Lunar Lander: A Continuous-Action Case Study for Policy-Gradient Actor-Critic Algorithms F57

Roshan Shariff, Travis Dick {roshan.shariff,tdick}@ualberta.ca

UNIVERSITY OF ALBERTA
DEPARTMENT OF COMPUTING SCIENCE

Introduction

Reinforcement learning has successfully been applied to a wide variety of control problems, usually with a finite number of actions. Policy-gradient algorithms use the ideas of reinforcement learning in domains that may have an infinite or continuous set of actions. We investigate the use of such an algorithm with a simple domain with continuous states and actions, based on a two-dimensional simulation of the Apollo lunar lander.

The Lunar Lander Domain

We introduce a new domain in which the agent must learn to control the Apollo lunar lander and guide it to a safe landing on a target on the lunar surface. The state of the lander is specified by six variables—its position and orientation (x, y, and θ) and its translational and rotational velocities (v_x, v_y, and ω). The agent has two controls—the main engine (which points down) and the attitude control system (which changes the lander's rate of rotation). These two controls constitute the two-dimensional continuous action space.

The simulator uses lunar gravity ($g = 1.622\,\mathrm{m/s^2}$). It updates the state variables at a rate of 20 Hz ($\Delta t = 1/20\,\mathrm{s}$) according to the following equations:

$$\begin{aligned}
\tilde{v}_{x,t} &:= v_{x,t-1} + \Delta t \cdot (-a_{\text{thrust}} \sin\theta) \\
\tilde{v}_{y,t} &:= v_{y,t-1} + \Delta t \cdot (a_{\text{thrust}} \cos\theta - g) \\
\tilde{\omega}_t &:= \omega_{t-1} + \Delta t \cdot (a_{\text{acs}}) \\
(v_{x,t}, v_{y,t}, \omega_t) &:= \texttt{do\_collision}(x_t, y_t, \theta_t, \tilde{v}_{x,t}, \tilde{v}_{y,t}, \tilde{\omega}_t) \\
x_t &:= x_{t-1} + \Delta t \cdot v_{x,t} \\
y_t &:= y_{t-1} + \Delta t \cdot v_{y,t} \\
\theta_t &:= \theta_{t-1} + \Delta t \cdot \omega_t
\end{aligned}$$

The agent updates its actions (a_{thrust} and a_{acs}) at a rate of 2 Hz. The `do_collision` function modifies the velocity of the lander when it collides with the lunar surface, using the algorithm of Guendelman et al. (2003). It determines when the lander has settled and calculates the severity of the collision.

