ESE 650 Final Project
Adaptive gain-scheduling of a Raibert-Style Hopper using Q-learning

Tarik Tosun

May 2, 2014

Abstract

Q-learning was applied to improve control performance of a simulated Raibert-style hopping robot in three dimensions. Rather than learning a policy over control outputs directly, Q-learning was used to learn an a policy over controller gains as a function of state. In this report, we present the structure and methodology of the system, and show that Q-learning produces a gain-scheduling policy which outperforms hand-tuned gains in real time.

1 Background

1.1 Raibert Hopper

The Hopper is a one-legged hopping robot, developed by Marc Raibert and collaborators at Carnegie Mellon in the 1980's (Figure 1). Early research in the dynamics and control of the Hopper formed the basis for modern research in legged locomotion. In their papers, Raibert *et al.* describe an effective feedback controller scheme for the Hopper [1, 2]. The controller has three distinct parts, controlling lateral velocity of the center of mass, body attitude, and hopping height. For brevity, we present the controller for the 2d case here; it may be easily generalized to the 3d case. During the FLIGHT phase of the Hopper's gait, the body flies through the air, and the leg is positioned to stabilize and control lateral velocity:

$$(x_{foot})_d = \frac{\dot{x} T_{st}}{2} + K_{foot} (\dot{x} - \dot{x}_d)$$

$$(\phi_{hip})_d = \phi_{body} + \sin^{-1} \left(\frac{x_{foot,d}}{l_{leg}} \right)$$

Here, $(x_{foot})_d$ is the necessary foot placement relative the body to maintain the desired velocity, and $(\phi_{hip})_d$ is the corresponding hip angle to achieve correct placement. Hip joint velocity is controlled with a standard PD controller:

$$\dot{\phi}_{hip} = K_{p,flight} * \left((\phi_{hip})_d - \phi_{hip} \right) - K_{d,flight} * \dot{\phi}_{hip}$$

During the STANCE phase, the Hopper's foot is in contact with the ground, and it may correct the attitude of body by moving the hip joint. Note that this is impossible during the flight phase, as angular momentum will always be conserved. Attitude control is implemented with a standard PD controller. We assume that the desired attitude of the body is always zero.

$$\dot{\phi}_{hip} = -K_{p,stance} * \phi_{body} + K_{d,stance} * \dot{\phi}_{body}$$

Hopping height is regulated by applying a constant thrust force during every STANCE phase. Some frictional losses are present in the system, so by conservation of energy the hopper will bound to a unique stable height when the energy added by the thrust equals the losses of a single bound.

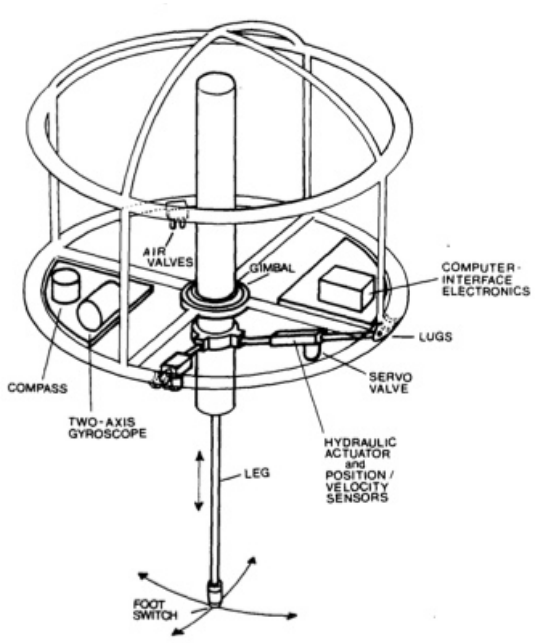

Figure 1: Schematic of the Raibert Hopper [1]

1.2 Q-learning

Q-learning is a temporal-difference learning (TD) algorithm. TD methods are a form of reinforcement learning that combine Monte-Carlo techniques (MC) and Dynamics Programming (DP). The goal is to select a control policy π that will maximize the performance of a system as measured by some reward function $R(s)$. In the TD estimation problem, we estimate the expected total value of policy π at state s as:

$$\begin{aligned} V^{\pi}(s) &= E_{\pi}\left\{R_t \mid s_t = s\right\} \\ &= E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s\right\} \\ &= E_{\pi}\left\{r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} \mid s_t = s\right\} \\ &= E_{\pi}\left\{r_{t+1} + \gamma V^{\pi}\left(s_{t+1}\right) \mid s_t = s\right\} \end{aligned}$$

