Policy Gradient Methods for Automated Driving

Stanford AA228 Winter 2015

Tim A. Wheeler
Department of Aeronautics and Astronautics
Stanford University
Stanford, California 94305–4035
Email: wheelert@stanford.edu

Abstract—Automated driving requires designing a system capable of maintaining safety while simultaneously maintaining passenger comfort. Models of highway driving have high dimensionality and stochasticity traditionally specifying the histories for a large, varying number of agents in a continuous state and action space. Traditional value-based reinforcement learning methods require exponential time in the size of the state space and thus often require coarse state and action space discretization to remain tractable. Policy gradient methods instead maintain constant complexity proportional to the complexity of their parameterization and are thus candidates for use in such applications if suitable parameterizations can be found. This project explores several policy gradient methods against the problem of optimal autonomous highway driving framed as a Markov decision process.

I. INTRODUCTION

Modern automotive safety systems can no longer rely on real-world testing for development and certification as systems are more complicated and collision rates are measured per millions of miles [1]. Validation through simulation allows for fast and cheap parallel evaluation of trajectories spanning the entire state space. Such methods have already been adopted in civil aviation, in which detailed airspace encounter models are used to accurately capture the behavior of encounter participants [2], [3].

These same simulation models used for policy validation can be directly used to optimize driving policies. A natural progression is to derive optimal driving policies against such a validation system. In this work, a probabilistic traffic propagation model is assumed to accurately capture the driving behavior of other traffic participants.

Consider the problem of automotive collision avoidance in highway traffic. A vehicle equipped with autonomous driving technology is capable of constantly monitoring its environment and acting within it. It is desirable that the vehicle maintain a target speed with minimal acceleration while simultaneously maintaining safe distances between itself and other vehicles. This problem is framed as a Markov decision process (MDP) in which other vehicles act according to a probabilistic driving model. The solution to this MDP is a driving policy which appropriately balances driver comfort and safety.

Traditional approaches in reinforcement learning to solving such problems suffer from the curse of dimensionality, requiring exponential time in the size of the state space. [4] These methods often require approximations, and even then can only remain tractable with excessively coarse granularity. Policy gradient methods have constant complexity depending only on their parameterization, with policy representations typically far more compact than in value-function based approaches. The policy parameterization can be chosen to encode domain-specific knowledge, often with far fewer parameters that need to be learned in other approaches. Several policy gradient methods with strong theoretical underpinnings exist in the literature.

The main goal of this paper is to investigate the use of policy gradient methods for problems with continuous state and action spaces with high dimensionality which are difficult to solve using value-based approaches. A car-following problem is devised using stochastic human behavior models from the literature. The remainder of this paper will proceed as follows: first, an overview of Markov decision processes and the notations used in this paper are given. Secondly, an overview of policy gradient methods is given. Several specific algorithms are reviewed. Thirdly, the performance of these methods is tested in two base problems. Finally, the autonomous driving problem is formulated as an MDP and the results of applying these methods is presented.

II. MARKOV DECISION PROCESSES

A finite stationary MDP is described by a five-tuple $< S, A, T, R, T >$ where $S$ represents the system state space, $A$ the action space, $T(s, a, s')$ the transition probabilities between states, $R(s, a)$ the reward obtained for taking action $a$ at state $s$ and $T$ the finite horizon to which the MDP is evaluated. MDPs satisfy the Markov property that a successor state $s'$ depends only on the current state $s$ and the action taken $a$. The problems considered in this article possess continuous state and action spaces but assume that the policy is executed in discrete time steps. The current time step will be denoted by $t$.

A policy $\pi(s)$ is a mapping of states to actions. Following a policy results in a sequence of states and actions, together with a trajectory denoted by $\tau = [s_0:T, a_0:T]$. An MDP is solved by finding a policy which maximizes the expected return...
\[ J(\pi) = \mathbb{E} \left\{ \sum_{t=0}^{T} r_t \right\}. \] (1)

### III. POLICY GRADIENT METHODS

Policy gradient methods assume that actions are generated by a stochastic policy \( a_t \sim \pi_\theta(a_t \mid s_t) \) with continuous parameterizations given by \( \theta \). The problem of solving an MDP thus becomes one of optimizing the policy parameters \( \theta \) to maximize the expected return \( J(\theta) = J(\pi_\theta) \).

