility of the stock price, and can be seen to be mean-reverting with noise. The parameters are $\mu = 1.03$, $\hat{v} = 0.3$, $\phi = 10$ and $\rho = 5.0$.

It is assumed that the whole system is mean-adjusted so that the riskless asset provides zero interest. Given, this the reward to the system can be defined as the wealth of the investor at the end of the period relative to their initial wealth. In this case, we consider a risk averse investor with a concave utility function– in this case $\log$. This is convenient

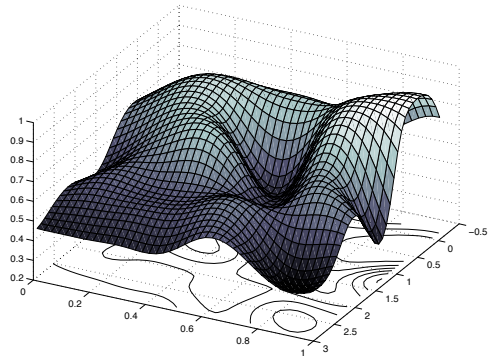

Figure 5: Another policy is shown here. This one corresponds to the simple log-reward maximization in the middle of the learning processes. This also outperforms the best fixed strategy.

for dynamic programming solutions as it makes the reward look additive:

$$\sum_t log(1 + A_t \frac{(S_{t+1} - S_t)}{S_t})$$

for a sufficiently small time discretization.

Unfortunately, this problem as described in [12] isn't a true sequential decision making problem. Instead, the agent can simply chose their investment level at each time independently to maximize their one step gain, as they have no impact on the market. To make the problem more interesting, we have modified it so that decision are instead required to be small modifications in the portfolio. This is a very reasonable restriction as it allows us to represent both capital limitations of the investor and the problem of "moving the market" that happens with large position changes in somewhat illiquid markets. Perhaps most importantly, this restores the sequential decision process to the problem, as the agent is forced to reason about how changing its position now will affect it further in time.

It is difficult to make quantitative comparison with [12] as we have made the problem significantly more challenging, and because only a fairly small sample set is represented there. Here we show performance over a two-month window, against the popular "buy-and-hold" strategy where one invests completely in the risky asset. With even a small number of gradient steps, our algorithm outperforms this baseline. We present results comparing a modified Sharpe ratio of each strategy. The Sharpe ratio, a common performance measure in the financial literature, divides our measure of performance by the average volatility (std. dev.) over the time period in consideration. This has the advantage of penalizing "luck" on the part of the investor due to high volatility swings in the risky asset. For time periods with a net loss, it is not a sensible metric to interpret. We do not have any wish to down-weight high-volatility investments that cost us. That is, it is better that our metric down-weight good luck, but not down-weight bad luck. There are many cost-functions that implement this– our simple version which we label as Sharpe ratio in Figure (7) simply uses the mean volatility whenever the investment amounts to a net loss. One nice aspect of the policy search algorithms is that they are able to directly optimize such path-dependent reinforcement functions.

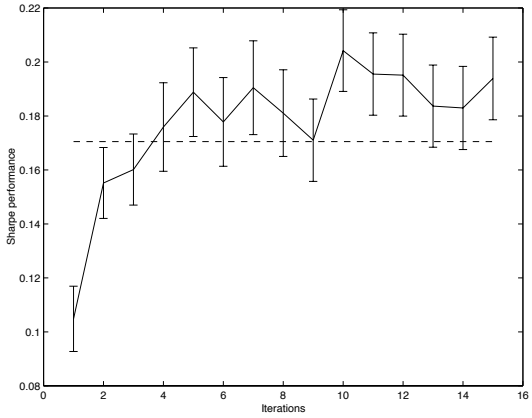

Figure 7: This graph illustrates the performance of our algorithm learning the portfolio strategy as a function of iterations. Error bars are negligible on the baseline performance of the best stationary strategy.

We also visualize the resulting policies for the portfolio optimization problem. The policies here can depend on the full state-variables. Here we a slice of the policy allowing the the control (A_t) and volatility (v_t) to vary. We are unable to declare that the details of the policy are in any way obvious, but it does have the intuitively gratifying property that for high volatilities, it prefers the riskless asset. (Figure 6)

5 Extensions and Conclusions

For the sake of clarity, we explicitly kept to "fully kernelized" algorithms; that is, without any parametric part. In reality, of course, this may be foolish– if a good parametric guess is available it should be added to the function f in defining the probability of actions. This kind of semi-parametric type approach gives us the best of both worlds– combining the domain knowledge advantage that parametric policy search brings with the flexibility of the non-parametric approach.

