Sigma Point Policy Iteration

Michael Bowling and Alborz Geramifard
Department of Computing Science
University of Alberta
Edmonton, AB T6G 2E8
{bowling,alborz}@cs.ualberta.ca

David Wingate
Computer Science and Engineering
University of Michigan
Ann Arbor, MI 48109
wingated@umich.edu

ABSTRACT

In reinforcement learning, least-squares temporal difference methods (e.g., LSTD and LSPI) are effective, data-efficient techniques for policy evaluation and control with linear value function approximation. These algorithms rely on policy-dependent expectations of the transition and reward functions, which require all experience to be remembered and iterated over for each new policy evaluated. We propose to summarize experience with a compact policy-independent Gaussian model. We show how this policy-independent model can be transformed into a policy-dependent form and used to perform policy evaluation. Because closed-form transformations are rarely available, we introduce an efficient sigma point approximation. We show that the resulting Sigma-Point Policy Iteration algorithm (SPPI) is mathematically equivalent to LSPI for tabular representations and empirically demonstrate comparable performance for approximate representations. However, the experience does not need to be saved or replayed, meaning that for even moderate amounts of experience, SPPI is an order of magnitude faster than LSPI.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning

General Terms

Algorithms

Keywords

Reinforcement learning, least-squares, policy iteration

1. INTRODUCTION

Linear least-squares value function approximation has shown to be a promising tool for reinforcement learning. Linear methods are well understood, easily analyzed, and have attractive theoretical properties. For these reasons, many successful applications of reinforcement learning have opted to use some sort of linear architecture possibly in a nonlinear feature space, to avoid the theoretical quagmire of general nonlinear function approximation.

There has been a steady progression of ideas in value function approximation with linear architectures. The original ideas began with the TD algorithm [8], which was shown to be convergent under a linear approximation of the value function. While TD is easy to implement, it is not data efficient: each experience tuple is used once to update the value function weights and then discarded. The TD approach was extended to least-squares TD (LSTD) [2], which summarizes experience tuples and finds the best weight vector given a whole stream of experience. Although more data efficient, LSTD's experience summary is specific to the policy being evaluated, and a new policy requires gathering more data. Least-squares policy iteration (LSPI) [6] further extends LSTD by creating a fully off-policy algorithm capable of repeatedly improving a policy without gathering any further data. The main drawback is that the algorithm must store all past experience tuples and "replay" them to update key statistics whenever the policy is changed. Thus, while TD is guaranteed to converge and is policy-independent, it is not data efficient. Conversely, LSPI is data-efficient, but policy-dependent.

We propose a new algorithm which takes the next step. We maintain the data efficiency of LSPI, but do not require further data after improving the policy, and do not require storing or replaying past experience. The key contribution is a general method of transforming a policy-independent model into the policy-dependent statistics needed for use in calculating the linear least-squares approximation of the value function.

We begin by defining a *Gaussian MDP*, which is a compact, policy-independent model of the system. We then introduce a policy iteration algorithm where the critical step involves transforming this policy-independent model into policy-dependent statistics. The step relies on two critical (although reasonable and common) assumptions about the features used by the approximator. Since the needed transformation is often nonlinear, we present a principled and efficient approximation based on sigma-points — the resulting algorithm is named "Sigma Point Policy Iteration" (SPPI). We conclude with empirical results demonstrating that SPPI achieves comparable performance to LSPI, but with an order of magnitude less computation. In addition we note that SPPI is far more amenable to online control, can easily and efficiently make use of new data, and can naturally incorporate prior knowledge.

2. BACKGROUND

Reinforcement learning is an approach to sequential decision making in an unknown environment by learning from past interactions with that environment [9]. In this paper we are specifically interested in learning in Markov decision processes (MDPs). An MDP is a tuple, $(\mathcal{S}, \mathcal{A}, \mathcal{P}^a_{ss'}, \mathcal{R}^a_{ss'}, \gamma)$, where $\mathcal{S}$ is a set of states, $\mathcal{A}$ is a set of actions, $\mathcal{P}^a_{ss'}$ is the probability of reaching state s' after taking action a in state s, $\mathcal{R}^a_{ss'}$ is the reward received when that

Cite as: Sigma Point Policy Iteration, M. Bowling, A. Geramifard and D. Wingate, *Proc. of 7th Int. Conf. on Autonomous Agents and Multi-agent Systems (AAMAS 2008)*, Padgham, Parkes, Müller and Parsons (eds.),May,12-16.,2008,Estoril,Portugal,pp.379-386.

transition occurs, and $\gamma \in [0, 1]$ is a discount rate parameter[1]. A trajectory of experience is a sequence $s_0, a_0, r_1, s_1, a_1, r_2, s_2, \ldots$, where the agent in s_0 takes action a_0 and receives reward r_1 while transitioning to s_1 before taking a_1, *etc.*. The agent's actions are assumed to come from a policy π, where $\pi(s, a)$ is the probability of selecting action a from state s.

We are interested in the *control* problem: find a policy that maximizes the expected sum of future discounted rewards. Formally,

$$\pi^* = \underset{\pi}{\operatorname{argmax}} E_\pi \left[\sum_{t=1}^{\infty} \gamma^{t-1} r_t \middle| s_0, \right]. \quad (1)$$

It is often convenient to look at a policy's value function. The state-action value for a given policy is the expected sum of future discounted rewards if the agent started from a particular state taking a particular action,

$$V^\pi(s, a) = E_\pi \left[\sum_{t=1}^{\infty} \gamma^{t-1} r_t \middle| s_0 = s, a_0 = a, \right].$$

The optimal policy maximizes this function at every state for some action. We can write the value function recursively as,

$$V^\pi(s, a) = E_\pi \left[r_{t+1} + \gamma V^\pi(s_{t+1}, a_{t+1}) | s_t = s, a_t = a \right].$$

Notice that experience tuples $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$ can be seen as essentially samples of this expectation. For a particular value function V let the TD error at time t be defined as,

$$\delta_t(V) = r_{t+1} + \gamma V(s_{t+1}) - V(s_t). \quad (2)$$

Then, $E_\pi \left[\delta(V^\pi) \right] = 0$, that is, the mean TD error for the policy's true value function given experience following that policy must be zero. Given a policy π finding or approximating that policy's value function is know as *policy evaluation* and, as we will see, is an important step to finding a good control policy.

