Reinforcement Learning in Continuous Spaces

Daniel Nikovski

Problem: Reinforcement learning algorithms such as Q-learning and TD(λ) can operate only in discrete state and action spaces, because they are based on Bellman back-ups and the discrete-space version of Bellman's equation. However, most robotic applications of reinforcement learning require continuous state spaces defined by means of continuous variables such as position, velocity, torque, etc. The usual approach has been to discretize the continuous variables, which quickly leads to combinatorial explosion and the well known "curse of dimensionality". The goal of our research is to provide a fast algorithm for learning in continuous spaces that does not use a discretization grid of the whole space, but instead represents the value function only on the manifold actually occupied by the visited states.

Impact: Handling continuous spaces has been identified as one of the most important research directions in the field of reinforcement learning. The greatest impact of our approach is expected to be in robotic applications of reinforcement learning to high-dimensional perceptual spaces, such as visual servoing or any other vision-based task. Visual spaces have huge dimensionality (a dimension per pixel), while all possible percepts occupy a low-dimensional manifold of that space. For example, the percepts recorded by a mobile robot with fixed camera lie on a three-dimensional manifold, which grid-based approaches cannot represent efficiently.

State of the Art: A clear distinction exists in reinforcement learning between the state space and the action space of a task. In general, it is much easier to deal with a continuous state space than with a continuous action space. The value function of reinforcement learning problems has been commonly represented by means of a universal function approximator such as a neural net. The inputs to an approximator are the state variable, which can just as well be continuous, at least in theory. In practice, learning the value function with a function approximator often fails, because of the two separate approximations going on simultaneously — one of the value function by means of Bellman back-ups, and another one by means of some general supervised learning rule. Moore and Boyan proposed an algorithm that safely approximates the value function, avoiding convergence; however, this algorithm does not escape the "curse of dimensionality".

Dealing with continuous action further complicates matters, because Bellman back-ups are not applicable. Instead, the Hamilton-Jacobi-Bellman (HJB) PDE has to be used. Doya 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. Atkeson's algorithm finds policies in continuous spaces by means of refining trajectories in state space, but does not scale up to higher dimensional problems either. Baird and Klopf used a special representation of state space, called *control wires*, which has the same limitation.

Approach: Our algorithm is close to Doya's approach, with several differences. The most significant departure is that neither temporal nor spatial differencing is considered at all. The underlying HJB partial differential equation is solved directly by means of techniques derived from the finite-element method (FEM). 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

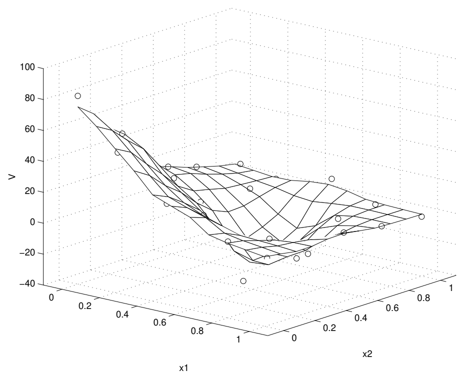

Figure 1: Learned value function over 25 irregularly spaced nodes, after nine iterations. The circles show the value function at the nodes, and the surface shows a smoother approximation by means of kernel regression.

---

manifold and not the whole space, thus avoiding the "curse of dimensionality". In order to avoid controls of infinite magnitude, they are constrained to lie on a constant-radius hyper-sphere in action space, and Pontryagin's optimality principle is applied on this restricted space only. Furthermore, if locally-weighted learning with Gaussian kernels is used, 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.

So far, the algorithm has been tested on simulated tasks to test its convergence. As expected, it performs well on regular grids and quickly approximates the correct value function. In another experiment, the performance of the algorithm on an irregular grid was tested. A total of 25 nodes were used in the solution of the HJB equation on a square of side one in state space (Fig. 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"). For better stability, successive under-relaxation was performed with learning rate $\eta = 0.5$. i.e., at every iteration, the new values for the value function were the average of the old ones and what would be returned by a plain Gauss-Seidel step. Fig. 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.

Future Work: 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, a single iteration over all T nodes (examples) is $O(T^2)$. In high-dimensional state spaces, the computational burden might be significant. One solution might be to compile the optimal controls into an "actor" network, similar to Doya's "actor-tutor" architecture. Another problem to be solved is the proper triangulation of the state manifold. Like any other application of the finite-element method, if triangulation is not dense enough, the iterations converge to a distorted solution. Once stable convergence is achieved in high-dimensional spaces, the algorithm will be tested on a real robotic application such as visual servoing or manipulation.

Selected References:

[1] Daniel Nikovski. Fast reinforcement learning in continuous action spaces. *To be submitted*.