We have obtained a recursion for $V^{\pi}(s)$, which we may exploit through dynamic programming. Rather than performing full back-ups (estimating the expected value by inspecting all successor states s_{t+1}), in TD we perform a sample back-up, using our currently cached value of $V^{\pi}\left(s_{t+1}\right)$ as our estimate. In this way, TD combines elements of MC and DP techniques. At each time step, we perform the update:

$$V(s_t) \leftarrow V(s_t) + \alpha\left(r_{t+1} + \gamma V(s_{t+1}) - V(s_t)\right)$$

Where $\alpha \in (0, 1]$ is the learning rate. Given enough time and a small enough step size, this TD algorithm is guaranteed to converge to V^π in the mean for a fixed policy π [5].

In Q-learning, we aim to solve the TD control problem, selecting an optimal policy $\pi*$. The Q-function is defined over pairs of states and actions, and represents the total expected value of a state-action pair, assuming that we will follow the (originally unknown) optimal policy $\pi*$. The update equation is:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left(r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right)$$

Q-learning is an off-policy method, meaning that we may run it without being tied to any particular control policy. Consequently, we may run it online or train on batches of data. To act under the optimal policy, we simply select $\arg\max_a Q(s_{t+1}, a)$, the action which maximizes Q. To continue learning as we run the policy, we may take an ϵ-greedy approach, selecting a random action with some probability ϵ.

There is another algorithm, known as sarsa, which is the on-policy version of Q-learning. The update rule is very similar:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left(r_{t+1} + \gamma Q(s_{t+1}, a) - Q(s_t, a_t) \right)$$

Rather than updating with $\max_a Q(s_{t+1}, a)$, we simply select the Q value of the state-action pair we actually chose at time t+1. This can be advantageous in practice because it will account for the stochasticity of an ϵ-greedy policy.

2 Approach

The Raibert controller was implemented as described in [1] for control of a 3D Hopper simulated in Matlab. After some hand-tuning, good performance was achieved over a wide range of initial conditions (initial body attitudes) using gains $K_{foot} = 0.15$, $K_{p,att} = 30$, $K_{d,att} = 0.1$. The goal was to use Q-learning to generate an optimal gain-scheduling policy which would improve the performance of the Hopper as measured by the following reward function:

$$r(t) = \begin{array}{ll} 1 - \lambda \left(\theta^2 + \phi^2 + \dot{x} + \dot{y} \right) & \text{if standing} \\ 0 & \text{if fallen over} \end{array}$$

Here, λ is a reward shaping parameter controlling the penalty placed on state error, and provides some reward gradient information when the robot is not about to fall over, as described by Tedrake [3]. In all experiments, $\lambda = 0.05$ was used.

A limited set of state variables was used for Q-learning, to reduce the dimensionality of the learning problem. The state variables were: $\{\dot{x}, \dot{y}, \theta_{body}, \phi_{body}, \theta_{hip}, \phi_{hip}\}$, and the control policy parameters were: $\{K_{foot}, K_{p,att} K_{d,att}\}$, giving a nine-dimensional state-action space.

2.1 Radial Basis Functions

The canonical formulations of TD learning problems assume a discrete state-action space. For control of a dynamical system like the Hopper, a continuous representation is preferable. Radial basis functions were used as a finite-dimensional representation of the continuous state-action space. N identical isotropic gaussians were distributed over the state-action space to approximate the continuous Q-function as follows:

$$Q(s, a) = \sum_i^N k_i(s, a) Q_i$$

$$k_i(s,a) = \frac{1}{\sqrt{(2\pi)^9\sigma^2}} \exp\left(\frac{-1}{2\sigma^2}(\underline{\mathrm{x}} - \underline{\mu}_i)\right)$$

The corresponding update rule is:

$$Q_i \leftarrow Q_i + \alpha_i \left\{ r_{t+1} + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q_i \right\}$$

$$\alpha_i = \alpha k_i(s_t, a_t)$$

The centers $\underline{\mu}_i$ of the gaussian kernels were placed by performing K-means clustering on the training data. To ensure that the set of gaussians could appropriately cover the state-action space, each dimension of the training data was mean-centered and scaled to unit variance before performing batch Q-learning. In all experiments presented here, $N = 200$ kernels were used, with variance $\sigma^2 = 25$. Literature indicates that wide radial basis functions tend to produce better results in reinforcement learning [4].

