Value Iteration and DDP for an Inverted Pendulum

By:
Gregory M. Horn

CS229 Final Project
Professor: Andrew Ng
December 10, 2009

Abstract

My intention in this project was to learn about nonlinear control from a reinforcement learning standpoint. I got value iteration working on a simple gravity pendulum with torque controller by discretizing the state space and quadratically interpolating in between grid points. I couldn't apply this technique to an inverted cart-pole because of the "curse of dimensionality" so I explored differential dynamic programming (DDP) as an alternative. I built a cart-pole swing up controller with receding horizon DDP.

Discretized Value Iteration

A nonlinear dynamical system can be described as

$$\begin{aligned} x_{k+1} &= f(x_k, u_k) \\ &\approx A_k\, x_k + B_k\, u_k. \end{aligned}$$

The value function is the total cost to get from a state to a final state assuming optimal control policies are used at each time step:

$$\begin{aligned} V(x_k) &= C(x_k, u_k) + C(x_{k+1}, u_{k+1}) + C(x_{k+2}, u_{k+2}) + \cdots \\ &= C(x_k, u_k) + V(x_{k+1}) \end{aligned}$$

where $C(x, u)$ is the cost associated with a given state and action. Assuming that the value function is known the optimal control policy at state x_k maximizes $V(x_k)$:

$$\begin{aligned} u_k &= arg\max_u V(x_k) \\ &= arg\max_u \left(C(x_k, u_k) + V(x_{k+1})\right). \end{aligned}$$

A quadric cost function in u was adopted in order to derive a simple optimal policy:

$$\begin{aligned} u_k &= arg\max_u \left(C(x_k) - \frac{1}{2} u_k^T R\, u_k + V(x_{k+1})\right) \\ 0 &= \nabla_{u_k} \left(C(x_k) - \frac{1}{2} u_k^T R\, u_k + V(x_{k+1})\right) \\ 0 &= -R\, u_k + (\nabla_x V)^T \nabla_{u_k}(x_{k+1}) \\ R\, u_k &= (\nabla_x V)^T \nabla_{u_k}(A_k\, x_k + B_k\, u_k) \end{aligned}$$

$$
\begin{aligned}
R\,u_k &= (\nabla_x V)^T B_k \\
u_k &= R^{-1}\,(\nabla_x V)^T\,B_k.
\end{aligned}
$$

The technique for finding the optimal policy was taken from [1].

To find the value function the state space is discretized into states S and value iteration is applied. The value function is quadratically interpolated in between states when needed. When the optimal value field is found, a real-time controller would also interpolate the value function in deriving its policy.

$V(x) := 0;$

for n *iterations*

$V_{old} := V;$

for $\{x_k \in S\}$

$A_{ij} := \frac{\partial f(x_k,u)_i}{\partial x_j};$ *//linearize dynamics*

$B_{ij} := \frac{\partial f(x_k,u)_i}{\partial u_j};$ *//linearize dynamics*

$u := R^{-1}\,(\nabla_x V_{old}(x_k))^T\,B;$ *//compute policy at* x_j

$x' := A\,x_k + B\,u;$ *//propagate state with simulator*

$V(x') := interpolate(x', V_{old});$ *//interpolate value function at propagated state*

$V(x_k) \xleftarrow{\alpha} C(x_k, u) + V(x');$ *//update value function*

end

end

Mass on spring

The algorithm was debugged on the simple "mass on spring" system. The dynamical equation and cost function are:

$$
\begin{aligned}
\ddot{x} &= -k\,x - b\,\dot{x} + u \\
C(x,u) &= \frac{1}{2}x^T Q\,x + \frac{1}{2}u^T R\,u
\end{aligned}
$$

The algorithm worked and the learned controller found the exact same path as an LQR controller designed with MATLAB (see Figure 1).

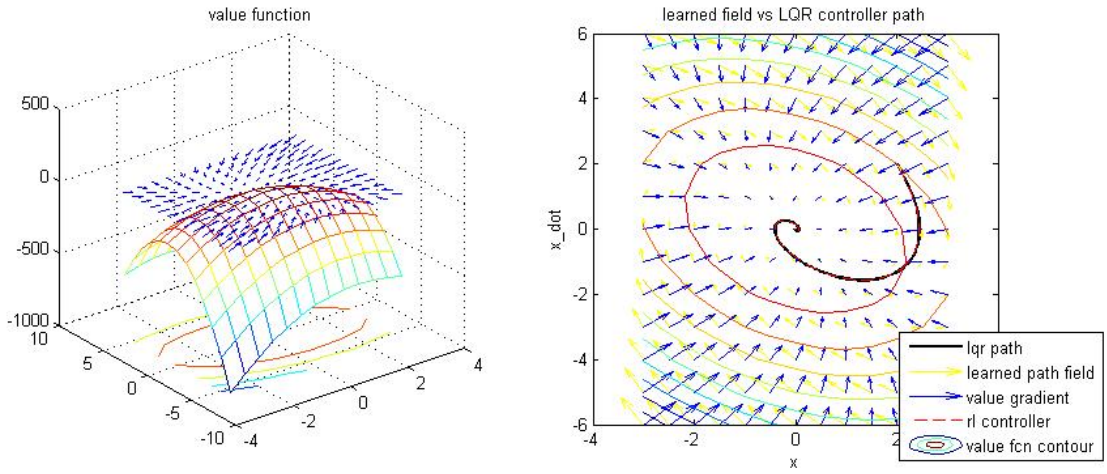

Figure 1: Mass on spring value function and learned controller

Gravity pendulum

The algorithm was then applied to controlling a simple gravity pendulum with saturating torque control. The pendulum dynamics are:

