KLearn: Stochastic Optimization Applied to Simulated Robot Actions Final Report

Kevin Watts
Willow Garage
Menlo Park, CA
watts@willowgarage.com

Abstract

Machine learning techniques and algorithms are prevalent in robotics, and have been used for computer vision, grasping, and legged walking. Reinforcement learning approaches have been developed over the past 15 years, with modern techniques using continuous action spaces for various robotic applications. Policy gradient learning allows various optimization techniques to quickly optimize robotic tasks, but gradient-based optimization can suffer from problems with stochastic objective functions. KLearn uses a stochastic optimization system to optimize the mean of the objective function. The optimization algorithm is applied to a simulated robot.

1. Introduction

Reinforcement Learing (RL) has been used to control robot actions and teach robot behaviors for tasks such as walking ([1]), playing pool and grasping. One approach to optimizing various robotic behaviours is to use policy-gradient learning, using gradient ascent to optimize a value function in policy space. For systems with stochastic output, it can be difficult or impossible to correctly determine a gradient. In this project, I have developed a novel stochastic optimizer which can be applied to reinforcement learning and applied it to an application with a simulated robot.

A stochastic optimization system has many uses in reinforcement learning. Some robotics applications may have process noise or unmodeled variables. In other cases, the observations may be noisy. For this problem, numerical stability in the robotics simulator introduces process noise into the system. Optimization methods designed for deterministic systems, such as the conjugate gradient method, do not converge for this problem. Using an optimization algorithm designed for stochastic inputs allows us to avoid sampling our objective function many times to find an average estimate of the value.

The stochastic optimization system has been applied to the example problem of rolling a ball across a table using a simulated PR2 robot (Figure 1). The remainder of this paper describes prior work in reinforcement learning and stochastic optimization, system architechure, the optimization algorithm, and the application results.

2. Prior Work

Reinforcement learning traditionally describes an agent attempting maximize a value or reward function by choosing actions. The environment or world can be represented as discrete states, with a Markov process controlling transition probabilities between different states. Research areas can include exploration versus exploitation tradeoffs and value function convergence [2].

More recent reinforcement learning work involving robots has combined learning from demonstration and reinforcement learning to teach new tasks to robots. The reinforcement learning framework no longer uses discrete state spaces, but may use policy gradient learning ([3]). Recent work, from IROS 2011, uses previous sensor data as direct feedback to robot commands, allowing for robust grasping

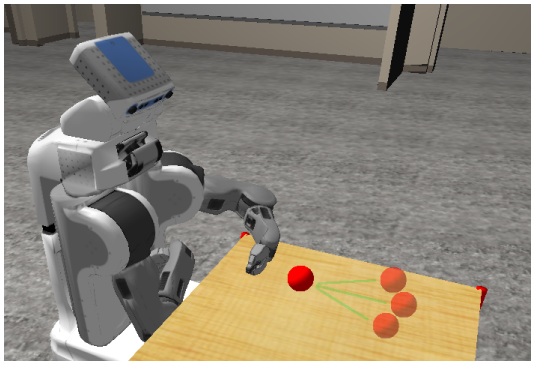

Figure 1. PR2 ready to roll a ball in a simulated world. Re-running the simulation with identical control inputs can give very different results.

in the prescence of poor object detection [4].

Stochastic function optimization has been studied by many authors since the 1950's ([5]). In 1951, Robbins and Monro introduced a root finding algorithm for the expected value of a stochastic function ([6]). They proved convergence (with a given probability) for a bounded, monotonic, differentiable function, and used a decreasing step size. Later authors noted improved convergence by changing the learing rate ([7].

In 1952, Kiefer and Wolfowitz introduced a method for optimizing stochastic functions that used a gradient method to optimize ([8]). Like Robbins and Monro, it used a learning rate that slowly anneals. Kiefer and Wolfowitz proved convergence of their algorithm for a concave function with a unique maximum.

For this project, I chose not to use the Kiefer-Wolfowitz algorithm because I wanted to try an algorithm that used variance information from neighboring points, and use expected improvement (or more than just mean estimation) to determine the next sample point. Future work in this area could compare a Kiefer-Wolfowitz approach with the optimization algorithm used in this project.

An algorithm we studied in class, "stochastic gradient descent" ([9]), would not be appropriate for this application. Stochastic gradient descent uses the approximate gradient for each training sample to approach an optimum value. Although that may give interesting results for this problem, it could also be confused by function noise from a single function evaluation. An approach that may be promising is modifying the algorithm to use a small subset of training data, instead of a single sample, or to account for a likelihood based on the estimated variance of the function.