3 Experiments

Simulations were performed with combinations of 10 random initial conditions and 10 random controllers, for a total of 1000 training episodes each of length $t = 500$. A test set of 30 random initial conditions (different from those in the training set) was prepared. Control policy performance is measured as the average total reward per episode (ATR) over the test set, with a maximum episode length of $t = 500$. As a baseline, the human-tuned controller with parameters $K_{foot} = 0.15$, $K_{p,att} = 30$, $K_{d,att} = 0.1$ was tested, and scored an ATR of 413.73. In all experiments, batch Q-learning was performed using reward discount $\gamma = 0.9$.

In the first experiment, the learning rate $\alpha(i) = 0.8 * exp(-i)$, where i is the iteration number, for 5 iterations over the entire batch of 1000 episodes. Figure 2 plots test set ATR for each of the five iterations. In the second experiment, a constant learning rate of $\alpha(i) = 0.01$ was used for 50 iterations. Figure 3 plots test set ATR over the 50 iterations. In both experiments, a simulation timestep of 0.01 seconds was used, so that each experiment represented a maximum 5 seconds of simulation time. Running a simulation while selecting optimal control gains based on the Q-table took about 3 seconds, so the Q-learning control could be run in real-time on an actual robot.

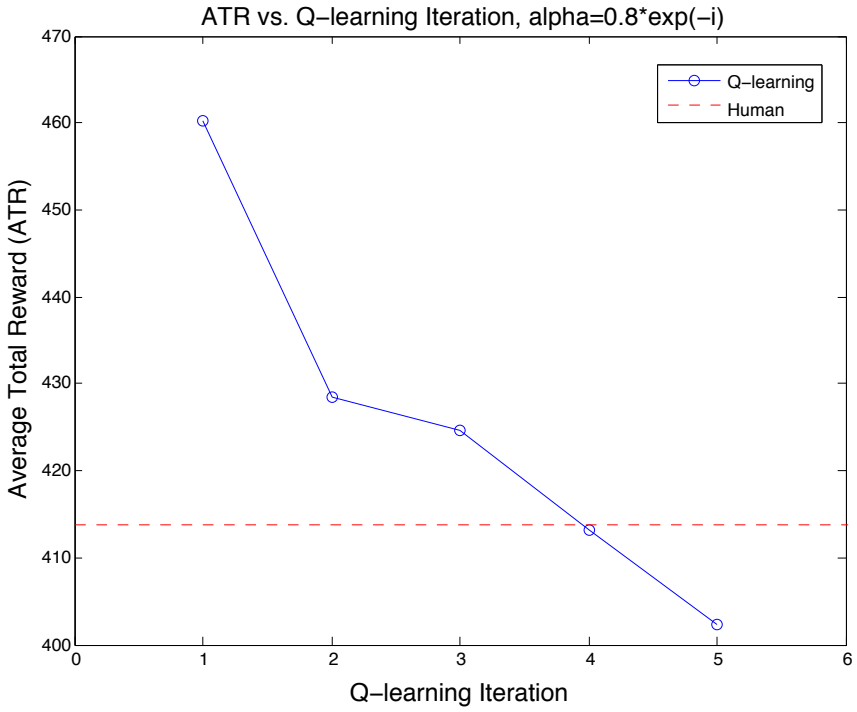

Figure 2: ATR vs. Q-learning iteration, $\alpha(t) = 0.8\exp(i)$, 5 iterations

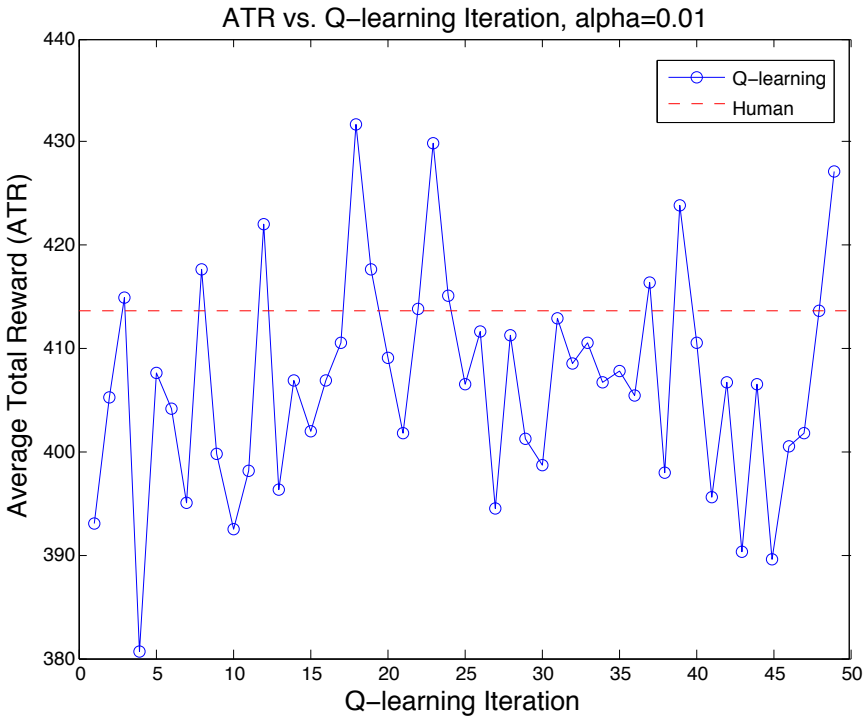

Figure 3: ATR vs. Q-learning iteration, $\alpha(t) = 0.01$, 50 iterations

4 Analysis and Conclusions

Q-learning produced a control policy which outperformed hand-tuned gains for the Hopper, as measured by the defined reward metric. In both experiments, repeating batch training for many iterations did not improve performance - in fact, in the first experiment it had an adverse effect on performance. This implies that the model of Q is overfit to the training data, possibly because the learning rate was to large. In an effort to avoid this, Experiment 2 used a much smaller constant learning rate (0.01). This resulted in large fluctuations in ATR over iterations, and no significant performance improvement over the course of iteration. This may imply that a step size of 0.01 was still too large, resulting in large jumps around a performance maximum without actually converging to the maximum. Future work would explore the affect of learning rate on performance, and attempt to find an optimal learning rate.