These methods continuously refine a given policy parameterization vector by following the steepest ascent on the expected return. This is naturally formalized by the gradient update rule

\[ \theta_{h+1} = \theta_h + \alpha_h \nabla_\theta J |_{\theta=\theta_h}, \] (2)

where \( \alpha_h \in \mathbb{R}^+ \) denotes a learning rate and \( h \in \{0, 1, 2, \ldots\} \) denotes the current update iteration. It can be shown that if the gradient is known and the learning rates satisfy \( \sum_{h=0}^{\infty} \alpha_h > 0 \) and \( \sum_{h=0}^{\infty} \alpha_h^2 = \text{const} \) then the learning process is guaranteed to converge to a local minimum\[5\].

The methods presented next estimate the gradient \( \nabla_\theta J |_{\theta=\theta} \) necessary for ascent. Analytic solutions for the true gradient are often not available, and often times the detailed internal workings of a system from which such a gradient may be calculated will be lacking. One must thus estimate the gradient through policy rollouts using a black box trajectory generator.

#### A. Finite Differences

Finite difference methods are perhaps the most straightforward approaches for gradient estimation. The gradient with respect to each policy parameter is estimated by a simple difference estimate

\[ \nabla_{\theta_i} J(\theta) = \frac{J(\theta + \epsilon \hat{e}_i) - J(\theta - \epsilon \hat{e}_i)}{2\epsilon} \] (3)

where \( \hat{e}_i \) is the unit vector with direction only in the \( i \)-th coordinate and \( \epsilon \) is a small scalar. The expected return is computed via rollouts following the stochastic parameterized policy. The policy gradient estimate \( g_{FD} \approx \nabla_\theta J |_{\theta=\theta_h} \) is directly obtained by setting each element \( i \) to the variation \( \nabla_{\theta_i} J(\theta) \).

Gradient estimation by finite differences is easy to understand and implement but is sensitive to how the policy is parameterized. Problems may have parameterizations where variables differ by several orders of magnitude, possessing different sensitivities to gradient estimate step size \( \epsilon \) and potentially leading to policy instabilities\[5\].

#### B. REINFORCE / Likelihood

The REINFORCE or Likelihood methods are derived by considering the space of trajectories \( \tau \) generated by rollout. Here, one can consider the policy to act as a probability function over trajectories, \( \tau \sim p(\tau \mid \theta) \), producing rewards

\[ r(\tau) = \sum_{t=0}^{T} r_t. \] The policy gradient can be estimated using the likelihood ratio\[6\]

\[ \nabla_\theta J(\theta) = \int_\tau \nabla_\theta p_\theta(\tau) r(\tau) d\tau = \mathbb{E} \{ \nabla_\theta \log p_\theta(\tau) r(\tau) \} \] (4)

\[ = \mathbb{E} \{ \nabla_\theta \log p_\theta(\tau) r(\tau) \} \] (5)

\[ , \] due to the integration being readily factorized: \( \int_\tau \nabla_\theta p_\theta(\tau) r(\tau) d\tau = \int_\tau p_\theta(\tau) \nabla_\theta \log p_\theta(\tau) r(\tau) d\tau. \) The gradient \( \nabla_\theta \log p_\theta(\tau) \) can be computed directly from rollouts and does not require knowledge of the underlying distribution \( p_\theta(\tau) \). The general path likelihood ratio estimator is given by\[5\]

\[ g_{RF} = \left\langle \left( \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t) \right) \left( \sum_{t=0}^{T} r_t - b \right) \right\rangle \] (6)

where \( b \) is a constant baseline that can be chosen to minimize the gradient variance and \( \langle \cdot \rangle \) denotes the average over rollout trajectories.

Likelihood ratios require that the log policy gradient be known. This is often easy to ensure as it merely requires taking the gradient over the policy probability distribution which is specified by the problem implementor, not the model. Furthermore, policy parameter variations are no longer required. Finally, a single rollout can trace out a large chunk of the state space and gather a significant amount of policy gradient information, thus reducing the number of necessary rollouts.

#### C. Natural

The natural policy gradient, or episodic actor critic policy gradient, addresses an issue with REINFORCE concerning the variation of the gradient with respect to the parameterization. It is desirable that the estimated gradient remain invariant with affine transformations of the parameterization. This variation leads to ambiguities in the size of gradient steps needed between these parameterizations.