3. Stochastic Optimizer

Optimization algorithms like the conjugate gradient method do not converge for stochastic functions. As a replacement, I'm using a novel stochastic optimization algorithm. In designing this algorithm, I assumed that my function had random output independent of parameters, but that neighboring points would be close in value to their neighbors.

repeat
  Sample values around point P
  for p_i near P **do**
    Evaluate function at point
  end for
  for p_i near P **do**
    Calculate expected improvement
  end for
  Set P to p_i with best expected improvement
until converged

Values around point P are chosen using all $2n$ points which are $\pm\epsilon$ away from P along the coordinate axes. Therefore we have $2n + 1$ function evaluations for each iteration. In this implementation, I allow the function to be evaluated multiple times at a single point. Multiple evaluations give us more confidence in our mean and variance estimates.

3.1. Expected Improvement

To calculate the expected improvement near any point, I use the sample mean and variance of the function near that point and calculate the expected improvement by assuming a Gaussian distribution, similar to the "Gaussian Processes method" of Frean and Boyle [10]. Equation 1 gives us the expected improvement at a point p_i above value V given the mean and variance of our equation at that point.

$$\begin{aligned} u &= \frac{\mu - V}{\sigma} \\ \Phi &= \frac{1}{2} erf(\frac{u}{\sqrt{2}}) + 0.5 \\ \phi &= \frac{1}{\sqrt{2\pi}} \exp(-\frac{u^2}{2}) \\ EI &= \sigma(u\Phi + \phi) \end{aligned} \tag{1}$$

3.2. Mean and Variance

Mean and variance are calculated at the point P using the weighted average point all function evaluations at that point, with neighboring points weighted less. Variance is averaged with a high "base variance", σ^2_{base}, which penalizes having few samples of a function at a particular point (this value must be set higher than the function variance). The weight of the base variance versus the sample variance is determined by β, and τ determines the relative weight of the nearby samples when calculating our sample mean and variance. See Equation 2.

For this problem, τ was set so points ϵ away from our test point counted half as much as samples at the point, and β was set such that 5 samples at a point gave a variance weighted evenly between the base variance and the sample variance.

$$
\begin{aligned}
w_i &= \frac{1}{1+\tau||p_i - P||_2^2} \\
\mu &= \frac{\sum_{i=0}^{m} w_i x_i}{\sum_{i=0}^{m} w_i} \\
\sigma_{local}^2 &= \frac{\sum_{i=0}^{m} w_i (x_i - \mu)^2}{\sum_{i=0}^{m} w_i} \\
W_{local} &= \exp(\frac{1}{\beta} \sum_{i=0}^{m} w_i) \\
\sigma^2 &= W_{local}\sigma_{base}^2 + (1 - W_{local})\sigma_{local}^2 \qquad (2)
\end{aligned}
$$

3.3. Termination

The algorithm terminates when one of the following conditions are met:

Max Iterations Configured through a user-specified max functions parameter.

No Improvement Neighboring points do not have better expected improvement than our starting point.

Similar Values Using the Kullbeck-Leibler divergence, the observed mean and variance of P and the last value of P are compared to see if we have plateaued.

This stochastic function optimizer conducts a local search, so it can get stuck in a local minimum. One significant downside to this optimizer is that it travels slowly from test point to test point, only moving by ϵ each time. If the optimizer was modified to use gradient information to "jump" farther to the next test point, it could converge faster. This is an opportunity for future work.

3.4. Sample Optimization Problem

The optimizer was applied to a sample optimization project for verification. In this case, the utility function was a two-dimensional quadratic bowl, with Gaussian noise added. The unmodified quadratic is quite easy to optimize.

From Figure 5, we can see the optimizer starts out at $(-3, 3)$ and approaches the origin. The optimization algorithem does not reach the true minimum, since it is confused by function noise. In this case, the difference between the final value and the true minimum is one quarter of the standard deviation of the noise.

4. Simulation System Architecture

KLearn uses the Gazebo ([11]) robot simulator to model the Willow Garage PR2 robot. The Gazebo robot simulator provides instrumentation for the the world, so the algorithm can use ground-truth information about object pose and velocity data directly in the utility function.

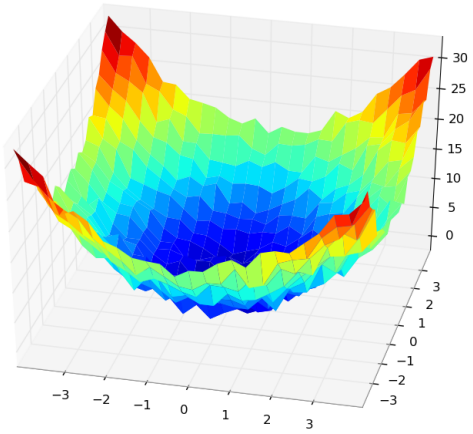

Figure 2. Quadratic function with added Gaussian noise.