Finding a controller that outperformed the hand-tuned gains is a significant achievement for the Q-learning technique applied in these experiments, because the initial controller was already quite performant, often capable of stabilizing the robot after being dropped from initial attitudes as large as $\frac{\pi}{2.2}$. The Q-learned policy has the additional advantage of adaptability. Although it was not explored in this report, Q-learning could be run on-line on an actual robot, selecting a control policy in an ϵ-greedy fashion to continually improving policy performance. This could be very beneficial for long-term deployments, in which the robot actuators and sensors would wear over time, causing the optimal control policy to change. In such a deployment, it would likely be better to employ the sarsa algorithm rather than Q-learning, to account for the stochasticity of ϵ-greedy policy selection.

5 Running the Code

To run the code, run the script called `RUN_ME.m` in my submission folder. Provided ODE is installed and you have the appropriate mex file, you should see plots of the gains and Q-values changing as the Hopper hops.

References

[1] Raibert, M. H., Brown, H. B., Chepponis, M., Hastings, E., Koechling, J., Murphy, K. N., ... Stentz, A. J. (1983). Dynamically Stable Legged Locomotion. Pittsburgh, PA.

[2] Raibert, M. H. (1984). Hopping in legged systems — Modeling and simulation for the two-dimensional one-legged case. IEEE Transactions on Systems, Man, and Cybernetics, SMC-14(3), 451–463. doi:10.1109/TSMC.1984.6313238

[3] Tedrake, R., & Seung, H. S. (n.d.). Improved Dynamic Stability Using Reinforcement Learning.

[4] Kretchmar, R. M., & Anderson, C. W. (1990). Comparison of CMACs and Radial Basis Functions for Local Function Approximators in Reinforcement Learning (pp. 4–7).

[5] Sutton, R. S., & Barto, A. G. Reinforcement Learning: An Introduction. Cambridge, Massachusetts: the MIT Press. doi:10.1109/TNN.1998.712192