The Fisher information matrix \( F_\theta \) is used to transform the estimated gradient to result in a restricted gradient ascent step size

\[ \Delta \theta = \alpha_n F_\theta^{-1} \nabla_\theta J \] (7)

\[ F_\theta = \langle \nabla \log p_\theta(\tau) \nabla \log p_\theta(\tau)^\top \rangle \] (8)

where \( \nabla_\theta J \) is a vanilla gradient estimated by REINFORCE without a baseline.

The natural policy gradient method theoretically converges faster than the REINFORCE method and frees the method from the policy parameterization.
D. Bayesian

The Bayesian policy gradient is a new class of gradient estimation methods pioneered by Ghavamzadeh and Engel[7]. They argue that the previous gradient estimators follow a frequentist approach while a Bayesian method would also consider the uncertainty associated with variations in exploration. They consider the posterior over the expected return given the observed rollouts D:

\[
\nabla E(J(\theta) \mid D) = \mathbb{E} \left( \int R(\tau) \frac{\nabla p_\theta(\tau)}{p_\theta(\tau)} p_\theta(\tau) d\tau \mid D \right). \tag{9}
\]

The model used in this paper corresponds to “Model 1” in their paper and is suited to problems without error in the observed reward.

The advantage of using a Bayesian approach is that it incorporates uncertainty associated with limited gradient exploration, and thus results in better gradient estimates.

IV. IMPLEMENTATION TESTING

The four policy gradient methods presented above are compared on two standard small-scale problems, a continuous-action bandit problem and a continuous state and action linear quadratic regulator (LQR) problem as covered by Ghavamzadeh and Engel[7]. This allows for the comparison of the four approaches to develop an intuitive understanding of their behavior prior to the more challenging and complicated autonomous driving problem.

A. Bandit Problem

This simple problem allows for the comparison of estimated gradients where the true value is known. The bandit problem consists of a single state and continuous one-dimensional action space \( A = \mathbb{R}^1 \). Paths generated over this system have a depth of one and contain a single action. Two reward functions, \( r(a) = a \) and \( r(a) = a^2 \) are considered, each with analytically tractable gradients. It can be easily shown that the optimal action tends to either extreme in the first case and to positive infinity for the second.

The policy considered here has the form \( a \sim \mathcal{N}(\theta_1, \theta_2^2) \) and the gradient is estimated for \( \theta = [0, 1]^T \). The likelihood ratio \( u(\tau) = \sum_{t=0}^{T} \nabla \log p_\theta(a_t \mid s_t) \) is given by \([a, a^2 - 1]^T \) and the Fisher information matrix is \( F = \text{diag}(1, 2) \).

Table I shows the results of estimating the gradient using each algorithm and 20 samples on the two reward functions. The average gradient from \(10^4\) runs along with their standard deviations are listed. The true gradient is listed under “Exact”. Both the natural and Bayesian methods were run assuming the Fisher information matrix was known.

<table>
<thead>
<tr>
<th>Reward</th>
<th>Exact</th>
<th>FiniteDiff</th>
<th>REINFORCE</th>
<th>Natural</th>
<th>Bayesian</th>
</tr>
</thead>
<tbody>
<tr>
<td>( r(a) = a )</td>
<td>( 0.00 \pm 0.00 )</td>
<td>( 0.02 \pm 0.00 )</td>
<td>( 0.00 \pm 0.00 )</td>
<td>( 0.00 \pm 0.00 )</td>
<td>( 0.00 \pm 0.00 )</td>
</tr>
<tr>
<td>( r(a) = a^2 )</td>
<td>( 0.00 \pm 0.00 )</td>
<td>( 0.00 \pm 0.00 )</td>
<td>( 0.00 \pm 0.00 )</td>
<td>( 0.00 \pm 0.00 )</td>
<td>( 0.00 \pm 0.00 )</td>
</tr>
</tbody>
</table>

The results indicate a high degree of variance among the gradient estimates for the non-Bayesian methods. The Bayesian method, on the other hand, produces highly-accurate gradient estimates in both magnitude and direction with little variance. Note that neither REINFORCE nor the natural policy method could gain an advantage over the finite difference method from knowledge of gradients evaluated through long rollouts.