Our current implementation is rather naive in that the number of kernels either grows linearly in the number of sample steps we observe, or is sub-sampled by trivial algorithms. Larger problems will presumably require sparse, greedy choices of basis for approximation. This is fruitful research because we must consider approximations that preserve kernels where the current policy spends time– that

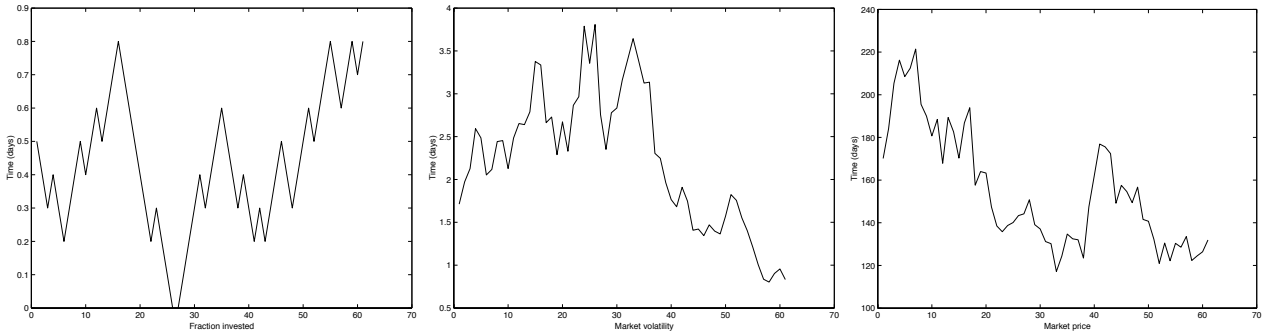

Figure 6: The above images depict trajectories encountered while learning. Note that when market volatility is high, the trader lowers the fraction invested in it to reduce its risk.

is, our approximation accuracy must be considered as a function of the paths that are induced by the controller.

A number of generalizations are natural for the non-parametric approach we have described here. For instance, algorithms that re-use (importance weighted) old trajectories [14], can be easily derived by simply changing the cost function to reflect the probability of these paths. If we can bear the computational burden, these trajectories can lower the experience cost of the algorithms. Next, simple modifications allow us to consider reinforcements that depend on both control and next observation, allowing us to guide policy search by simultaneously learning value-function approximations [2] [11]. The algorithms described here also can be changed to be more appropriate to the average reward setting in a number of ways.

In summary, we have described a simple generalization of likelihood-ratio type policy search to allow search over Reproducing Kernel Hilbert Function Spaces. Our hope is that we can effectively apply kernel policy search to problems where intuition fails us in coming up with an appropriate parametric policy space, yet we are able to define a natural notion of similarity by means of a kernel. We believe many interesting problems will have this property.

References

[1] J. Bagnell and J. Schneider. Covariant policy search. In *to appear in International Joint Conference on Artificial Intelligence*, 2003.

[2] L. Baird and A. Moore. Gradient descent for general reinforcement learning. In *Neural Information Processing Systems 11*, 1998.

[3] J. Baxter, L. Weaver, and P. Bartlett. Direct-gradient-based reinforcement learning I: Gradient estimation algorithms. Technical report, Computer Sciences Lab, Australian National University, 1999.

[4] D. Goldberg, V. Cicirello, M.B. Dias, R. Simmons, S. Smith, T. Smith, and A. Stentz. A distributed layered architecture for mobile robot coordination: Application to space exploration. In *Proceedings of the 3rd International NASA Workshop on Planning and Scheduling for Space*, 2002.

[5] Alexander Gray and Andrew Moore. N-body problems in statistical learning. In Todd K. Leen and Thomas G. Dietterich, editors, *Advances in Neural Information Processing Systems*. MIT Press, 2001.

[6] S. Kakade. A natural policy gradient. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems 14*, Cambridge, MA, 2002. MIT Press.

[7] J. Kivinen, A. J. Smola, and R. C. Williamson. Online learning with kernels. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems 14*, Cambridge, MA, 2002. MIT Press.

[8] M. Littman. *Algorithms for Sequential Decision Making*. PhD thesis, Brown University, 1996.

[9] L. Mason, J.Baxter, P. Bartlett, and M. Frean. *Functional gradient techniques for combining hypotheses*. MIT Press, 1999.

[10] A. Moore. *Efficient Memory-Based Learning for Robot Control*. PhD thesis, University of Cambridge, November 1990.

[11] A. Ng, D. Harada, and S. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *International Conference on Machine Learning*, 1999.

[12] D. Ormoneit and P. Glynn. Kernel-based reinforcement learning in average cost problems: An application to optimal portfolio choice. In *Advances in Neural Information Processing Systems 13*, Cambridge, MA, 2001. MIT Press.

[13] B. Schoelkopf, R. Herbrich, A. J. Smola, and R. C. Williamson. A generalized representer theorem: Nc-tr-00-081. Technical report, NeuroCOLT Technical Report, 2000.

[14] C. R. Shelton. *Importance Sampling for Reinforcement Learning with Multiple Objectives*. PhD thesis, Massachusetts Institute of Technology, 2001.