Fast Reinforcement Learning in Continuous Action Spaces

Daniel N. Nikovski*
Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213
E-mail: nikovski@cs.cmu.edu

Category: reinforcement learning and control. Not published previously. Preference: poster.

Abstract

A new fast algorithm for reinforcement learning (RL) in continuous state and action spaces is proposed. Unlike algorithms based on dynamic programming, the proposed algorithm uses neither temporal nor spatial differencing. Instead, it couples the solution of the Hamilton-Jacobi-Bellman (HJB) partial differential equation with the structure of the function approximators that are commonly used together with RL algorithms to provide generalization over state space. The proposed technique bears close resemblance to the finite element method (FEM) widely used in physics and engineering. Pontryagin's optimal principle is applied without introducing any penalty functions, and locally-weighted function approximation is employed for generalization. As a result, the HJB equation is reduced to a set of linear algebraic equations which can be solved iteratively to yield the optimal value function of the problem in very few iterations.

1 INTRODUCTION

Much of the early work and the first major applications of reinforcement learning (RL) algorithms have been in environments characterized with discrete state and action spaces — board games, simulated mazes, etc. Notwithstanding their success, these algorithms can rarely be transferred directly to physical systems in the real

*Current address until August 31, 1997: Siemens Corporate Research, Princeton, NJ 08540. E-mail: nikovski@scr.siemens.com

world, which are usually described in continuous state and action spaces. The requirement for a finite set of actions and states of the world is essential to the work of most popular RL algorithms such as Q-learning (Watkins, 1989) and TD(λ) (Sutton, 1988), and stems from their origins in dynamic programming (Bellman, 1957).

Considerable research effort has been devoted recently to the development of new algorithms that can learn optimal policies for the control of dynamic systems described in continuous state and action spaces. While there have been successful attempts to adapt discrete-space algortihms to such systems by discretizing the continuous variables (Boyan and Moore, 1995; Atkeson, 1994), this approach does not scale up favorably because of the well-known "curse of dimensionality", accompanying algorithms based on dynamic programming (Bellman, 1957).

Another existing approach is that of Baird and Klopf (1993), based on a special parametric representation ("control wires") of the state-action space. While successful, this solution would be difficult to apply to the cases when the system evolves along a low-dimensional manifold in state space, since the "control wires" cannot follow the manifold.

Doya (1996, 1997) extended Sutton's TD(λ) algorithm to continuous time and space and used radial basis functions on a regular grid for the representation of the optimal value function. He introduced penalty functions into the HJB equation of the underlying dynamic system and considered an extremal case for these penalty functions, which resulted in "bang-bang" control. As a consequence, the controls of the system that he simulated degenerated back to a discrete set of two actions.

The algorithm presented in this paper is closest to Doya's approach, with several differences. The most significant departure is that neither temporal nor spatial differencing is considered at all. Instead, we propose to solve the underlying HJB partial differential equation directly by means of techniques similar to the finite-element method. A second difference is that locally-weighted learning is used, which permits learning in high-dimensional state spaces. A third technique is to constrain the control to a constant-radius hypersphere in control space and apply Pontryagin's optimality principle on this restricted control space only. As a result of these three techniques, the HJB equation can be reduced to a linear system of algebraic equations, the iterative solution of which gives the optimal value function of the problem.

2 OPTIMAL CONTROL IN CONTINUOUS SPACES

The dynamics of a general system evolving in continuous state space and having continuous controls can be described with a set of first-order ordinary differential equations:

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}^{(t)}, \mathbf{u}^{(t)}).$$

where $\mathbf{x} \in \mathbf{X} \subset \mathbf{R}^n$ is the n-dimensional state space and $\mathbf{u} \in \mathbf{U} \subset \mathbf{R}^m$ is the m-dimensional control vector.