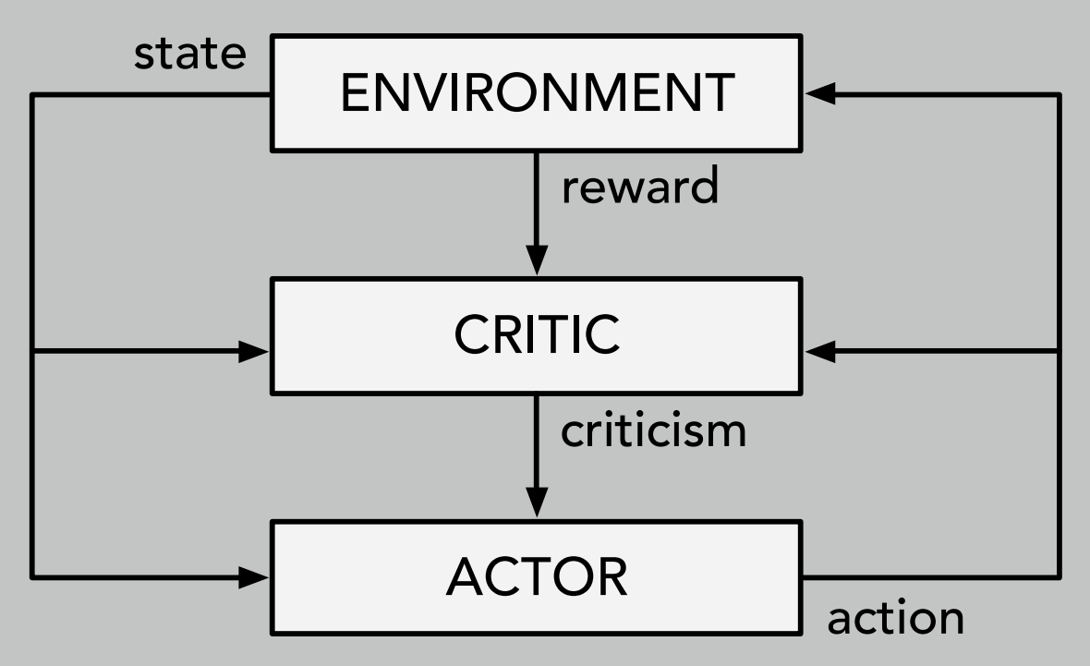
The Learning Task and Reward

The goal of the agent is to land the lunar module softly at a designated target. At each time step of the episodic task, the agent receives a small penalty proportional to its use of the main engine, and another penalty for the accumulated impact forces on the lander. An episode ends when the agent lands safely (+1 reward), exceeds a time limit (-1 penalty), suffers a catastrophic impact, or flies too far from the target. In all these situations, it also receives a penalty proportional to the lander's final distance from the target.

Actor-Critic

Actor-critic algorithms have two components: the actor and the critic. The actor chooses actions and the critic evaluates those actions. The actor uses the critic's feedback to improve its behaviour. By using a parametric policy, the actor can efficiently choose from a large (possibly infinite or even continuous) set of actions.

The Actor-Critic Architecture



Policy-Gradient

A parametric policy π_θ (with parameters θ) selects action a in state s with probability $\pi_\theta(s, a)$. The performance of the policy can be measured by its average per-episode return:

$$\rho(\theta) = \mathbb{E}\left[\sum_{t=1}^{T} R_t\right],$$

where the sequence of rewards R_t is generated by following π_θ.

The policy gradient theorem (Sutton et al., 2000) gives an unbiased estimate of the gradient of performance with respect to the policy parameters:

$$\nabla_\theta\,\rho(\theta) = \mathbb{E}\left[\frac{\nabla_\theta\,\pi_\theta(S,A)}{\pi_\theta(S,A)} Q_\theta(S,A)\right].$$

Q_θ is the policy's action-value function and the states S and actions A are sampled from the policy's stationary distribution. Since the agent's experience is approximately sampled from the stationary distribution, it can be used to estimate the gradient of $\rho(\theta)$ without an explicit model of the environment. The estimator of Q_θ can be viewed as the critic in the actor-critic architecture. The actor maximizes $\rho(\theta)$ by using these gradient estimates to perform a stochastic gradient ascent in the space of policy parameters.

We use the AC-S policy-gradient algorithm of Degris et al. (2012), which estimates the state value function with TD(λ) and gives the TD error as feedback to the actor. It also uses eligibility traces so that credit for a reward is given to the history of actions that led to it.