B. Linear Quadratic Regulator

The gradient methods were also used to solve a simple LQR problem. This problem has a horizon of 20 steps.

\[
\begin{array}{ll}
\text{System} & \text{Policy} \\
\text{Initial State: } s_0 \sim \mathcal{N}(0.3, 0.001) & \text{Actions: } a_t \sim \mathcal{N}(\lambda s_t, \sigma^2) \\
\text{Reward: } R(s, a) = -s^2 & \text{Parameters: } \theta = [\lambda, \sigma]^T \\
\text{Transition: } s_{t+1} \sim \mathcal{N}(s_t + a_t, 0.01) & \text{Initial Parameters: } \theta_0 = [0.0, 1.0]^T
\end{array}
\]

The optimal solution to the problem is given by \( \theta = [-1, 0]^T \). The Fisher information matrix for this problem is unknown and was estimated using rollouts following equation 8.

The results of applying each gradient estimation algorithm to the LQR problem are shown in figure 1. The finite differences was conducted with \( \epsilon = 0.1 \) and had its rollout count adjust such that each iteration used the same number of rollouts, 10, as did the other methods. The shaded regions correspond to the maximum and minimum reward out of all trials, respectively.

![Fig. 1: Convergence of each policy gradient algorithm on the LQR problem with 10 rollouts](image)

The REINFORCE and finite difference methods performed equally well, with Natural policy gradients and the Bayesian method similarly under performing. Both of these methods require the Fisher information matrix to compute the derivative, which here had to be estimated using only the 10 rollouts. The extremely poor performance of the Bayesian method leads the author to believe there is an implementation issue.
An investigation of gradient evaluation between each algorithm was conducted for $\theta = [0,1]^T$. The averaged results from 100 estimations are listed in Table II. The gradient estimations of finite differences and REINFORCE are comparable, though REINFORCE has greater variance. Both the natural and Bayesian policy gradients are pointed in the correct direction but are significantly underestimated. Their relative variation is much higher.

<table>
<thead>
<tr>
<th>TABLE II: gradient estimates for the LQR problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>FiniteDiff</td>
</tr>
<tr>
<td>$(0.04 \pm 0.86)$</td>
</tr>
<tr>
<td>$(-17.85 \pm 1.28)$</td>
</tr>
</tbody>
</table>

V. Problem Formulation

The driving problem considered here consists of an ego-vehicle and three other vehicles, two in front of the ego vehicle and a single one behind. The task of the ego vehicle is to maintain safe separation distances while compensating for the delayed reactions of each vehicle to its surroundings.

The desired speed for each vehicle was set to $0.5 \text{m/s}^{-1}$ greater than the speed of the next car. The vehicles are spaced in 30m increments with initial speeds equal to their desired speeds plus noise with standard deviation $\sigma = 0.2 \text{m/s}^{-1}$. The history is initialized by assuming each vehicle travelled with zero acceleration to reach its present position.

<table>
<thead>
<tr>
<th>TABLE III: Randomly initialized driver behavior parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau \in \mathcal{U}(0.5, 1.5) \text{ sec}$</td>
</tr>
<tr>
<td>$\alpha_{\text{accel}} \in \mathcal{N}(9.21, 1.0)$</td>
</tr>
<tr>
<td>$\beta_{\text{accel}} \in \mathcal{N}(-1.67, 0.1)$</td>
</tr>
<tr>
<td>$\gamma_{\text{accel}} \in \mathcal{N}(-0.88, 0.1)$</td>
</tr>
</tbody>
</table>

Dynamics Model

The action space consists of a constant acceleration $a$ placed between time $t$ and $t+1$ with a timestep $\Delta T$ of 10Hz. Each actor is deterministically propagated over the timestep:

$$x_{t+1} = x_t + u_t \Delta T + \frac{1}{2} a_t \Delta T^2$$

$$u_{t+1} = u_t + a_t \Delta T$$

After propagation each vehicle is checked for collision. Colliding vehicles have their crashed indicator set to true and subsequently decelerate proportional to their speed, $a_{\text{crashed}} = -0.15 u_t$.

A. Behavior Model

The human behavior model literature has identified three primary forms of longitudinal driving behavior: driving under free-flow in which a driver tracks their desired speed, car following in which one maintains a safe distance from the leading car, and emergency braking in which one avoids an imminent collision[8].

