
RL Project Template

Alireza Samar, Arnold Gomes, Danny Demarco,
Department of Computer Science
University of Bath
Bath, BA2 7AY
{as4731, ag2342, dd480, rdw41}@bath.ac.uk

1 Problem Definition

A clear, precise and concise description of your chosen problem, including the states, actions, transition dynamics, and the reward function. You will lose marks for an unclear, incorrect, or incomplete problem definition.

Application of reinforcement learning methods to modern artificial intelligence applications are both interesting and challenging, especially when the environment is unknown and there may be delayed rewards. This group project is an investigation into some of these methods and will attempt to train a number of agents to land a simulated spacecraft on the moon and compare their results for efficacy. To enable this the choice was made to make use of the LunarLander-v2 environment provided by OpenAI in their Gym package. The environment provides a classic rocket trajectory problem in which the rocket, starting at an elevated position is tasked with landing safely on a designated landing pad. There are two options in using this environment, discrete or continuous and we have chosen to use the discrete option. The 2d environment consists of the moon on the lower boundary with a section of it designated as the landing pad limited by the area between 2 flags, and an area of space above it in which the rocket may fly. The rocket itself is fitted with 2 legs to its base and three engines to provide thrust vertically or horizontally. To understand how the environment functions in more depth we can break it down into 4 main sections:

1. States

(a) Starting state

Each episode begins with the rocket in the same position; at the top, center of the viewport. The variable factor is that there is a random force applied to the rockets center of mass at the outset.

(b) Observation space

During an episode there are 8 possible states for the rocket to be in and consist of an 'x' and 'y' coordinate to denote its position, the linear velocity in both 'x' and 'y', the current angle of the rocket, the angular velocity, and two boolean values that represent whether each leg of the rocket is in contact with the ground or not. The landing pad is designated with (0,0) for its 'x' and 'y' coordinates.

2. Actions

There are 4 discrete actions available at any point: no action, fire right engine (left orientation), fire left engine (right orientation), fire main engine (upward orientation). Fuel is considered infinite and there is no range of acceleration, engines are binary and either on or off.

3. Rewards

Agents are incentivised/decentivised with the following reward structure:

(a) Positive Rewards

The reward for successfully moving from the starting state and coming to rest at the

landing pad range between 100-140 points depending on the accuracy of the landing. Each leg that is in contact with the ground provides an extra 10 points to incentivise a level approach to the landing pad. Once an episode is successfully solved, a reward of 200 points is awarded.

(b) Negative Rewards

To incentivise the agent to land on the platform as quickly as possible, negative rewards are given for each use of any of the engines in each frame of an episode of -0.3. If the rocket crashes (touches the moon), it receives -100 points.

4. Episodes

Each episode starts in the given starting state and runs until the terminal state is activated. A terminal state is activated in any of the following instances:

- (a) The rocket moves outside of the viewport (moves too far left, right, or up)
- (b) The rocket crashes into the moon (misses the landing platform and hits the lower boundary)
- (c) A sleep state is reached, i.e. not in motion and not involved in a collision. This includes once the rocket has successfully landed on the landing platform.

With these particulars in mind we can see that our agent is incentivised to direct the rocket from its starting position, to the landing pad with the minimal amount of engine use (fuel) as possible, while keeping the rocket stabilised as much as possible so that its legs may both make contact with the landing pad simultaneously. Incentivisation is produced by a large positive reward for reaching its platform, with bonus points for landing there in an upright position, and negative rewards for moving too far off course or colliding with the moons surface.

2 Background

Testing danny branch

A discussion of reinforcement learning methods that may be effective at solving your chosen problem, their strengths and weaknesses for your chosen problem, and any existing results in the scientific literature (or publicly available online) on your chosen problem or similar problems.

3 Method

A description of the method(s) used to solve your chosen problem, an explanation of how these methods work (in your own words), and an explanation of why you chose these «««« Updated upstream specific methods.

4 Results

A presentation of your results, showing how quickly and how well your agent(s) learn (i.e. improve their policies). Include informative baselines for comparison (e.g. the best possible performance, the performance of an average human, or the performance of an agent that selects actions randomly).

5 Discussion

An evaluation of how well you solved your chosen problem.

===== specific methods.

It was decided that 3 algorithms would be implemented to train our agents so that the different approaches could be compared and contrasted for efficacy against this particular problem. The 3 algorithms selected were; REINFORCE, a traditional policy gradient method chosen as a baseline method, and the more modern DQN and PPO algorithms. To give a better understanding of our chosen method PPO, we will first introduce TRPO, as this is where this method is derived from.

5.1 TRPO

Trust Region Policy Optimization (TRPO) (?) is a policy-gradient method for on-policy, online learning. It uses the concept of a 'Trust Region' in policy space, which permits the policy to move only within a specific 'trusted' bound per gradient ascent step. This mitigates the risk of an agent moving into a region of policy space from which it cannot recover from a sample collected under a poor policy.