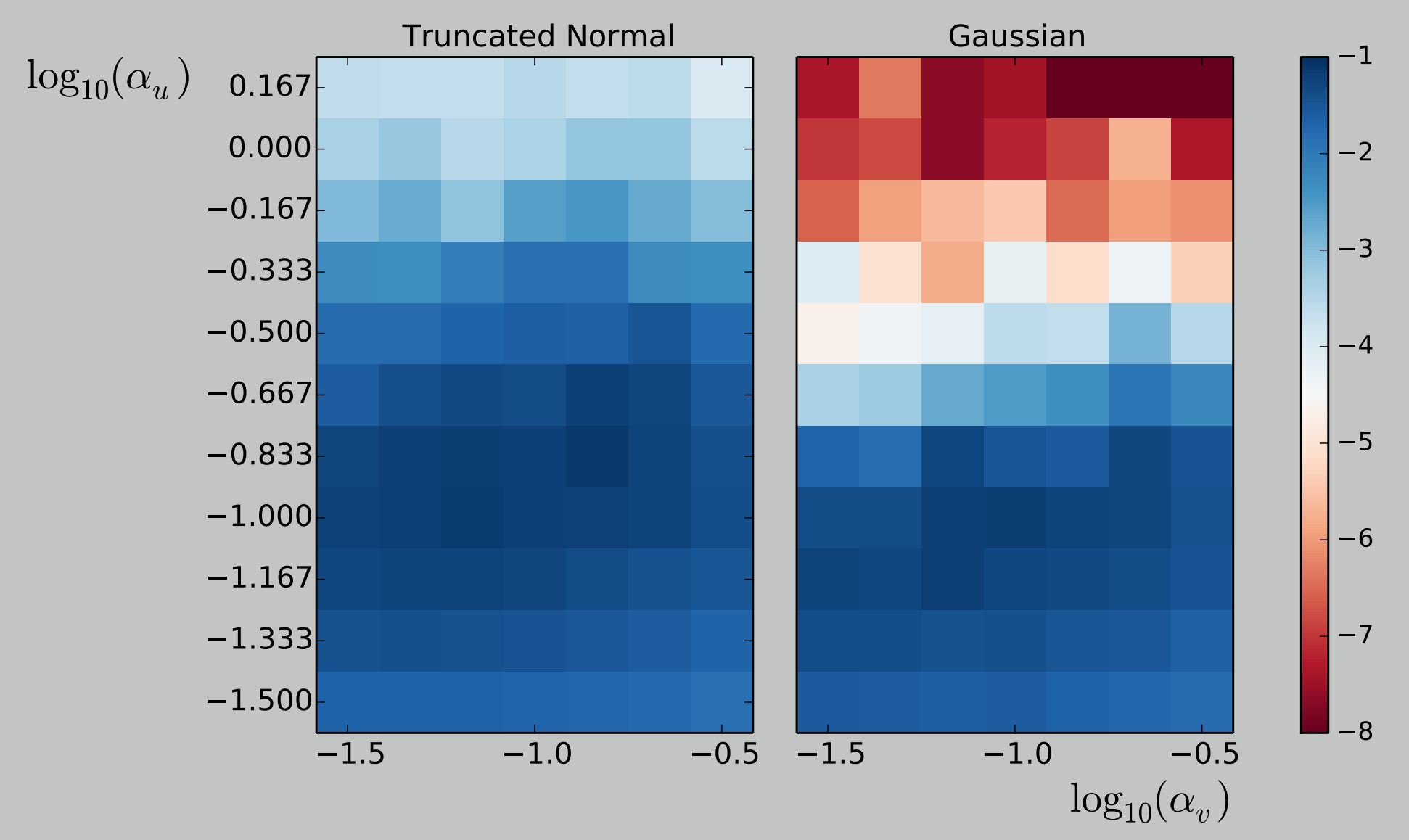
Empirical Results

We propose a version of the AC-S algorithm with eligibility traces, feature weighting, and truncated action proposal distributions (each is explained in one of the following sections). In the following experiments, we compare this primary algorithm with versions missing one of these features. For each variation, we performed a coarse parameter study to find the best actor and critic step sizes, α_u and α_v. The optimized parameters for the primary algorithm were $\alpha_u = \alpha_v = 0.147$, and $\lambda = 0.8$.

Truncated Action Proposal Distribution

In domains like the lunar lander, the range of allowable actions is bounded. The AC-S algorithm of Degris et al. (2012) samples actions from a normal distribution whose parameters μ and σ are linear functions of the state feature vector (passed through a transfer function). The normal distribution, however, can always propose actions outside the allowed range. Instead, we use a truncated normal distribution to propose actions, which makes the algorithm aware of the action range and never propose invalid actions.