$$\ddot{\theta} = \frac{g}{l}\sin(\theta) - \frac{k}{ml^2}\dot{\theta} + \frac{u}{ml^2}.$$

The algorithm converged and the value function was learned (see Figure 2).

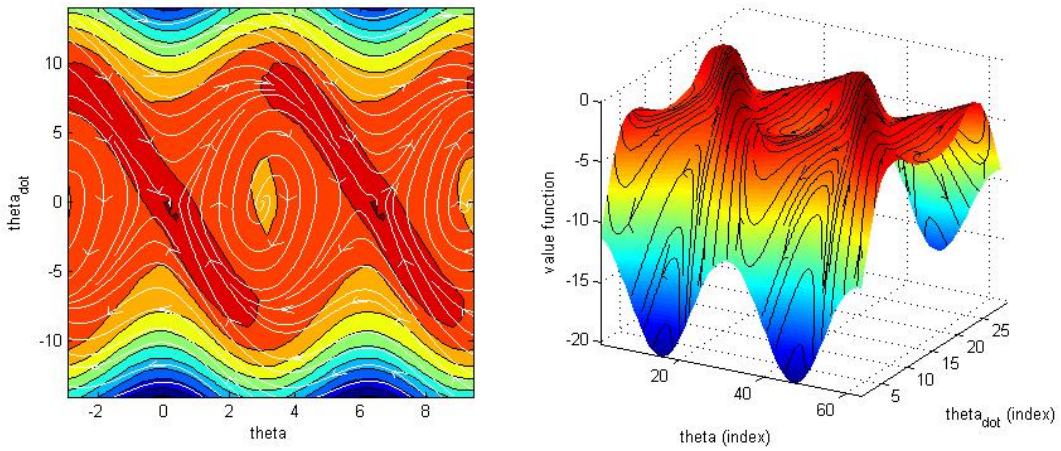

Figure 2: Value function for limited-torque pendulum. Streamlines show the phase portrait of the optimally controlled system.

Differential Dynamic Programming

I was unsuccessful in applying value iteration to a cart-pole system (largely because of the curse of dimensionality) so I tried differential dynamic programming (DDP). In DDP a path $\{(x_1, u_1), (x_2, u_2), ...(x_N, u_N)\}$

is iteratively improved upon in two steps. In the backward sweep the value function is quadratically approximated about each state in the sequence and the policy u is improved. In the forward sweep the new policy is simulated and a more optimal path x is found. This iterates until convergence.

To quadratically expand the value function, first quadratically expand the Q function (the unoptimized value function)

$$\begin{aligned} Q(x_k, u_k) &= C(x_k, u_k) + V^{k+1}(f(x_k, u_k)) && (1) \\ &\approx Q_0 + Q_x^T \delta x + Q_u^T \delta u + \frac{1}{2}\begin{pmatrix} \delta x & \delta u \end{pmatrix}^T \begin{pmatrix} Q_{xx} & Q_{ux} \\ Q_{ux}^T & Q_{uu} \end{pmatrix} \begin{pmatrix} \delta x \\ \delta u \end{pmatrix} && (2) \end{aligned}$$

where subscripts denote derivatives. The Q function is maximized with respect to δu

$$\begin{aligned} 0 &= \nabla_{\delta u} Q(x, u) = Q_u + Q_{ux}\delta x + Q_{uu}\delta u \\ \delta u &= -Q_{uu}^{-1}(Q_u + Q_{ux}\delta x) \end{aligned}$$

Plugging the optimal δu back into $Q(x, u)$ yields the value function

$$V(x_k) = \left(Q_0 - \frac{1}{2}Q_u^T Q_{uu}^{-1} Q_u\right) + \left(Q_x^T - Q_u^T Q_{uu}^{-1} Q_{ux}\right)\delta x + \frac{1}{2}\delta x^T \left(Q_{xx} - Q_{ux}^T Q_{uu}^{-1} Q_{ux}\right)\delta x$$