We assume that the reinforcement is a scalar function of the state only, $r = r(\mathbf{x})$, and postulate the existence of a control function (policy) $\mathbf{u} = \pi(\mathbf{x})$ that is time independent; this is possible when infinite horizon problems are considered (Lewis, 1986). Based on the derivation in Doya (1996), the value $V^\pi(\mathbf{x})$ of a state $\mathbf{x}$ under a policy $\pi(\mathbf{x})$ is a scalar expressing the expected discounted reward (reinforcement) if that policy is followed:

$$V^\pi(\mathbf{x}) = \int_0^\infty \frac{1}{\tau} e^{-\frac{t}{\tau}} r(\mathbf{x}^{(t)}) dt, \qquad \mathbf{x}^{(0)} = \mathbf{x},$$

where $\mathbf{x}^{(t)}$ describes the trajectory followed by the system governed by the equation

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}^{(t)}, \pi(\mathbf{x}^{(t)})).$$

Here τ is a time constant that controls the discounting of future rewards. The optimal policy $\pi^*(\mathbf{x})$ is the one that maximizes the expected reinforcement for all states $\mathbf{x}$:

$$V^*(\mathbf{x}) = \max_{\mathbf{u}[0,\infty]} \left[\int_0^\infty \frac{1}{\tau} e^{-\frac{t}{\tau}} r(\mathbf{x}^{(t)}) dt \right].$$

While there might be many optimal policies, there is always a unique optimal value function $V^*(\mathbf{x})$, and the goal of our algorithm is to find it. Following the general approach in optimal control (Lewis, 1986), we can divide the trajectory in two periods and optimize separately over them:

$$V^*(\mathbf{x}) = \max_{\mathbf{u}[0,\infty]} \left[\int_0^{\Delta t} \frac{1}{\tau} e^{-\frac{t}{\tau}} r(\mathbf{x}^{(t)}) dt + e^{-\frac{-\Delta t}{\tau}} \int_0^\infty \frac{1}{\tau} e^{-\frac{s}{\tau}} r(\mathbf{x}^{(s)}) ds \right].$$

After time Δt the system will be in a new state

$$\mathbf{x}' = \mathbf{x} + \int_0^{\Delta t} \mathbf{f}(\mathbf{x}^{(t)}, \mathbf{u}^{(t)}) dt,$$

which to a first-order approximation is

$$\mathbf{x}' = \mathbf{x} + \mathbf{f}(\mathbf{x}, \mathbf{u}^{(0)})\Delta t + O(\Delta t^2) \qquad \Delta\mathbf{x} = \mathbf{x}' - \mathbf{x} = \mathbf{f}(\mathbf{x}, \mathbf{u}^{(0)})\Delta t + O(\Delta t^2).$$

If optimal policy is followed after time Δt, the second term in the sum can be recognized to be the discounted optimal value of the new state $\mathbf{x}'$:

$$V^*(\mathbf{x}) = \max_{\mathbf{u}[0,\Delta t]} \left[\int_0^{\Delta t} \frac{1}{\tau} e^{-\frac{t}{\tau}} r(\mathbf{x}^{(t)}) dt + e^{-\frac{\Delta t}{\tau}} V^*(\mathbf{x}') \right].$$

We can expand the value function at the new state to first order and use in the process the expansion for the system dynamics from above:

$$V^*(\mathbf{x}') = V^*(\mathbf{x}) + \nabla_{\mathbf{x}} V^*\big|_{\mathbf{x}} \cdot \Delta\mathbf{x} + O(\Delta\mathbf{x}^2) \qquad V^*(\mathbf{x}') = V^*(\mathbf{x}) + \nabla_{\mathbf{x}} V^*\big|_{\mathbf{x}} \cdot \mathbf{f}(\mathbf{x}, \mathbf{u}^{(0)})\Delta t + O(\Delta t^2).$$

Here $\nabla_{\mathbf{x}} V^*|_{\mathbf{x}}$ is the gradient of the value function evaluated at point $\mathbf{x}$. In the limit of Δt to zero, we obtain the discounted version of the Hamilton-Jacobi-Bellman equation:

$$V^*(\mathbf{x}) = r(\mathbf{x}) + \tau \max_{\mathbf{u}(\mathbf{x})} \left[\nabla_{\mathbf{x}} V^*\big|_{\mathbf{x}} \cdot \mathbf{f}(\mathbf{x}, \mathbf{u}(\mathbf{x})) \right]. \qquad (1)$$

3 SOLVING FOR THE OPTIMAL VALUE FUNCTION

The derived HJB equation is a partial differential equation (PDE) in the optimal value function $V^*(\mathbf{x})$. Two popular approaches to solving PDEs are the finite difference method and the finite element method (Burden and Faires, 1993). While virtually all researchers in the field of reinforcement learning have used finite differencing, we will pursue the other alternative. Two features of the finite element method make it much preferable to finite differencing in the context of RL:

- Unlike the finite difference method, FEM does not require a regularly spaced grid or some other regular decomposition of state space — instead, space is decomposed by means of irregular volume elements. If the system dynamics evolves on a low-dimensional manifold of state space, the elements need only cover the manifold and not the whole space, thus avoiding the "curse of dimensionality".
- It is usual practice in RL to represent the value functions by means of a neural net structure such as a multilayer perceptron, radial basis function network, or some other universal approximator. Once a decision has been made to use such an approximator, the derivative terms in the PDE can be easily expressed by differentiating the universal approximator (which is differentiable by design), instead of expressing them as finite differences in state space.

To clarify these arguments, let us consider the solution of the HJB equation by means of kernel regression. The evolution of an unknown dynamical system is sampled at regular time intervals and all transitions are recorded in a database whose records are tuples of the following format:

$$[\mathbf{x}^{(t)}, \mathbf{u}^{(t)}, \mathbf{x}^{(t+1)}, r^{(t)}, V(\mathbf{x}^{(t)})], \qquad t = 0, T-1.$$

The first elements of the tuples, $\mathbf{x}^{(t)}$, form the nodes necessary for the work of the FEM approximation. The last elements, $V(\mathbf{x}^{(t)})$ are the unknown estimates of the optimal value function $V^*(\mathbf{x}^{(t)})$ that are to be determined by the algorithm. The basis functions $G(\mathbf{x})$ are Gaussians centered at the nodes $\mathbf{x}^{(t)}$:

$$G_t(\mathbf{x}) = e^{-\frac{(\mathbf{X}-\mathbf{X}^{(t)})^2}{h}} \qquad \omega^{(t)} = G_t(\mathbf{x}),$$

where h is the width of the kernel and $\omega^{(t)}$ are weights. The approximated value function $V(\mathbf{x})$ at an arbitrary point $\mathbf{x}$ in state space is calculated by means of kernel regression (Moore et al., 1995):

$$V(\mathbf{x}) = \frac{\sum_{t=0}^{T-1} \omega^{(t)} V(\mathbf{x}^{(t)})}{\sum_{t=0}^{T-1} \omega^{(t)}}. \tag{2}$$

We can differentiate directly equation (2) to obtain the gradient of the value function $\nabla_{\mathbf{X}} V$, instead of approximating it by the difference of two values of $V(\mathbf{x})$. A single component of the gradient will be:

$$\frac{\partial V}{\partial x_i} = \frac{(\sum_{t=0}^{T-1} \omega_i^{(t)'} V^{(t)}) \sum_{t=0}^{T-1} \omega^{(t)} - (\sum_{t=0}^{T-1} \omega^{(t)} V^{(t)}) \sum_{t=0}^{T-1} \omega_i^{(t)'}}{(\sum_{t=0}^{T-1} \omega^{(t)})^2}$$

where all partial derivatives $\omega_i^{(t)'}$ of the weights $\omega^{(t)}$ can be easily computed once the weights themselves are known:

$$\omega_i^{(t)'} = \frac{\partial \omega^{(t)}}{\partial x_i} = -\frac{2}{h}\omega^{(t)}(x_i - x_i^{(t)}).$$

Note that the gradient is linear in the values $V^{(t)}$. In order to solve the HJB equation, we linearize the dynamics of the system around the current state $\mathbf{x}$:

$$\mathbf{f}(\mathbf{x}, \mathbf{u}) = \mathbf{f}(\mathbf{x}, 0) + \frac{\partial \mathbf{f}}{\partial \mathbf{u}}\Big|_{x,0}\mathbf{u} + O(\mathbf{u}^2),$$

where $\frac{\partial \mathbf{f}}{\partial \mathbf{u}}|_{x,0}$ is the Jacobian matrix of control gains, evaluated for $(\mathbf{x}, 0)$. This matrix can be obtained again by means of kernel regression, building a forward dynamic model as described in Moore et al. (1995), and then differentiating the model.