Often the number of state-action pairs is too large to represent V^π as a table of values. A common approach is to approximate V^π using a linear function approximator. In particular, suppose we have a function $\phi : \mathcal{S} \times \mathcal{A} \rightarrow \Re^m$, which gives a feature representation of the state-action space. We are interested in approximate value functions of the form $V_w(s, a) = \phi(s, a)^T w$, where $w \in \Re^m$ are the parameters of the value function. Because the policy's true value function is probably not in our space of linear functions, we want to find a set of parameters that approximates the true function. A common approach is to use the observed TD error on sample trajectories of experience to guide the approximation. We will look at two such approaches, TD and LSTD, and show how in each case finding an approximate value function can be used to find a good policy.

2.1 Temporal Difference Learning

Temporal difference learning (TD) is the traditional technique for computing an approximate value function of a policy through interaction with the process. The basic idea of one-step TD learning, also known as TD(0), is to adjust a state's predicted value to reduce the observed TD error. Given some new experience tuple $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$, the update with linear function approximation is,

$$w_{t+1} = w_t + \alpha_t u_t(w_t),$$

where α_t is a learning rate parameter and,

$$u_t(w) = \phi(s_t, a_t) \left(r_{t+1} + [\gamma \phi(s_{t+1}, a_{t+1}) - \phi(s_t, a_t)]^T w \right).$$

[1] γ can only equal 1 if the total reward is known to be bounded.

Algorithm 1 : Least-squares temporal difference (LSTD)

1. Given samples of experience $\left(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1}\right)$ from following policy π.
2. Estimate A and b from data.
$$\widehat{A} = \sum_t \phi(s_t, a_t) \left[\phi(s_t, a_t) - \gamma \phi(, s_{t+1}, a_{t+1}) \right]^T$$
$$\widehat{b} = \sum_t \phi(s_t, a_t) r_{t+1}$$
3. Return w with extimated zero expected TD update.
$$w = \widehat{A}^{-1} \widehat{b}$$

The weight vector is thus updated in the direction that will reduce TD error. We call u_t the *TD update* for the tth experience tuple.

The TD learning rule finds an approximate value function for a given policy, but it can also be used to find a good policy through policy iteration. Given an initial policy π_1, an approximate value function is found using TD having weight vector w_1. A new policy π_2 is constructed to be greedy with respect to this approximate value function parameterized by w_1. π_2 is then evaluated with TD to find w_2 and the procedure is repeated until convergence or a maximum number of iterations is reached. When evaluating a policy it is not strictly necessary to wait for w to converge when employing TD. In fact, if only one step of experience is used when evaluating a policy, we have the *gradient-descent Sarsa* algorithm. This is the most common method for using TD to find a good control policy.

2.2 Least-Squares TD

The TD update u_t is just one sample. Rather than repeatedly applying sampled updates until w converges, it may be more efficient to directly estimate the weight vector w for which the *expected TD update* is zero. The expected TD update can be written as,

$$E_\pi[u] = E_\pi \left[\phi(s, a)(r + [\gamma \phi(s', a') - \phi(s, a)]^T w) \right],$$

where s, a, r, s', a' are samples of $s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1}$ while following policy π. If we set the expectation to zero and solve for w we get,

$$w = A^{-1} b \quad (3)$$
$$A = E_\pi \left[\phi(s, a)[\phi(s, a) - \gamma \phi(s', a')]^T \right] \quad (4)$$
$$b = E_\pi[\phi(s, a) r]. \quad (5)$$

By using a Monte Carlo estimate of A and b based on samples of experience from following policy π, we have the LSTD algorithm, which is summarized in Algorithm 1. It has been shown that the move from the TD algorithm to the LSTD algorithm is equivalent to moving from a model-free method to a model-based method [1].

2.3 Least-Squares Policy Iteration

LSTD approximates the value of one policy π, but suppose the goal is to find a good policy. It seems natural to embed LSTD inside of policy iteration. Since LSTD requires experience that is explicitly sampled from the policy being evaluated, each improvement to the policy requires new data to be gathered. Not only is it often not convenient to gather new data after each policy change, it also can be very data inefficient to discard all previous experience. Least-squares policy iteration (LSPI) solves this problem by

Algorithm 2 : Least-squares policy iteration (LSPI)

1. Given samples of experience $\left(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1}\right)$.
2. Estimate $\tilde{b}$,
$$\widehat{b} = \sum_t \phi(s_t, a_t) r_{t+1}.$$
3. Repeat while $i <$ maximum number of iterations:
   (a) *Policy Evaluation:* Estimate $\tilde{A}$ and compute w_i
   $$\widehat{A} = \sum_t \phi(s_t, a_t)(\phi(s_t, a_t) - \gamma E_\pi[\phi(s_{t+1}, a_{t+1})|s_{t+1}])^T$$
   (b) *Policy Improvement:* Set π_i to be the greedy policy with respect to w_i.
   (c) If $||w_i - w_{i-1}|| < \epsilon$ return w_i.
   (d) Increment i.