The coefficients of (2) are found by quadratically expanding (1):

$$\begin{aligned} C(x_k, u_k) &\approx C(x_k, u_k) + C_x^T \delta x + C_u^T \delta u + \frac{1}{2}\delta x_k^T C_{xx} \delta x_k + \frac{1}{2}\delta u_k^T C_{uu} \delta u_k + \delta u_k^T C_{ux} \delta x_k \\ V^{k+1}(f(x_k, u_k)) &\approx V^{k+1} + (V_x^{k+1})^T (A_k \delta x_k + B_k \delta u_k) + \frac{1}{2}\left((A_k\,\delta x)^T\ (B_k\,\delta u)^T\right) V_{xx}^{k+1} \begin{pmatrix} A_k\,\delta x \\ B_k\,\delta u \end{pmatrix} \\ &\quad + \frac{1}{2}\left(\delta x^T\ \delta u^T\right) H \begin{pmatrix} \delta x \\ \delta u \end{pmatrix} \end{aligned}$$

where elements of H are given by

$$H_{ij} = \sum_l \frac{\partial V^{k+1}}{\partial x_l} \frac{\partial^2 f(x_k)_l}{\partial x_i \partial x_j}.$$

DDP was run on the gravity pendulum with an array of starting points. From the resulting array of optimal paths it seems that DDP is more or less finding the same controller as in value iteration (Figure 3). There are some noticeable discrepancies which I currently believe are results of buggy code.

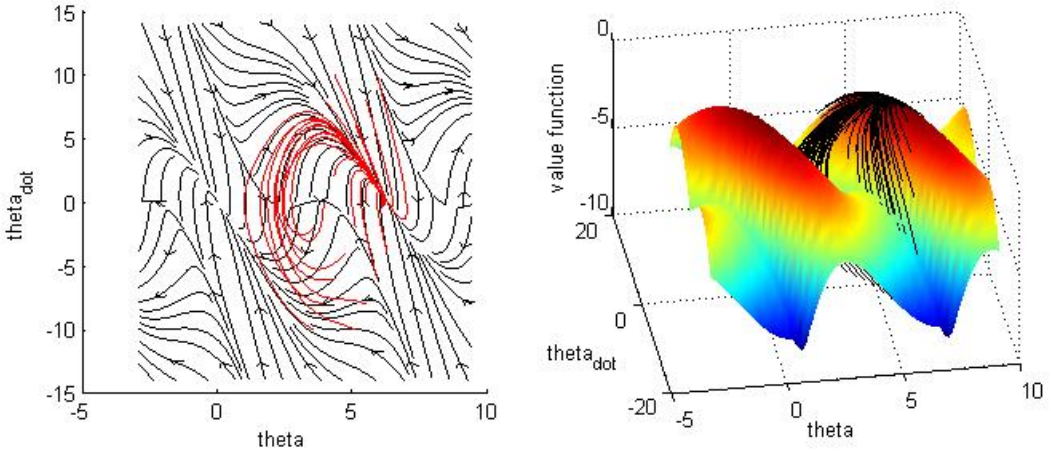

Figure 3: DDP results overlaid on value iteration results. Left: DDP (red) traces out paths similar to those found with value iteration (black). Right: value functions found with DDP (black) hug the value function found with value iteration (colored).

Finally, DDP was run on a standard cart-pole system. Using a receding horizon a nice swing-up controller was obtained (Figure 4).

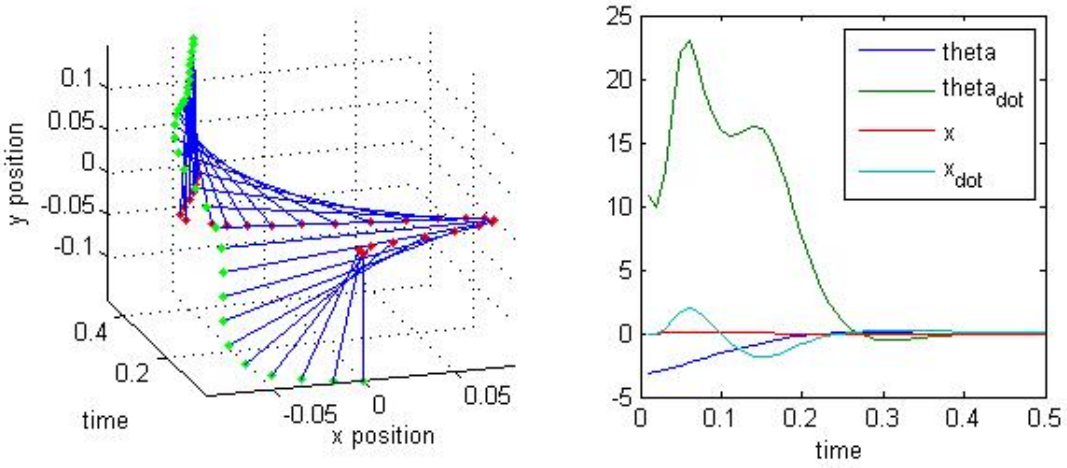

Figure 4: Cart-pole swing up controller found with DDP. (Left: "strobelight-style" transient plot.)

References

[1] Kenji Doya. Reinforcement learning in continuous time and space. *Neural Computation*, 12(1):219–245, 2000.