Empirically, we found that on this domain this modification did not significantly affect the learning curves for the optimal step sizes. However, a parameter study over α_u and α_v showed that the truncated normal distribution makes the algorithm less sensitive to the choice of step sizes.

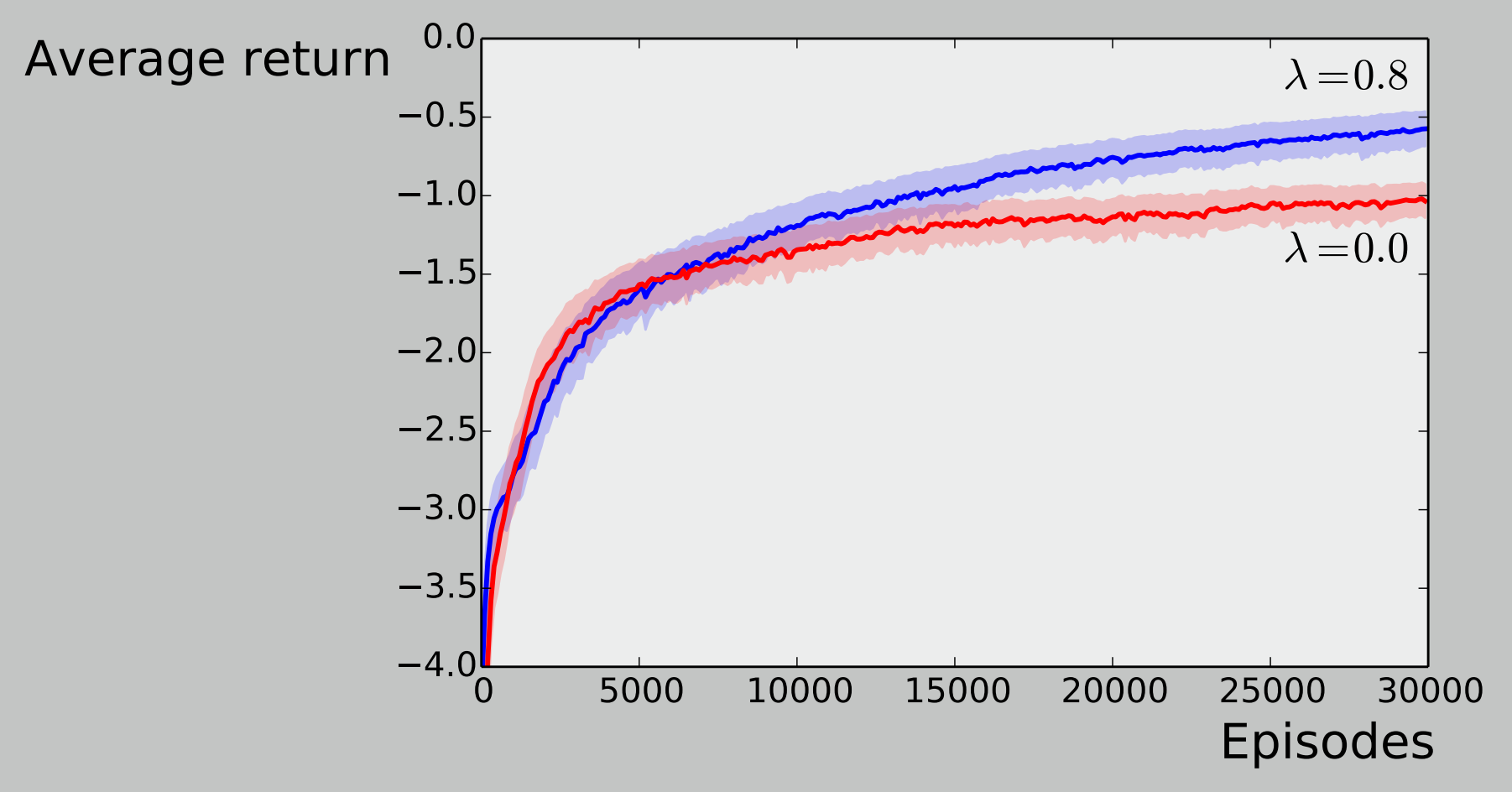


Eligibility Traces

The TD(0) algorithm credits each action with the reward that immediately follows it. In many situations, however, TD(λ) (with $\lambda > 0$) performs better because it uses an *eligibility trace* to assign credit for each reward to a (limited) history of earlier actions. The importance of each action decays geometrically by a factor λ per time step. An important contribution of the AC-S algorithm (Degris et al., 2012) is to introduce eligibility traces to policy gradient algorithms.

Unlike feature vectors produced by tile coders, the eligibility trace is not sparse. To maintain sparsity (allowing efficient implementation) we store the eligibility trace as a queue of sparse feature vectors. As each vector in the queue is implicitly multiplied by λ at each time step, its importance drops below any small constant ε after $\log_\lambda \varepsilon$ steps. By truncating the queue to this length, our algorithm approximates the eligibility trace with a modest amount of computation. For example, with $\lambda = 0.8$ and $\varepsilon = 0.05$, the queue contains a maximum of $\lceil 13.25 \rceil = 14$ of the newest feature vectors.

We measured the effect of eligibility traces by comparing the performance of the AC-S algorithm with $\lambda = 0.8$ to the version with no traces ($\lambda = 0$).

