RL Project Template

Alireza Samar, Arnold Gomes, Danny Demarco, Tyler Christie
Department of Computer Science
University of Bath
Bath, BA2 7AY
{as4731, ag2342, dd480, rdw41, tc905}@bath.ac.uk

1 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 succesfully 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 succesfully 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 succesfully 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

Reinforcement learning is a powerful tool for decision making under uncertainty. Previous work has been done in solving the lunar lander environment using different techniques. This section will briefly introduce MDPs, Bellman Equation, Policy Gradient Methods, and other methods and then describe their strengths and weaknesses in solving the lunar lander problem.

Markov-decision processes (MDPs) are a type of mathematical model used in decision analysis and operations research. MDPs are used to model decision making in situations where the outcomes of a decision are partly determined by chance and partly by the decision maker's actions. In an MDP, the decision-maker is represented by a "agent" who takes actions in a discrete set of possible states, and a random process determines the state of the world at any given time. The MDP framework allows us to calculate the expected value of any decision, given the agent's current state and the probabilities of different outcomes in each state (Sutton, Precup and Singh, 1999).

MDPs are also used to model problems in machine learning and control theory. In machine learning, MDPs can be used to model the problem of learning how to take actions that maximize a reward. In control theory, MDPs can be used to model the problem of controlling a system to maximize a particular criterion, such as the expected value of the reward.

A common approach for such a state and action pair is Q-learning. In this approach, a Q-table gets updated iteratively to maximize the reward of every given state. This approach is framed by the Bellman Equation:

$$Q^*(s,a) = \mathbb{E}_{s' \sim \varepsilon}[r + \gamma max_{a'} Q^*(s^{'}, a^{'})|s,a] \tag{1}$$

Another common approach to solving MDPs is dynamic programming. Dynamic programming is a technique for breaking down a problem into smaller subproblems, solving each subproblem, and then combining the solutions to the subproblems to solve the original problem. This approach can be used in RL because the agent can learn which actions lead to the best outcomes by solving smaller MDPs.

There are a few different ways to implement dynamic programming for RL problems. The most common approach is to use a table to store the solutions to the subproblems. The table can be updated as the agent learns which actions lead to the best outcomes (Lewis and Vrabie, 2009).

The current reward and discounted maximum future reward can be concluded as the optimal value for a given pair of action-state. Since the number of Q-values would be computed as finite, the typical approach to solving iteratively is to use dynamic programming:

$$Q_{i+1}(s,a) = \mathbb{E}[r + \gamma max_{a'} Q_i(s^{'}, a^{'})|s,a], \lim_{i \to \infty} Q_i = Q^* \tag{2}$$

On the other hand, the policy gradient algorithm works by estimating the gradient of the expected reward function concerning the policy and then adjusting the policy accordingly.

One of the benefits of the policy gradient algorithm is that it is relatively easy to implement. Additionally, the algorithm can adapt to changes in the environment, allowing the agent to learn new tasks and strategies.

In contrast, while described approaches try to optimise the Bellman Equation, the policy gradient aims to maximise the return by converging toward its policy.

Finally, Vanilla Policy Gradient (VPG) trains a policy known as the on-policy way (Peters and Schaal, 2006), which is different and often less sample efficient than the off-policy way, such as Deep Q-Network (DQN) (Mnih et al., 2015).

While the off-policy methods take advantage of experiences from previous policies, the policy gradient cannot reduce the bias by reusing experience in picking trajectories of prior policies. This difference describes why policy gradient requires significantly higher training episods to return average (Gadgil, Xin and Xu, 2020).

3 Method

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.

3.1 TRPO

Trust Region Policy Optimization (TRPO) (Schulman et al., 2015) 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:

$$\max_{\theta} \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \hat{A}_t \right] \tag{3}$$

$$\text{subject to } \hat{\mathbb{E}}_t \left[KL \left[\pi_{\theta_{old}}(\cdot|s_t), \pi_\theta(\cdot|s_t) \right] \right] \leq \delta \tag{4}$$

Where KL is Kullback-Leibler Divergence (Joyce, 2011), 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.

3.2 PPO

Proximal Policy Optimization (Schulman et al., 2017) 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:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[min(r_t(\theta)\hat{A}^t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}^t) \right] \tag{5}$$

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 (Schulman et al., 2017). In addition to this, PPO provides some non-trivial code-level optimisations which significantly improve performance over TRPO (Engstrom et al., 2020).

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 (Schulman et al., 2017).

3.3 DQN

DQN (Deep-Q Networks) (Mnih et al., 2015) were the first foray into combining traditional reinforcement learning methods, Q-Learning (Watkins and Dayan, 1992), with deep neural networks. Revolutionary at the time, it allowed an agent to approximate the Q values of previously intractable large state/action spaces. Using these networks in place of the policy $\pi(s, a)$, or value $q(s, a)$ functions.

They work by feeding in the observable information of the environment, then training the network parameters via gradient descent with the aim of minimising a chosen loss function, typically the Mean Squared Error. The algorithm functions similarly to function approximated Q-learning, however it also incorporates a Replay memory and a target network, to help deal with the instability from function approximators. By using a two separate networks, learning is performed from the stable target network onto the local network. The Replay memory is used to store all transitions (S_t, A_t, R_t, S_{t+1}) encountered in the environment, then periodically a batch is sampled from these stored transitions and used to perform an update on the local network parameters, defined as:

$$\begin{aligned} &\text{Set} y_j = \begin{cases} r_j & \text{for terminal } s_{t+1} \\ r_j + \gamma \text{max}_{a'} Q(s_{t+1}, a'; \theta_2) & \text{for non terminal } s_{t+1} \end{cases} \\ &\text{Perform gradient descent on } \nabla_{\theta_1} L_\delta(y_j, \hat{q}_1(S_j, A_j; \boldsymbol{\theta}_1)) \end{aligned} \tag{6}$$

In which every C steps the target network θ_2 is updated to the local network θ_1.