using an off-policy approximation of the expected TD update. This approximation can be estimated in a Monte Carlo fashion from experience, regardless of the policy that the experience was sampled from. Returning to the expected TD update,

$$\begin{aligned} E_\pi[u] &= E_\pi\left[\phi(s,a)(r + [\gamma\phi(s',a') - \phi(s,a)]^T w)\right] \\ &= E_\pi\left[\phi(s,a)(r + [\gamma\phi(s',a') - E_\pi[\phi(s',a')|s']]^T w)\right] \quad (6) \\ &\approx E_{\pi_0}\left[\phi(s,a)(r + [\gamma\phi(s',a') - E_\pi[\phi(s',a')|s']]^T w)\right] \quad (7) \end{aligned}$$

where Equation 6 is an application of iterated expectations, and the final approximation just involves replacing π with any convenient policy π_0, for example, the policy used to collect the experience samples. We can then compute the weight vector with a zero approximate expected TD update,

$$w = \tilde{A}^{-1}\tilde{b} \quad (8)$$

$$\tilde{A} = E_{\pi_0}\left[\phi(s,a)\left(\phi(s,a) - \gamma E_\pi[\phi(s',a')|s']\right)^T\right] \quad (9)$$

$$\tilde{b} = E_{\pi_0}[\phi(s,a)r]. \quad (10)$$

Least-squares policy iteration uses the above approximation in an iterative fashion. Given the current best policy, $\tilde{A}$ and $\tilde{b}$ are estimated in a Monte Carlo fashion by taking a pass over the input experience, computing $E_\pi[\phi(s',a')|s']$, the expected resulting next state-action feature vector if policy π selected the next action a' (If π is deterministic then this expectation is just $\phi(s',a')$ where a' is the action selected by π in state s'. This was how LSPI was originally defined since all the policies being evaluated were greedy with respect to some value function.) The Monte Carlo estimates are then used to compute the weight vector with zero estimated TD update.

The new weight vector implies a new policy, e.g., greedy with respect to the resulting value function. An approximate value function can then be estimated for this new policy, and this is repeated until either the change in the computed weight vector is small or a predetermined maximum number of iterations is reached. The LSPI algorithm is summarized in Algorithm 2.

3. THE GAUSSIAN MDP MODEL

Our goal is learn a policy-independent model of a given MDP, which can then be transformed into a policy-dependent model. This model will be used in a policy-iteration loop, which will allow us to successively improve an initial policy.

As shown in Equations 3–5, least-squares methods use second-order statistics about the relationships between feature vectors as a key quantity. For example, the first term comprising the matrix A is the expected outer product $E_\pi\left[\phi(s_t,a_t)\phi(s_t,a_t)^T\right]$, which can be thought of as a covariance matrix. In effect, least-squares methods are implicitly modelling the system as a large Gaussian with specific first and second moments.

It is this insight that motivates our development of an explicit Gaussian MDP model, which completely characterizes the information needed for policy iteration, but in a policy-independent way. We will now develop our Gaussian MDP model, but wish to emphasize that we are *not* making any Gaussian assumptions about the domains that will be modeled: we are not, for example, assuming that reward functions or transition functions have any sort of Gaussian form. Rather, we will be focusing on covariance-like relationships between feature vectors.

In order to create a policy-independent model of a given MDP, we want to model the system's transitions from state-action pairs to next states. Therefore, we will make use of two different feature mappings. In addition to $\phi(s,a) \in \Re^m$, we also assume we are given a feature mapping on states only $\phi(s) \in \Re^n$. We make two explicit assumptions about this representation:

A1. There exists a mapping Φ, such that $\phi(s,a) = \Phi(\phi(s), a)$.

A2. For any policy π that the agent can execute, there exists a mapping Π, such that $\pi(s,a) = \Pi(\phi(s), a)$.

Assumption A1 requires that $\phi(s)$ contain no less information about state s than $\phi(s,a)$. Assumption A2 requires $\phi(s)$ to contain all the information about s that a policy might depend upon. These two assumptions are quite common. Consider the very basic representation choice that takes an arbitrary set of features over states $\phi(s) \in \Re^n$ and replicate these features for each action. So $\phi(s,a) \in \Re^{|A|n}$ has the ith block of the vector set to $\phi(s)$ when taking action i and zero otherwise. For any value based policies (i.e., greedy, ϵ-greedy, Boltzmann), this representation satisfies both Assumptions A1 and A2. Many other approximation architectures satisfy these assumptions as well.

We now define the *Gaussian MDP model*, which will form the basis for our model-based approach. In our proposed model, states themselves are vectors in $\Re^n$, but they can be thought of as state feature vectors. Actions are from an arbitrary set $\mathcal{A}$. Given a state feature vector $\phi(s)$ and an action a the MDP stochastically produces a next-state feature vector $\phi(s')$ and a reward r. In a Gaussian MDP these random variables are distributed as a multivariate Gaussian with the following form,

$$\phi(s') \sim N(m_{s'} + F\Phi(\phi(s),a),\ S_{s'}) \quad (11)$$

$$r \sim N(m_r + R\Phi(\phi(s),a),\ s_r^2), \quad (12)$$

where $m_{s'} \in \Re^{n\times 1}$, $m_r \in \Re^{1\times 1}$, $S_{s'} \in \Re^{n\times n}$, $s_r \in \Re^{1\times 1}$, $F \in \Re^{n\times m}$, and $R \in \Re^{1\times m}$ are all parameters of the model. *Notice that unlike the statistics computed by LSTD, this model is independent of policy and is a fully generative model.*

3.1 Learning a Gaussian MDP

Now we will show how we can estimate the needed parameters from data. Then, in the next section, we will show how we can perform an efficient form of policy iteration with this model.

Assume we are given samples from the transition and reward model, $(\phi(s_t), a_t, r_{t+1}, \phi(s_{t+1}))_{t=1\ldots T}$. We can find the maximum likelihood estimate for the parameters $(\mu_{s'}, F)$ and (μ_r, R) through simple linear regression between the inputs $\phi(s_t, a_t)$ and the outputs $\phi(s_{t+1})$ and r_{t+1}, respectively. The maximum likelihood estimates of the covariance and variance parameters, $\mathbf{\Sigma}_{s'}$ and σ_r^2, are then just the averages of the outer products of the residuals. We can arrive at the same maximum likelihood parameter estimates with an alternative procedure, which will prove convenient later. Let $x_t = [\phi(s_t, a_t)\ \ r_{t+1}\ \ \phi(s_{t+1})]^T$ be a column vector summarizing the tth sample of experience. Consider the unknown joint distribution from which the experience x_t is generated. Assume this joint distribution is a multivariate Gaussian and so has the form,

$$x \sim N\left(\begin{bmatrix} \mu_{sa} \\ \mu_r \\ \mu_{s'} \end{bmatrix}, \begin{bmatrix} \mathbf{\Sigma}_{sa} & \mathbf{\Sigma}_{sa,r} & \mathbf{\Sigma}_{sa,s'} \\ \mathbf{\Sigma}_{r,sa} & \mathbf{\Sigma}_r & \mathbf{\Sigma}_{r,s'} \\ \mathbf{\Sigma}_{s',sa} & \mathbf{\Sigma}_{s',r} & \mathbf{\Sigma}_{s'} \end{bmatrix}\right). \quad (13)$$