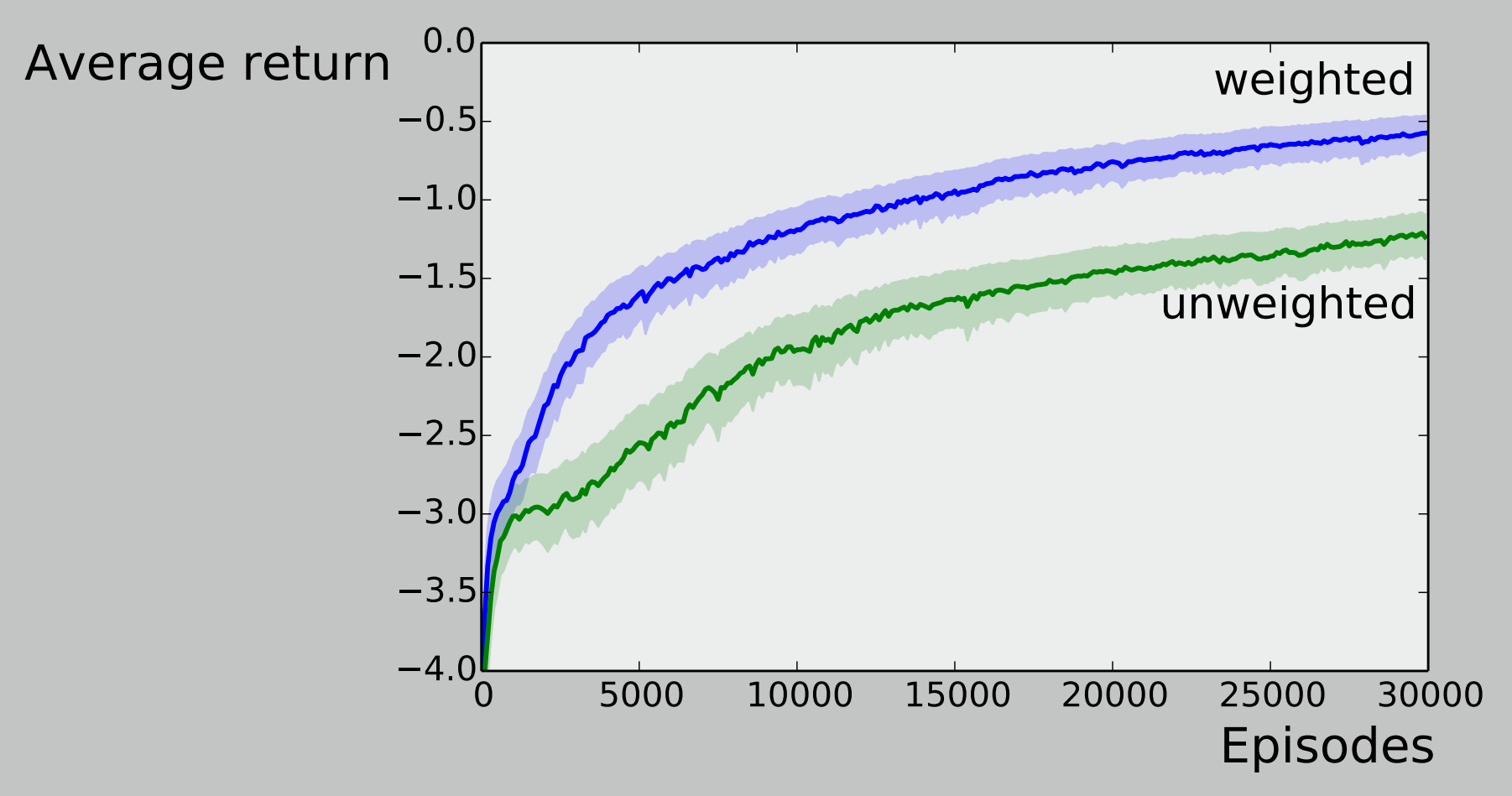


The experiments demonstrate the improved learning possible with eligibility traces, even with our approximate implementation. Without eligibility traces the algorithm is forced to use larger step sizes ($\alpha_u = 0.215$, $\alpha_v = 0.5$) for the actor and critic to remain competitive, which may increasing its susceptibility to noise.

Feature Weighting

Tile coding is often used to represent continuous states as feature vectors, where each feature representing a cell in a tiling of the state space. The tilings are often offset to increase resolution without sacrificing generalization. Tiling subsets of the state variables in addition to the entire space also helps generalization. Because the number of offsets required to adequately cover the space increases exponentially with the number of variables in the tiling, subsets of many state variables contribute more tilings to the feature vector than smaller subsets. Since the lower dimensional tilings are under represented in the feature vector, their benefits are negated.

We propose a feature weighting scheme where the features representing each subset of state variables are weighted inversely to the number of tilings of those variables. In this scheme each subset of variables contributes equally to the learning of the agent. We evaluate whether this scheme improves learning performance.



Conclusions