Plugging in the linearized dynamics into the HJB equation (1), we obtain:

$$V^*(\mathbf{x}) = r(\mathbf{x}) + \tau \nabla_{\mathbf{x}} V^*\big|_{\mathbf{x}} \cdot \mathbf{f}(\mathbf{x}, 0) + \tau \max_{\mathbf{u}(\mathbf{x})}\left[\nabla_{\mathbf{x}} V^*\big|_{\mathbf{x}} \frac{\partial \mathbf{f}}{\partial \mathbf{u}}\Big|_{x,0}\mathbf{u}\right].$$

According to Pontryagin's maximum principle (Lewis, 1986), the optimal control is the one that maximizes the right-hand side of the above equation. Since neither the gradient $\nabla_{\mathbf{x}} V^*$, nor the control gains $\frac{\partial \mathbf{f}}{\partial \mathbf{u}}$ depend on $\mathbf{u}$, no finite value of $\mathbf{u}$ can maximize the right hand side of the equation above. In order to avoid infinite controls, usually some form of penalty function is employed (Lewis, 1986; Doya, 1996). We will use a different approach. Instead of optimizing $\mathbf{u}$ over the whole of $\mathbf{R}^m$, we can constrain it to lie on a hypesphere with prespecified radius, i.e., $||\mathbf{u}|| = u_0$. This constraint is realistic, because controls in real physical systems are usually constrained within an interval. With it, the optimal control can be easily found to be:

$$\mathbf{u}_{opt} = u_0 \frac{\nabla_{\mathbf{x}} V^* \frac{\partial \mathbf{f}}{\partial \mathbf{u}}}{||\nabla_{\mathbf{x}} V^* \frac{\partial \mathbf{f}}{\partial \mathbf{u}}||}$$

Once the optimal control $\mathbf{u}_{opt}$ is determined, the differential constraint defined by means of the HJB equation can be enforced in the direction of application of $\mathbf{u}_{opt}$:

$$V^*(\mathbf{x}) = r(\mathbf{x}) + \tau \nabla_{\mathbf{x}} V^*\big|_{\mathbf{x}} \cdot \left[\mathbf{f}(\mathbf{x}, 0) + \frac{\partial \mathbf{f}}{\partial \mathbf{u}}\Big|_{x,0}\mathbf{u}_{opt}\right]. \quad (3)$$

This dependency must hold for each node $\mathbf{x}^{(t)}$, $t = 0, T-1$. In order to find an approximation $V(\mathbf{x})$ of the true value function $V^*(\mathbf{x})$, we can start with arbitrary values for $V(\mathbf{x})$ and iteratively improve them to satisfy the differential constraint (3). The expression for the gradient of the value function $\nabla_{\mathbf{x}} V|_{\mathbf{x}}$ is of the form $\nabla_{\mathbf{x}} V|_{\mathbf{x}} = \mathbf{A}_1\mathbf{v} + \mathbf{b}_1$, where $\mathbf{A}_1$ is an n-by-T matrix, $\mathbf{b}_1$ is a vector, and $\mathbf{v} = [V^{(0)}, V^{(1)}, \ldots, V^{(T-1)}]$ is the vector of estimates of the optimal value function for all nodes. The term corresponding to the linearized dynamics does not depend on V and can be represented as another vector

$$\mathbf{b}_2 = \mathbf{f}(\mathbf{x}, 0) + \frac{\partial \mathbf{f}}{\partial \mathbf{u}}\Big|_{x,0}\mathbf{u}_{opt}.$$

It can be seen that the application of the differential constraint at every node results in a linear system of algebraic equations of the form $\mathbf{A}\mathbf{v} = \mathbf{b}$.

What is more, the form of these equations is very close to the form required for the application of iterative relaxation methods (Burden and Faires, 1993). The variable $V(\mathbf{x}^{(t)})$ is already on the left-hand side of the equation, and there are only two more terms involving $V(\mathbf{x}^{(t)})$ on the other side. Once all terms involving $V(\mathbf{x}^{(t)})$ are on the left-hand side of the equation, it is possible to perform Jacobi, Gauss-Seidel, successive over/under relaxation, or Richardson iterations (Burden and Faires, 1993).