We call this joint distribution the *joint Gaussian model*. Now, we can write the conditional distributions $\phi(s')|\phi(s,a)$ and $r|\phi(s,a)$,

$$\phi(s')|\phi(s,a) \sim N\big(\mu_{s'} - F(\phi(s,a) - \mu_{sa}), \mathbf{\Sigma}_{s'} - F\mathbf{\Sigma}_{sa,s'}\big)$$
$$r|\phi(s,a) \sim N\big(\mu_r - R(\phi(s,a) - \mu_{sa}), \mathbf{\Sigma}_r - R\mathbf{\Sigma}_{sa,r}\big)$$

where,

$$F = \mathbf{\Sigma}_{s',sa}\mathbf{\Sigma}_{sa}^{-1} \qquad R = \mathbf{\Sigma}_{r,sa}\mathbf{\Sigma}_{sa}^{-1}$$

Notice that these distributions match the form of the Gaussian MDP model from Equations 11 and 12, with model parameters,

$$\begin{aligned} F &= \mathbf{\Sigma}_{s',sa}\mathbf{\Sigma}_{sa}^{-1} & R &= \mathbf{\Sigma}_{r,sa}\mathbf{\Sigma}_{sa}^{-1} \\ m_{s'} &= \mu_{s'} - F\mu_{sa} & m_r &= \mu_r - R\mu_{sa} \\ S_{s'} &= \mathbf{\Sigma}_{s'} - F\mathbf{\Sigma}_{sa,s'} & s_r^2 &= \mathbf{\Sigma}_r - R\mathbf{\Sigma}_{sa,r}. \end{aligned} \quad (14)$$

If we use maximum likelihood to estimate the parameters of the joint Gaussian model from Equation 13, then the MDP model parameters above are, in fact, the maximum likelihood estimators for the MDP model.

In summary, we can estimate the parameters of a Gaussian MDP model by finding the maximum likelihood parameters of the joint Gaussian model of x_t. We can then apply the Equations in 14 to compute the MDP model parameters, although we will find it more convenient to simply work in the space of the joint Gaussian model.

4. SIGMA POINT POLICY ITERATION

In this section, we show how to perform policy iteration on our Gaussian MDP model. Like any policy iteration method, there are two elements: policy evaluation and policy improvement. Recall that the Gaussian MDP model itself is independent of any policy. We need to transform this Gaussian MDP model into a form suitable for use in evaluating a specific policy using a linear least-squares method.

Our method works as follows. We can think of the information needed for linear least-squares policy evaluation as statistics about some large random variable $\mathcal{M}_\pi$. We can think of our Gaussian model as some other large random variable $\mathcal{M}_\emptyset$. On each iteration, we will transform our policy-independent joint Gaussian model $\mathcal{M}_\emptyset$ into a policy-dependent model $\mathcal{M}_\pi$, which will contain exactly the information needed to compute the linear least-squares estimate of the value function.

The key conceptual element to our algorithm involves a mapping T_π which transforms $\mathcal{M}_\emptyset$ into $\mathcal{M}_\pi$. Specifically, it transforms the experience summarized about $\phi(s')$ into information about $\phi(s', a')$, where a' is drawn according to policy π. The existence of such a mapping relies explicitly on Assumptions A1 and A2 (see Section 3). Because T_π will generally be nonlinear, a closed-form solution of the transformation will rarely be available. For such cases, we propose a sigma point approximation as a principled and efficient solution.

We will now explain the mapping T_π in detail. Recall that we can approximate the expected TD update as,

$$E_\pi[u] \approx E_{\pi_0}\Big[\phi(s,a)[r + (\gamma E_\pi[\phi(s',a')|\phi(s')] - \phi(s,a))^T w]\Big], \quad (15)$$

where π is the policy to be evaluated and π_0 a policy of convenience. In particular, we will choose π_0 to be the policy used to collect the data for estimating the Gaussian model. Define the following transformation T_π on vectors x from our joint Gaussian model,

$$T_\pi\left(\begin{bmatrix} \phi(s,a) \\ r \\ \phi(s') \end{bmatrix}\right) = \begin{bmatrix} \phi(s,a) \\ r \\ \sum_{a'} \Phi(\phi(s'),a')\Pi(\phi(s'),a') \end{bmatrix}.$$

Notice that T_π leaves the first two components of the vector unchanged, but replaces $\phi(s')$ with the expectation of $\phi(s', a')$ if π were used to select a' from s'. In other words, the third component is $E_\pi[\phi(s', a')|\phi(s')]$.

The results of the mapping have exactly the needed properties. The distribution of $T_\pi(x)$ is such that (i) the marginal distribution of $\phi(s, a)$ is from π_0, (ii) the conditional distribution of $\phi(s')$ and r given $\phi(s, a)$ is from the maximum likelihood estimate of the Gaussian MDP, and (iii) the expectation of $\phi(s', a')$ given $\phi(s')$ for policy π is part of $T_\pi(x)$. These are exactly the terms appearing in the approximation of the expected TD update in Equation 15. Moreover, the first and second moments of $T_\pi(x)$ are all that are needed to compute the approximation. Specifically,

$$\begin{aligned} E[u] \approx (\Sigma_{sa,r} + \mu_{sa}\mu_r^T) - (\Sigma_{sa} + \mu_{sa}\mu_{sa}^T)w \\ + \gamma(\Sigma_{sa,s'a'} + \mu_{sa}\mu_{s'a'}^T)w. \end{aligned}$$

Given these mean and covariance terms, the weight vector w for which this approximation is zero can then be computed as,

$$\begin{aligned} A &= \Sigma_{sa} + \mu_{sa}\mu_{sa}^T - \gamma(\Sigma_{sa,s'a'} + \mu_{sa}\mu_{s'a'}^T) \\ b &= \Sigma_{sa,r} + \mu_{sa}\mu_r^T \\ w &= A^{-1}b. \end{aligned} \quad (16)$$

We now must estimate the first and second moments of a transformed multivariate Gaussian distribution. If the transformation T_π is linear then the resulting transformed random variable would also be an easily computed Gaussian. In general, though, T_π will be a nonlinear function.