An advantage estimate is calculated using a sampled discounted return under a policy π , parameterised by θ , from an episode rollout, and a noisy value function estimate from a function approximator. Importance sampling is used to calculate the policy gradient from the old policy with a different sample distribution, and as such achieves greater sample efficiency than vanilla policy gradient methods (which use current policy to compute gradient). The objective function and target for optimisation is: $\max_{\theta} \mathbb{E}_t \left[\frac{\pi_{\theta}(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \hat{A}_t \right]$

subject to $\mathbb{E}_t [KL[\pi_{\theta_{old}}(\cdot|s_t), \pi_{\theta}(\cdot|s_t)]] \leq \delta$ Where KL is Kullback-Leibler Divergence (?), and δ our trust region bound. Whilst the specifics of KL-Divergence are outside the scope of this project, it is essentially a statistical measure of the distance between two probability distributions; the policy update is subject to the constraint that KL-Divergence between the new and old policy must not exceed some hyperparameter δ for a gradient step. TRPO naturally falls into an Actor-Critic type model, where the value estimation is performed by the critic and the actor encodes the policy distribution.

5.2 PPO

Proximal Policy Optimization (?) is a simplification and improvement over TRPO. Problems with TRPO were perceived to be complexity of implementation, and tractability of computation with regards to the KL-Divergence bound, which is an expensive operation with second-order optimisation required. A simpler bound was found to be more effective:

$$L^{CLIP}(\theta) = \mathbb{E}_t \left[\min(r_t(\theta)\hat{A}^t, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon)\hat{A}^t) \right] \quad (1)$$

Where $r_t(\theta)$ is identical to the probability ratio from TRPO ($\frac{\pi_{\theta}(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$). The clip function replaces the KL-Divergence constraint, which clips the policy probability ratio such that the change does not exceed some hyperparameter ϵ , typically 0.2 (?). In addition to this, PPO provides some non-trivial code-level optimisations which significantly improve performance over TRPO (?).

PPO was chosen for the proposed domain as it is relatively simple to implement, provides relatively good sample efficiency over other policy gradient methods, and is not particularly sensitive to hyperparameters (?).

5.3 REINFORCE

The final method we chose for this environment was REINFORCE (?). REINFORCE is a traditional policy gradient method, but there are conflicting opinions about how exactly it should be implemented, possibly because as ? defines it is fairly vague. The update function for network parameters θ is defined as (?):

$$\theta_{t+1} = \theta_t + \alpha G_t \frac{\Delta_{\theta} \pi(A_t|S_t, \theta_t)}{\pi(A_t|S_t, \theta_t)} \quad (2)$$

Where α is a hyperparameter controlling learning rate and G_t is our Monte-Carlo sampled discounted returns. By exploiting the likelihood ratio trick (?), we can rewrite this as:

$$\theta_{t+1} = \theta_t + \alpha G_t \Delta_{\theta} \log \pi(A_t|S_t, \theta_t) \quad (3)$$

Notice that unlike PPO, this update method only takes into account current policy, and as such results in much lower sample efficiency. In many instances, due to the high variance of Monte-Carlo sampling methods, an arbitrary baseline $b(s)$ is subtracted from G_t in the update. $b(s)$ does not vary with a and as such reduces per-update variance.

The conflict surrounding implementation details for REINFORCE is with regard to the frequency of updates. Some accounts (?) suggest updates to the network should be done at every

timestep, Others (?) suggest updates on some time horizon T , and ? suggests a number of update points, including per-episode (i.e. traditional Monte-Carlo) updates. We chose to implement REINFORCE with per-episode updates, and no baseline.

Using REINFORCE as a baseline policy gradient method, we are able to compare the results of our more modern PPO and DQN algorithms.

6 Results and Discussion

Algorithms were coded in base PyTorch (?). Due to time constraints, Bayesian Optimization was not carried out to discover optimal hyperparameters. As a result, hyperparameters were sourced from the original papers, or by consulting other online implementations and heuristic search. We make no claim that the hyperparameters used in the below experiments are optimal. Algorithms were run 10 times, and the graphs below are the average of these runs. Linear interpolation was used to fit data points per run to the average run length. Full graphs for every run can be found in Appendix X.

6.1 PPO

PPO solved the Lunar Lander environment in an average of 608 episodes with a standard deviation of 102.6: »»»»> Stashed changes

7 Future Work

A discussion of potential future work you would complete if you had more time.

8 Personal Experience

A discussion of your personal experience with the project, such as difficulties or pleasant surprises you encountered while completing it.

References

Appendices

If you have additional content that you would like to include in the appendices, please do so here. There is no limit to the length of your appendices, but we are not obliged to read them in their entirety while marking. The main body of your report should contain all essential information, and content in the appendices should be clearly referenced where it's needed elsewhere.

Appendices should include (1) a detailed description of the problem domain, including the states, actions, reward function, and transition dynamics; (2) all experimental details so that the reader can fully replicate your experiments; and (3) how you selected your hyperparameters (if applicable).

Appendix A: Example Appendix 1

Appendix B: Example Appendix 2



Figure 1: (left-to-right) Average reward, episode length and loss for PPO.