We introduced the lunar lander task and discussed the implementation of an actor-critic policy-gradient agent for it. Various modifications were necessary for the algorithm to learn how to land the lunar module. We therefore consider this task a useful test-bed for investigating continuous action reinforcement learning algorithms.

Finally, we presented empirical evidence evaluating the impact of the algorithm modifications. The large improvement in performance we achieved with the same underlying algorithm suggests that eligibility traces and the details of tile coding have a significant impact on real-world performance.

Project Website

To obtain source code for the simulator and agent, go to **https://github.com/roshanshariff/lunarlander**

References

Thomas Degris, Patrick M. Pilarski, and Richard S. Sutton, "Model-Free Reinforcement Learning with Continuous Action in Practice," in *American Control Conference (ACC)*, Montreal, QC, Canada, 2012, pp. 2177–2182.

Eran Guendelman, Robert Bridson, and Ronald Fedkiw, "Nonconvex Rigid Bodies with Stacking," *ACM Transactions on Graphics (TOG)*, vol. 22, no. 3, pp. 871–878, July, 2003.

Richard S. Sutton, David McAllester, Satinder Singh, and Yishay Mansour, "Policy Gradient Methods for Reinforcement Learning with Function Approximation," in *Advances in Neural Information Processing Systems (NIPS)*, Denver, CO, USA, 2000, pp. 1057–1063.