4.1 Sigma-Point Approximations

Abstractly, we now have the problem of propagating the Gaussian random variable x through the nonlinear function T_π, and computing the first two moments of the result.

Sigma-point approximations, or "unscented transformations" [4], are a general method of propagating an arbitrary distribution through a nonlinear function. The method is conceptually simple, and should be thought of as a deterministic sampling approach. Suppose we are given a random variable $Y = f(X)$ that is a nonlinear function of a Gaussian random variable X. Instead of recording the distribution information of X in terms of a mean and covariance, we represent the same information with a small, carefully chosen number of *sigma points*. These sigma points are selected so that they

have the same mean and covariance as X, but the advantage is that they can be propagated *directly* through the nonlinear function f. We then compute the desired posterior statistics of the propagated points as approximations of the statistics of Y.

Sigma-point approximations should not be confused with particle filters. While they are similar in spirit, there are several important differences. Particle filters typically allow a multi-modal distribution over states, while sigma-point approximations require a Gaussian; it is the Gaussian assumption which gives the sigma-point approximation its strong theoretical guarantees with a small number of points. Also, where particle filters use random sampling, sigma-point approximations use deterministic sampling.

Sigma point approximations not only perform well in practice, but are theoretically well grounded. They are provably accurate to at least second order for any f and distribution of X, and are accurate to third order if X is Gaussian. Fourth order terms can even sometimes be corrected. Sigma point approximations are exact if the function f is linear, which will be important later as we relate sigma-point policy iteration to LSPI.

4.2 Using Sigma-Points to Transform Policies

We now present a more detailed explanation in the context of our problem of computing the first two central moments of $T_\pi(x)$. The sigma-point approximation deterministically selects a set of points $x_{i=1\ldots 2d+1}$ from the original distribution, where $d = m + n + 1$ is the dimensionality of the distribution x. Each sigma point is also assigned a weight, w_i, such that the weighted sum of sigma points is the original distribution's mean and the weighted sum of outer products of differences from the mean is the original distribution's covariance matrix.

The procedure we used for selecting sigma points is $x_1 = \mu_x$, $x_{2i:2i+1} = \mu_x \pm (\sqrt{\Sigma_x})_i$ with weights $x_1 = \kappa/(\kappa + d)$ and $x_{i>1} = 0.5/(\kappa + d)$. The experiments in this paper all use $\kappa = 0$. The nonlinear function is then applied to each point and the approximated first two central moments are just the weighted mean and outer product differences, *i.e.*,

$$\tilde{\mu}_y = \sum_{i=1}^{2d+1} w_i T_\pi(x_i) \tag{17}$$

$$\tilde{\Sigma}_y = \sum_{i=1}^{2d+1} w_i (T_\pi(x_i) - \tilde{\mu}_y)(T_\pi(x_i) - \tilde{\mu}_y)^T.$$

We can then use the resulting transformed mean and covariance ($\tilde{\mu}_y$ and $\tilde{\Sigma}_y$) in Equation 16 to find the weight vector for our approximation of the policy's value function.

The sigma point approximation thus forms the crux of sigma point policy iteration. On each iteration, we transform a set of sigma points from our joint Gaussian model using our current best policy and compute the first and second order statistics of the transformed points. The resulting means and covariances are used to solve for a new weight vector, which defines a new policy. This is repeated until either the change in the weight vector is small or a predetermined maximum number of iterations is reached. The SPPI algorithm is summarized in Algorithm 3.

4.3 Discussion

There are a number of interesting observations that can be made about SPPI and LSPI.

4.3.1 Time and Memory Analysis

SPPI summarizes the input experience in a compact policy-independent model that requires only $O(m^2)$ space. As such, after a single pass over the input, the experience can be deleted. For each iteration, an additional $O(m^2)$ space is used to transform and summarize the sigma points. Meanwhile, $O(Tm^2)$ time is needed to compute the joint Gaussian model, but then each iteration only requires $O(|A|m^2 + m^3)$ time. Contrast this with LSPI, which requires $O(m^2 + Tm)$ space and $O(T(|A|m + m^2) + m^3)$ time per iteration as the experience data needs to be saved and replayed with each change to the policy. Note that the efficient incorporation of new data, without any dependence on T makes SPPI significantly more applicable to online use.

Algorithm 3 : Sigma point policy iteration (SPPI)

1. Given samples of experience $(\phi(s_t), a_t, r_{t+1}, \phi(s_{t+1}))$ estimate a joint Gaussian model (as in Equation 13),

$$\boldsymbol{\mu} = \begin{bmatrix} \mu_{sa} \\ \mu_r \\ \mu_{s'} \end{bmatrix} = \tfrac{1}{T} \textstyle\sum_t \begin{bmatrix} \phi(s_t, a_t) \\ r_{t+1} \\ \phi(s_{t+1}) \end{bmatrix},$$

$$\boldsymbol{\Sigma} = \begin{bmatrix} \boldsymbol{\Sigma}_{sa} & \boldsymbol{\Sigma}_{sa,r} & \boldsymbol{\Sigma}_{sa,s'} \\ \boldsymbol{\Sigma}_{r,sa} & \boldsymbol{\Sigma}_{r} & \boldsymbol{\Sigma}_{r,s'} \\ \boldsymbol{\Sigma}_{s',sa} & \boldsymbol{\Sigma}_{s',r} & \boldsymbol{\Sigma}_{s'} \end{bmatrix} = \tfrac{1}{T} \textstyle\sum_t \begin{bmatrix} \phi(s_t, a_t) \\ r_{t+1} \\ \phi(s_{t+1}) \end{bmatrix} \begin{bmatrix} \phi(s_t, a_t) \\ r_{t+1} \\ \phi(s_{t+1}) \end{bmatrix}^T - \boldsymbol{\mu}\boldsymbol{\mu}^T$$