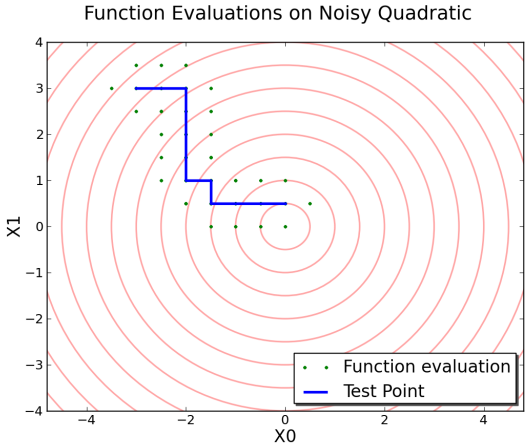

Figure 3. Optimizer function evaluations for stochastic quadratic. Optimizer path is in blue. Green points are evaluated when neighboring points are checked. Function level curves are in red.

The PR2 robot is almost fully-simulated in Gazebo. Mechanism kinematics and dynamics data are taken directly from design documents, and all cameras and laser sensors are simulated. At a 500Hz physics update rate, the simulator can get relatively good convergence for simple rigid body dynamics, such as grasping a cup on a table.

The simulated world has a ball resting on a table, directly in front of the simulated robot. The ball mass is 0.5kg, compared with roughly 200kg for the simulated PR2. The robot's arm is commanded to stay fixed during ball contact.

When using the simulator, function noise comes from numerical instability and timing jitter. Despite hours of work and significant experience with the simulator, I was not able to eliminate the process noise from the system.

The optimization framework optimized the robot actions in the simulator. The optimizer was written in Python, and ROS was used for interprocess communication.

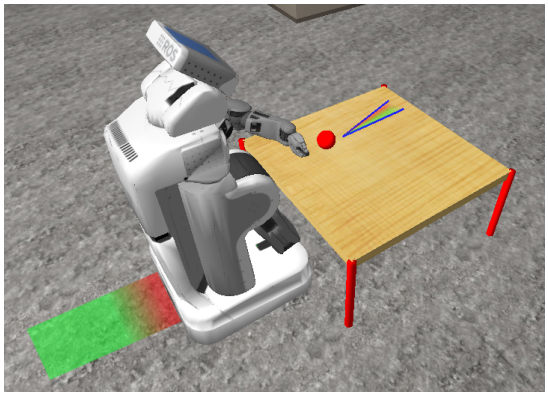

Figure 4. Utility function visuals for ball trajectory.

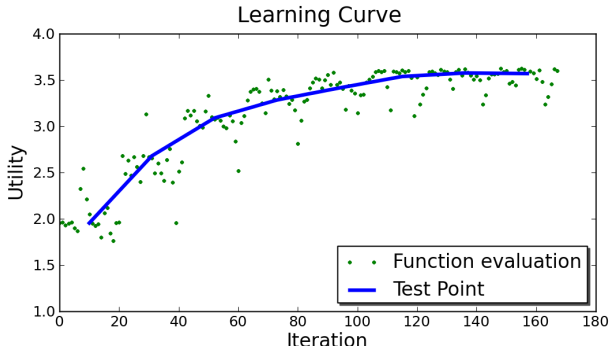

Figure 5. Optimizer utility with each iteration. Green points are all function evaluations, including gradient points around our test point. Blue line is function evaluations at each test point.

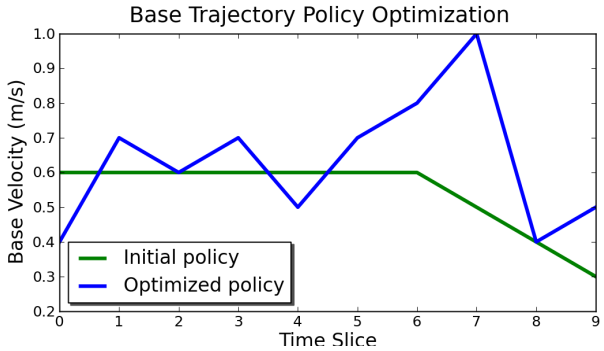

Figure 6. Optimized policy and the initial policy for robot base trajectory.

5. Robot Learning Approach

The optimizer was applied to a simulated robot pushing a ball across a table. The robot was commanded to push the ball by only moving the base of the robot, with the arm holding still. The simulation environment was reset after every trial.