In this work we use the General Motors nonlinear model[9] for car following. The acceleration of agent $i$ following agent $j$ is given by

$$a_i(t) = \alpha \frac{u_i(t-\tau)^\beta}{\Delta X_{i,j}(t-\tau)^\gamma} \Delta U_{i,j}(t-\tau) + \varepsilon$$

with noise term $\varepsilon$ here drawn from $\mathcal{N}(0, 0.238) \text{ m/s}^2$. The parameters come from a study by May and Keller[9], with different parameters $\alpha$, $\beta$, and $\gamma$ depending on whether the car must accelerate or decelerate. The values given for these values in Table III assume distance measured in feet.

Free-flow driving occurs when an agent merely tracks its desired speed. A simple proportional controller with the same stochastic error was used

$$a_i(t) = (u_{\text{des}} - u_i(t-\tau)) K + \varepsilon.$$  

Emergency behavior was only available to the ego vehicle, and was modelled as a sharp acceleration at $2 \text{m/s}^{-2}$ or deceleration at $3 \text{m/s}^{-2}$ to avoid collision. Sufficient variation in behavior was captured in the random initialization of the driving parameters.

Simulation of 400 time steps (40s) were conducted and simulated a hazardous highway case in which the front vehicle sharply decelerates. The front vehicle was forced to decelerate at $1.5 \text{m/s}^{-2}$ between steps 40 and 80. Response delay inherent in the driving model causes somewhat difficult-to-predict behavior for the trailing vehicles. The ego vehicle must therefore take care to maintain a safe distance from the car in front while taking care not to act too drastically to avoid a rear-end collision.
B. Policy Parameterization

The policy parameterization considered was a softmax probability distribution between free-flow, car following, and emergency behavior based on a linear feature weighting. Each action had weight \( \phi^T \omega_i \) with feature vector \( \phi = [1, a_{\text{ref}} \log(\Delta X_{\text{front}})] \) where \( a_{\text{ref}} \) is the constant acceleration required to avoid a frontal collision assuming constant velocity of the other vehicle. The probability of choosing action \( a_i \) under softmax is given by

\[
P(a_i) \propto e^{\lambda \phi^T \omega_i}.
\] (14)

The complete policy parameterization vector is given by \( \theta = [\omega_{\text{emergency}}, \omega_{\text{follow}}, \omega_{\text{free-flow}}, \lambda] \) with a total of ten parameters. Once the action class is chosen the actual acceleration is determined using the nominal driver parameters for that action class with zero lag time \( \tau = 0 \) and without noise.

This parameterization scheme was chosen due to its ease of implementation, its interperability, and the potential for straightforward extension via the inclusion of additional features and action classes. One immediate limitation is lack of knowledge of the trailing vehicle, which ends up being a hinderance.

C. Reward Function

The reward function specifies the tradeoff between safety and comfort. For simplicity, it is convenient to formulate the reward function as a linear combination of specific factors:

\[
R(s,a) = R_{\text{crash}}(s,a) + R_{\Delta \text{speed}}(s,a) + R_{\text{accel}}(s,a) + R_{\text{alive}}(s,a) + R_{\Delta X}(s,a)
\]

with \( R_{\text{crash}} = -10^5 \) a penalty for a collision with the ego vehicle, \( R_{\Delta \text{speed}} = -0.01 |u_{\text{des}} - u| \) a penalty for deviating from the desired speed, \( R_{\text{accel}} = -0.1a \) a penalty for acceleration, \( R_{\text{alive}} = 1.0 \) a bonus given only if the ego car has not yet crashed, and \( R_{\Delta X} = \min(-10 + 0.5 \Delta X, 0) \) a penalty for being within 20m of another vehicle.

VI. EXPERIMENTAL RESULTS

The one-dimensional car driving problem was solved with five methods: gradient ascent via finite differences, gradient ascent via REINFORCE, cyclic coordinate ascent via the bisection method, and value iteration conducted on a discretized version of the original problem. Each method was applied to the problem and compared using 10^4 rollouts using the derived policies. All methods involving rollouts used a depth of 400. An additional baseline policy, “Car Follow” was evaluated as well, consisting of always taking the car-following action. A summary of their respective performances are given in table IV.