2. Repeat while $i <$ maximum number of iterations:
   (a) *Sigma point approximation*: Compute the first two moments of $T_{\pi_i}(x)$ (see Equation 17).
   (b) *Policy Evaluation:* Compute w_i from the first two moments of $T_{\pi_i}(x)$ (see Equation 16).
   (c) Set π_i to be the greedy policy with respect to w_i.
   (d) If $||w_i - w_{i-1}|| < \epsilon$ return w_i.
   (e) Increment i.

4.3.2 Relationship to LSTD/LSPI

SPPI is using an approximation (based on sigma points) to compute the weight vector with approximately zero expected TD update, and so, in general, will find a different policy than LSPI. Under certain circumstances this approximation is exact and SPPI will find the same weight vector and policy as LSPI if given the same samples of experience. In particular, consider the case of a Markov chain where $|A| = 1$. Notice that Assumption A1 forces $\phi(s, a) = \phi(s)$ in this case. Hence, T_π is actually the identity map and as a result the sigma point approximation is exact. Moreover, SPPI will compute the same approximate value function as LSTD.

Similarly, consider a tabular representation of an MDP. Each state s and state-action pair (s, a) is given a unique feature and all feature vectors consist of a single non-zero entry. The transformation T_π can be written as a matrix multiplication,

$$T_\pi(x) = \begin{bmatrix} I & 0 & 0 \\ 0 & I & 0 \\ 0 & 0 & P \end{bmatrix} x,$$

where P is an $|S||A| \times |S|$ matrix such that the entry $((s, a), s')$ is the probability of π selecting action a in state s if $s = s'$ and zero otherwise. Since T_π is a linear transformation, the sigma point approximation is exact, and SPPI will produce the same output as LSPI.

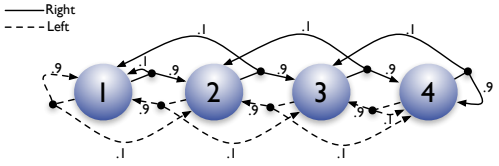

Figure 1: Chain walk domain with four states

4.3.3 *Implementation Details*

There are some subtleties to the implementation. First, the method calls for a matrix square root. There are an infinite number of suitable square roots, all of which are related to each other by orthonormal transformations. The literature prefers the Cholesky decomposition, because it is fast and numerically stable, and because it can be used in square-root versions of Kalman-filter algorithms [10].

The second issue is that when the new Gaussian random variable $T_\pi(x)$ is constructed, its covariance matrix is not guaranteed to be symmetric positive-definite, which means that a Cholesky decomposition will fail on it. There are a number of possible ways to force it to be positive definite: Higham presents an optimal method (with respect to the Frobenius norm) of finding the nearest SPD matrix to a given matrix A [3], and other similar methods exist. Unfortunately, his method essentially zeros out any negative eigenvalues, making the result rank-deficient. Another popular approach is to replace any negative eigenvalues with small positive constants, which preserves the rank.

Note that the inverse in Eq. 16 does not need to be computed explicitly, since only an inverse-vector product is needed. Instead, an iterative linear solver should be used, such as conjugate gradients or GMRES [7]. This is especially preferred if the matrix A is sparse.

5. SETUP OF EMPIRICAL EVALUATION

In this section we investigate the empirical performance of SPPI as compared to LSPI. As SPPI depends on an additional approximation, these results focus on evaluating the effect of this approximation on finding good policies. In addition, we are also interested in the computational costs of the two approaches.

The two algorithms were compared in four problem domains: random MDPs, the chain walk, the inverted pendulum, and a continuous maze domain. The chain walk and inverted pendulum were both used in the original evaluation of LSPI [6]. In all domains, the algorithms were given a maximum number of 25 iterations and a tolerance of $\epsilon = 10^{-6}$. They were each also given the same experience always generated from the random policy. In all cases a problem specific feature map was used for $\phi(s)$ and the features were replicated for each action to create the stacked $\phi(s, a)$ as described in Section 3.

For the random MDPs and chain walk domain, the resulting policies were evaluated in terms of average reward over 1000 steps with the whole process repeated 50 times. For the pendulum domain, the setup was the same except that trials were a maximum of 3000 steps repeated 200 times. For the continuous maze, 30 trials of 50,000 steps were used.

5.1 Random MDPs

We generated random MDPs with 50 states, 3 actions, randomly generated transition probabilities, and uniform random state action rewards from the interval $[-0.5, 0.5]$. Each algorithm was provided with a 1000 step trajectory as experience samples. In the first experiment, a tabular feature representation was used, and in the second experiment each state was given a randomly generated feature vector with $n = 25$. The discount rate was 0.9.

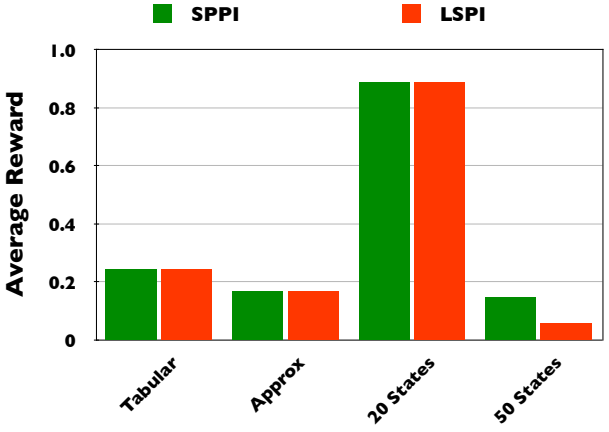

Figure 2: Average reward of SPPI and LSPI in random MDPs and chain walk.