Robot velocity commands were sent every 100ms for 10 time slices. Robot velocity commands were bounded to $\pm$1.0m/s, which is the maximum allowable speed of the PR2. Velocity commands were only sent for the forward direction. The initial base velocity was 0.6m/s for every time slice, slowing down by 0.1m/s per slice for the last three slices.

Penalty/Utility output description:

Hitting the Table Hitting the table with the robot will cause a significant penalty, making any parameters that cause the robot to hit the table infeasible.

Ball Trajectory Angle The angle of the ball trajectory from straight.

Ball Velocity Utility function increased with ball velocity forward.

6. Results

As we can see in Figure 5, the optimizer evaluated the function over 170 times. At 21 function evaluations per iteration (for checking neighboring points), this works out to over 16 cycles through our optimizer main loop. For our 10 dimensional problem, this is less than $2n$ iterations and $20n$ evaluations.

Our optimized policy, in Figure 6, shows that the robot's base velocity reaches the maximum of $1.0m/s$ at timestep 7, right at ball contact. The policy then ramps down the robot velocity quickly, to avoid hitting the table.

7. Conclusion

This project demonstrated the application of a stochastic optimization algorithm to a simulated robot application. The optimizer works for noisy, 10 dimensional systems. The stochastic optimization algorithm almost doubles the achieved utility in the simulated robot application.

7.1. Future Work

Future work can involve improvements to the optimizer. One possible improvement to this algorithm, not explored here, is to iterate the optimizer many times near the optimal point to get better estimates for mean and variance. This can help reduce "tail risk" in any optimized solution. Solutions near an optimum may also be near a constraint, and for some applications it can be important to guarantee, to with some probability, that the solution will be feasible or within some range.

The optimization algorithm in this problem works for the examples of the stochastic quadratic and the simulated robot. It would be worth comparing the algorithm to that method of Keifer-Wolfowitz, or similar modern methods. Applying a learning rate to vary the step size, used in the Keifer-Worfowitz algorithm, could improve convergence

dramatically in cases where the initial guess is far from the optimum.

Lastly, the learning approach needs to be tested on a real-world robotics problem. We would need to determine whether it is necessary to model real-world problems as stochastic, or whether the observed noise works with the developed optimizer.

References

[1] M. Kalakrishnan, J. Buchli, P. Pastor, M. Mistry, and S. Schaal, "fast, robust quadruped locomotion over challenging terrain," in *robotics and automation (icra), 2010 ieee international conference on*, 2010, pp. 2665–2670. [Online]. Available: http://www-clmc.usc.edu/publications/K/kalakrishnan-ICRA2010.pdf

[2] L. Kaelbling, M. Littman, and A. W. Moore, "Reinforcement learning: A survey," in *Journal of Artificial Intelligence Research 4 (1996) 237-285*, May 1996.

[3] P. Pastor, M. Kalakrishnan, S. Chitta, E. Theodorou, and S. Schaal, "Skill learning and task outcome prediction for manipulation," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, may 2011, pp. 3828 – 3834.

[4] P. Pastor, L. Righetti, M. Kalakrishnan, and S. Schaal, "Online movement adaptation based on previous sensor experiences," in *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, sept. 2011, pp. 365 –371.

[5] Wikipedia, "Stochastic approximation — Wikipedia, the free encyclopedia," 2011, [Online; accessed 13-Dec-2011]. [Online]. Available: http://en.wikipedia.org/w/index.php?title=Stochastic_approximation&oldid=464702710

[6] H. Robbins and S. Monro, "A stochastic approximation method," in *The Annals of Mathematical Statistics*, vol. 22, no. 3, Sept 1951, pp. 400 – 407.

[7] B. Polyak and A. Juditsky, "Acceleration of stochastic approximation by averaging," in *SIAM Journal on Control and Optimization*, vol. 30, no. 4, July 1992.

[8] J. Kiefer and J. Wolfowitz, "Stochastic estimation of the maximum of a regression function," in *The Annals of Mathematical Statistics*, vol. 23, no. 3, Sept 1952, pp. 462 – 466.

[9] Wikipedia, "Stochastic gradient descent — Wikipedia, the free encyclopedia," 2011, [Online; accessed 13-Dec-2011]. [Online]. Available: http://en.wikipedia.org/w/index.php?title=Stochastic_gradient_descent&oldid=465594045

[10] M. Frean and P. Boyle, "Using gaussian processes to optimize expensive functions," in *Proceedings of the 21st Australasian Joint Conference on Artificial Intelligence: Advances in Artificial Intelligence*, ser. AI '08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 258–267. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-89378-3_25

[11] W. Garage. (2011, November) Gazebo robotics simulator. [Online]. Available: http://www.ros.org/wiki/gazebo