Cyclic coordinate ascent is a straightforward function optimization algorithm which successively maximizes a function \( f(x) \) along each of its coordinates individually. It makes up for its often slow convergence with reliability and ease of implementation. The method used here used a bisection approach in which the upper and lower bounds for each parameter were evaluated, and the better of the two chosen as the new bounds coupled with a midpoint evaluation. Each parameter could vary between \( \pm 5 \) and bisection was run to a depth of 8 resulting in solutions with granularity of \( \approx 0.04 \). Note that due to the stochastic nature of the problem a perfect solution is not guaranteed.

Value function iteration is a straightforward algorithm for solving discrete state and action MDPs by repeatedly applying the Bellman equation:

\[
Q(s) = \arg \max_a \left\{ R(s,a) + \gamma \sum_{s'} T(s,a,s')Q(s') \right\}
\] (15)

where \( Q(s) \) is the expected utility obtained by following the optimal policy at state \( s \). The optimal policy is obtained directly from the expected value function.

Value iteration requires that the state and action space be discretized. The state and action space discretizations are summarized in table V. Note that this formulation also lacked information about the training vehicle.

<table>
<thead>
<tr>
<th>parameter</th>
<th>min</th>
<th>max</th>
<th>bincount</th>
<th>notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>( s_u )</td>
<td>60</td>
<td>70</td>
<td>10</td>
<td>velocity ( \text{m} \text{s}^{-1} )</td>
</tr>
<tr>
<td>( s_{\Delta X} )</td>
<td>0</td>
<td>50</td>
<td>10</td>
<td>distance to car in front ( \text{m} )</td>
</tr>
<tr>
<td>( s_{\Delta V} )</td>
<td>-2</td>
<td>2</td>
<td>10</td>
<td>velocity difference of car in front ( \text{m} \text{s}^{-2} )</td>
</tr>
<tr>
<td>( s_{\text{alive}} )</td>
<td>T</td>
<td>F</td>
<td>2</td>
<td>whether the ego car has crashed</td>
</tr>
<tr>
<td>( a )</td>
<td>-2</td>
<td>2</td>
<td>5</td>
<td>applied acceleration ( \text{m} \text{s}^{-2} )</td>
</tr>
</tbody>
</table>

The discrete transition and reward functions were estimated using rollouts with randomized actions seeking to maximally explore the space. A total of \( 10 \times 10^5 \) rollouts were used, taking 20 minutes to extract. The optimal policy was computed using value iteration with a discount factor \( \gamma = 0.9 \) and termination condition \( ||Q(s_k) - Q(s_{k+1})||_\infty \leq 0.01 \).

The Natural and Bayesian gradient estimation methods were found to produce nonsingular Fisher information matrices, even with large rollout counts \( 10^4 \) per step. This prevented their practical use. Several methods exist for improving the estimation, including sparsification of the kernel matrix \( K \) in the Bayesian approach.[2]

Results indicate that the REINFORCE method does outperform the finite differences approach. The reward function suffers from high variation; the nature of the problem is such that collisions can be inevitable given poor initialization of the driver parameters. Cyclic bisection produces the best policy
with both the lost average reward and the lowest variance but at the cost of increased computation time. Qualitatively, the solution found by cyclic bisection uses car-following for the majority of the time and issues emergency braking when $a_{req}$ has large magnitude.

The straightforward policy of always choosing the car follow action performs fairly well. This can be expected; the ego car is always following a slower lead vehicle and would only need to apply braking when very close to the other vehicle. The simple nature of this solution and the relatively small improvements available by more sophisticated methods requires further investigation, it is likely that a better policy parameterization or a better problem with a richer solution space could be obtained.

Value iteration takes a relatively short about of time to solve as the discretized state space is somewhat small ($10^3$ states, 5 actions). The result, however, is extremely poor. This is likely due to the coarse discretization coupled with poor exploration of the state space during rollout.

VII. CONCLUSION

This paper investigated policy gradient methods in the context of highway driving. An autonomous driving problem was constructed and evaluated using a variety of solution methods. The experimental results are marginaly encouraging, but high gains are conjectured subject to proper implementation of the natural and Bayesian policy gradient algorithms. The problem solved proved to be uninteresting. It is recommended that future work investigate better policy parameterizations and extend to problems where a good solution is less apparent.

REFERENCES