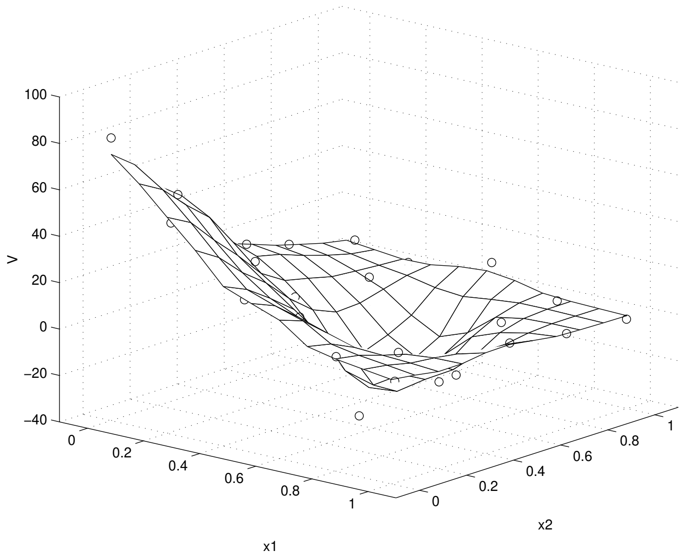

Figure 1: Learned value function over 25 irregularly spaced nodes, after nine iterations. The circles show the values of $V(\mathbf{x})$ at the nodes, and the surface shows a smoother approximation by means of kernel regression.

4 SIMULATION

The algorithm was tested on a simple two-dimensional system with two controls. The control gains were assumed to be known and constant throughout state space. The goal of the experiment was to verify if a good approximation of the value function can be learned without a regular cell grid and if the proper continuous controls can be determined from that value function. A total of 25 nodes were used in the solution of the HJB equation on a square of side one in state space (Figure 1). These nodes covered the whole square, but were not regularly spaced. The system got positive reinforcement $r_0 = 50$ at the node with coordinates $(0, 0)$ ("the goal"), and negative reinforcement $r_G = -80$ peaked in the middle of the square and decreasing exponentially around it (a simulated "obstacle"). As parameters, $h = 0.05$ and $\tau = 9$ were used. For better stability, successive underrelaxation was performed with $\eta = 0.5$. i.e., at every iteration, the new values for $\mathbf{v}$ were the average of the old ones and what would be returned by a pure Gauss-Seidel step.

5 DISCUSSION AND CONCLUSION

Figure 1 shows a reasonable approximation of the value function. If a system follows the gradient of the surface, it will reach the "goal" state while at the same time avoiding the "obstacle" region of state space. The solution converged in only nine iterations, and no boundary conditions for the HJB PDE had to be specified.

While successful in solving this learning problem, the algorithm has certain drawbacks with respect to a true finite element method. Because the chosen basis functions have infinite support, evaluating the gradient of the value function at a single node takes $O(nT)$ computations, instead of $O(n)$. This makes the cost of a single iteration over all T nodes $O(nT^2)$. In high-dimensional state spaces, the computational burden might be significant, both at time of training and when using the value function for control. One solution might be to compile the optimal controlls into an "actor" network, similar to Doya's "actor-tutor" architecture (Doya, 1997).

References

Atkeson, C.G. (1994). Using local trajectory optimizers to speed up global optimization in dynamic programming, In: Neural Information Processing Systems 6, Morgan Kaufmann.

Baird, L.C. and Klopf, A.H. (1993). Reinforcement learning with high-dimensional, continuous actions. Technical Report WL-TR-93-1147, Wright-Patterson Air Force Base Ohio: Wright Laboratory.

Bellman, R. (1957). Dynamic Programming. Princeton University Press.

Boyan, J.A. and Moore, A.W. (1995). Generalization in reinforcement learning: Safely approximating the value function. In:Neural Information Processing Systems 7, MIT Press.

Burden, R.L. and Faires, J.D. (1993). Numerical Analysis, PWS-KENT.

Doya, K. (1996). Temporal difference learning in continuous time and space. In: Neural Information Processing Systems 8, MIT Press.

Doya, K. (1997). Efficient nonlinear control with actor-tutor architecture. In: Neural Information Processing Systems 9, MIT Press.

Lewis, F.L. (1986). *Optimal Control.* Wiley.

Moore, A.W., Atkeson, C.G, and Schaal, S. (1995). Memory-based learning for control. Technical Report CMU-RI-TR-95-18, CMU Robotics Institute, 1995.

Sutton, R.S. (1988). Learning to predict by the method of temporal differences. *Machine Learning,* 3:9-44.

Watkins, C.J.C.H. (1989). Learning from delayed rewards. Doctoral thesis, Cambridge University, Cambridge, England.