5.2 Chain Walk

First introduced by Koller and Parr [5], the chain walk environment consists of n states with reward of 0, except when taking actions in the two goal states where it is $+1$. The two actions move the agent left or right, although with probability 0.1 the action has the opposite effect. Figure 1 illustrates a chain walk with 4 states. We examined two chain domains. The first had 20 states with the goals at state 1 and 20, and the algorithms were given a 1000 step trajectory. The second had 50 states with goals at state 11 and 40, and the algorithms were given a 2000 step trajectory. In both cases the state feature vector contained five features, with the the ith feature for state s being s^{i-1}. More detail about the chain walk and the difficulties with low degree polynomial feature approximation can be found in the work of Lagoudakis and Parr [6].

5.3 Inverted Pendulum

The episodic inverted pendulum task consists of a pendulum and a cart. The goal is to balance the pendulum on the cart. The states are the vertical angle (θ) and angular velocity ($\dot{\theta}$) of the pendulum, and there are three actions: *right-force*, *left-force*, and *no-force*, where uniformly distributed zero-mean noise up to 20% of normal force is added to the action. The reward is zero as long as $\theta < \frac{\pi}{2}$. Otherwise, the episode is finished with a reward of -1. The state features consisted of a constant and nine radial basis function features. The algorithms were evaluated based on the length of the episode which was capped at 3000. See the work of Lagoudakis and Parr [6] and Wang et al. [11] for details.

5.4 Continuous Maze

In this domain an autonomous robot must navigate a simple maze. The state space consists of x, y coordinates and an orientation θ. The agent has four actions: move forward, move backward, and turn left / right by 15 degrees. The reward signal is given as a Gaussian centered at the goal. The features were generated using 100 radial basis functions scattered randomly throughout the state space. Specifically, $\phi(s)_i = G(s - c_i; \sigma^2)$ where c_i is the center of the i'th Gaussian, σ^2 is its variance, and G is the Gaussian density function. This results in a "soft unit-basis encoding" of the state

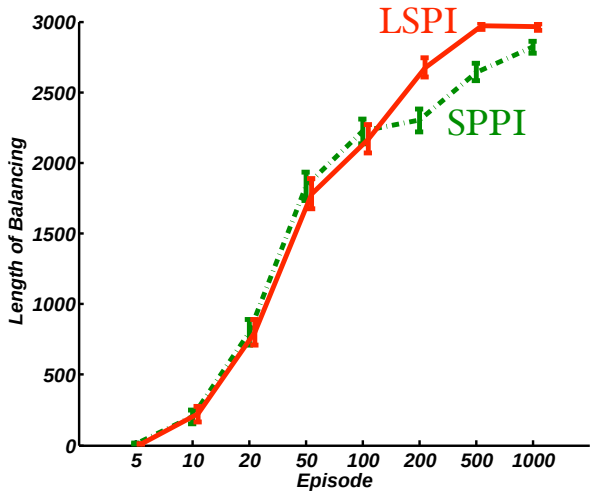

Figure 3: Average reward of SPPI and LSPI in the pendulum problem

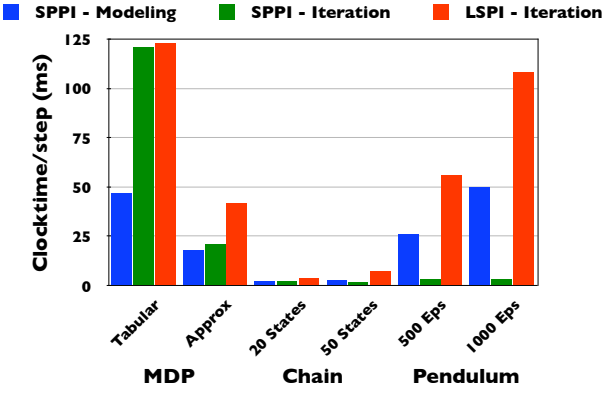

Figure 4: Timing results of SPPI and LSPI in our three domains.

space.

6. EMPIRICAL RESULTS

Figure 2 shows the average reward per time step obtained by the two algorithms for the random MDP and chain walk domains. For the tabular-based random MDPs, we know SPPI will produce the same policy as LSPI, and indeed the empirical performances of the approaches are identical. Somewhat surprisingly, SPPI suffered no observable degradation (relative to LSPI) when function approximation was used, even though the sigma point approximation in this case is inexact. In fact, its performance across all four experiments is indistinguishable from LSPI.

Figure 3 shows the performance of SPPI and LSPI on the pendulum task for increasing amounts of training data. Notice that for better visibility the horizontal axis is not uniformly distributed. On this more complex problem, SPPI remains competitive with LSPI, although LSPI has a noticeable advantage with medium to large amounts of data.

Figure 4 plots a time comparison of the two algorithms on the three domains. The horizontal axis shows the domain, while the vertical axis shows the clock-time in milliseconds. For each environment, we show both the time for the initial model construction as well as the per iteration time for SPPI along side the per iteration for LSPI. Although SPPI needs extra time to construct the Gaussian model, the time is not significant and is usually made up for after just one single step of policy iteration. In practice, both techniques required a similar number of policy iteration steps (5–7) but with SPPI running approximately an order of magnitude faster on the larger domains.

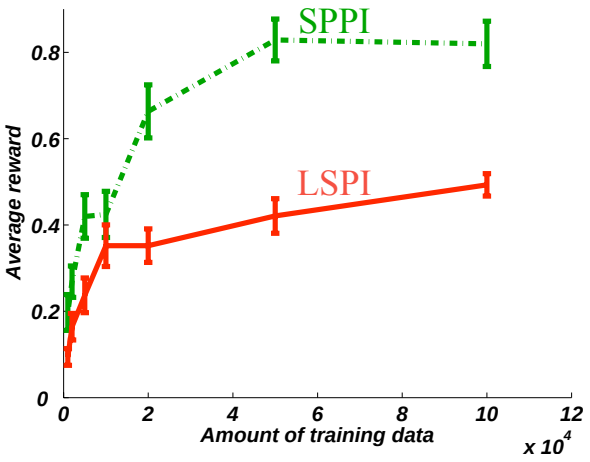

Figure 5: Performance results of SPPI and LSPI in the maze domain.