DQN was chosen for its historical significance, as it has led to many further improvements in modern reinforcement learning, such as the direct descendant Hessel et al. (2018) an amalgamation of many individual improvements to DQN, which was state of the art for its time, or any other deep reinforcement learning method which have built of the work of Mnih et al. (2015).

3.4 REINFORCE

The final method we chose for this environment was REINFORCE (Williams, 1992). REINFORCE is a traditional policy gradient method, but there are conflicting opinions about how exactly it should be implemented, possibly because as Williams (1992) defines it is fairly vague. The update function for network parameters θ is defined as (Sutton and Barto, 2018):

$$\theta_{t+1} = \theta_t + \alpha G_t \frac{\Delta_\theta \pi(A_t|S_t, \theta_t)}{\pi(A_t|S_t, \theta_t)} \tag{7}$$

Where α is a hyperparameter controlling learning rate and G_t is our Monte-Carlo sampled discounted returns. By exploiting the likelihood ratio trick (Silver, 2015), we can rewrite this as:

$$\theta_{t+1} = \theta_t + \alpha G_t \Delta_\theta \text{log} \pi(A_t|S_t, \theta_t) \tag{8}$$

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 (Sutton and Barto, 2018) suggest updates to the network should be

done at every timestep, Others (Silver, 2015) suggest updates on some time horizon T, and Williams (1992) 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.

4 Results and Discussion

Algorithms were coded in base PyTorch (Paszke et al., 2019). 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.

4.1 PPO

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

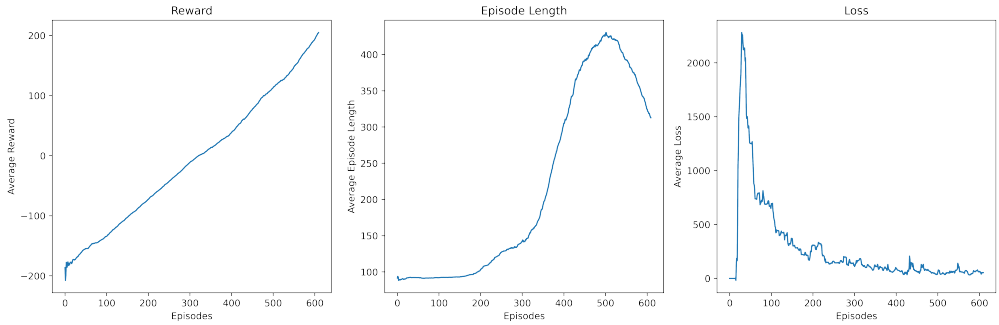

Figure 1: (left-to-right) Average reward, episode length and loss for PPO.

5 Future Work

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

6 Personal Experience

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

References

Engstrom, L., Ilyas, A., Santurkar, S., Tsipras, D., Janoos, F., Rudolph, L. and Madry, A., 2020. Implementation matters in deep rl: A case study on ppo and trpo. *International conference on learning representations* [Online]. Available from: `https://openreview.net/forum?id=r1etN1rtPB`.

Gadgil, S., Xin, Y. and Xu, C., 2020. Solving the lunar lander problem under uncertainty using reinforcement learning. *Southeastcon 2020*.

Hessel, M., Modayil, J., Van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M. and Silver, D., 2018. Rainbow: Combining improvements in deep reinforcement learning. *Thirty-second aaai conference on artificial intelligence*.

Joyce, J.M., 2011. *Kullback-leibler divergence*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp.720–722. Available from: https://doi.org/10.1007/978-3-642-04898-2_327.

Lewis, F.L. and Vrabie, D., 2009. Reinforcement learning and adaptive dynamic programming for feedback control. *Ieee circuits and systems magazine*, 9(3), pp.32–50.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G. et al., 2015. Human-level control through deep reinforcement learning. *nature*, 518(7540), pp.529–533.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J. and Chintala, S., 2019. Pytorch: An imperative style, high-performance deep learning library. In: H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox and R. Garnett, eds. *Advances in neural information processing systems 32* [Online]. Curran Associates, Inc., pp.8024–8035. Available from: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

Peters, J. and Schaal, S., 2006. Policy gradient methods for robotics. *2006 ieee/rsj international conference on intelligent robots and systems*. IEEE, pp.2219–2225.

Schulman, J., Levine, S., Abbeel, P., Jordan, M. and Moritz, P., 2015. Trust region policy optimization. In: F. Bach and D. Blei, eds. *Proceedings of the 32nd international conference on machine learning* [Online]. Lille, France: PMLR, *Proceedings of Machine Learning Research*, vol. 37, pp.1889–1897. Available from: https://proceedings.mlr.press/v37/schulman15.html.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A. and Klimov, O., 2017. Proximal policy optimization algorithms. *arxiv preprint arxiv:1707.06347*.

Silver, D., 2015. Lectures on reinforcement learning. URL: https://www.davidsilver.uk/teaching/.

Sutton, R.S. and Barto, A.G., 2018. *Reinforcement learning: An introduction*. Cambridge, MA, USA: A Bradford Book.

Sutton, R.S., Precup, D. and Singh, S., 1999. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2), pp.181–211.

Watkins, C.J. and Dayan, P., 1992. Q-learning. *Machine learning*, 8(3), pp.279–292.

Williams, R.J., 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3), pp.229–256.

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