Finally, Figure 5 shows the performance of SPPI and LSPI on the maze domain for increasing amounts of training data. In this case, SPPI consistently outperforms LSPI. The maximum reward possible in this domain is 1.0, implying that SPPI is performing very well relative to optimal performance. Again, SPPI also performed well in terms of wall clock time: to compute a policy with 100,000 training points, SPPI took 22 seconds while LSPI took 549 seconds.

Because SPPI is an approximation of LSPI, SPPI's superior performance is quite unexpected. Figure 6 illustrates why this occurs. The maze is simple, and there are two acceptable paths to the goal: LSPI chooses one path which is slightly shorter, but which has two turns, while SPPI chooses a slightly longer path with only one turn. The problem is with the final turn: LSPI consistently turns too soon, and clips the corner of the wall. In this domain, when an agent bumps into a wall, no motion occurs. Since the policy deterministically maps states to actions, and since the state does not change, the agent bumps into the wall forever. To help counter this, we also tested with a mildly stochastic policy. While the occassional random actions sometimes allowed the agent to escape this trap, it usually could not. In contrast, SPPI chose a path which did not involve the second turn, and never got stuck. This problem could perhaps be dealt with by adding more features (thereby giving higher resolution to problematic regions) or by adding a small negative reward for bumping into walls (thereby encouraging agents to walk down the middle of the hall). In any case, the improved performance seems to be largely due to chance, but does illustrate that different policies can have other properties (such as robustness) that are not explicitly optimized for.

These results suggest three broad conclusions: in tabular domains, SPPI and LSPI perform equally well, as expected. In some

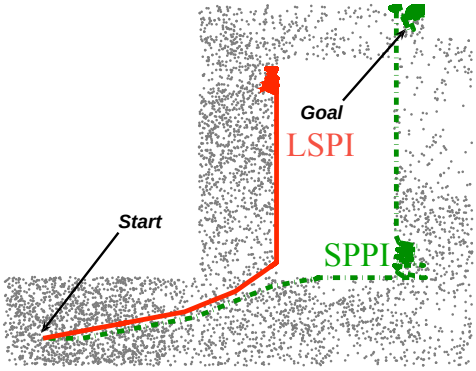

Figure 6: Example solution paths on the maze domain.

domains (pendulum) the approximate nature of SPPI harms performance, but in other domains (maze), the approximate nature helps. In all cases, SPPI is able to learn policies much more quickly than LSPI for large data sets, reflecting the fact that the complexity of the policy iteration stages do not depend on the amount of data.

7. CONCLUSION

We have proposed a new linear least-squares method called Sigma Point Policy Iteration. We introduced the Gaussian MDP model, which captures policy-independent statistics about the process. We then showed that under reasonable conditions on the features used for approximation that this model can be be transformed into a policy-dependent form suitable for least-squares policy evaluation. We described how sigma-points can be used to approximate the transformation efficiently. The result is an algorithm for model-based policy iteration with linear function approximation that does not require data to be saved or replayed. Our empirical results suggest SPPI is viable, demonstrating on a number of domains that it generates policies on-par with LSPI while requiring an order of magnitude less computation.

There are several exciting directions for future work. First, as we have briefly noted, the complexity of SPPI does not depend upon the number of data points used to estimate the model. This means that new experience can be efficiently incorporated into the Gaussian MDP model. Therefore, it may be possible to employ SPPI in an online fashion, which is not practical with LSPI. Second, SPPI's explicit construction of a model makes it possible to explore a more Bayesian approach for fitting the model from data. Even an uninformed prior might provide a useful regularization effect, and a more complicated prior could allow for the natural inclusion of domain and circumstance specific knowledge, such as experience with a similar MDP.

Third, higher-order unscented transforms may further improve the accuracy of the method by incorporating third and fourth order statistics into the model. Finally, it may be possible to incorporate knowledge of the policy transformation (e.g., greedy policies result in piecewise linear transformations) to further improve the sigma point approximation accuracy.

Acknowledgments

Michael Bowling and Alborz Geramifard were supported by NSERC, iCore, and Alberta Ingenuity through the Alberta Ingenuity Centre for Machine Learning. David Wingate was supported by an NSF Graduate Research Fellowship.

8. REFERENCES

[1] J. A. Boyan. Least-squares temporal difference learning. In *Proceedings of the Sixteenth International Conference on Machine Learning*, pages 49–56. Morgan Kaufmann, San Francisco, CA, 1999.

[2] S. Bradtke and A. Barto. Linear least-squares algorithms for temporal difference learning. *Machine Learning*, 22:33–57, 1996.

[3] N. J. Higham. Computing a nearest symmetric positive semidefinite matrix. *Linear Algebra and its Applications*, 103:103–118, 1988.

[4] S. Julier and J. K. Uhlmann. A general method for approximating nonlinear transformations of probability distributions. Technical report, University of Oxford, 1996.

[5] D. Koller and R. Parr. Policy iteration for factored MDPs. In *Proceedings of Uncertainty in Artificial Intelligence (UAI)*, pages 326–334, 2000.

[6] M. G. Lagoudakis and R. Parr. Least-squares policy iteration. *Journal of Machine Learning Research*, 4:1107–1149, 2003.

[7] Y. Saad. *Iterative Methods for Sparse Linear Systems*. PWS Publishing, Boston, 1996.

[8] R. S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3:9–44, 1988.

[9] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[10] R. van der Merwe and E. A. Wan. The square-root unscented Kalman filter for state and parameter-estimation. In *International Conference on Acoustics, Speech, and Signal Processing*, 2001.

[11] H. O. Wang, K. Tanaka, and M. F. Griffin. An approach to fuzzy control of non-linear systems: Stability and design issues. *IEEE Transactions on Fuzzy Systems*, 4(1):14–23, 1996.