Optimization and Engineering
https://doi.org/10.1007/s11081-021-09687-z

RESEARCH ARTICLE

Improving the efficiency of reinforcement learning for a spacecraft powered descent with Q-learning

Callum Wilson[1] · **Annalisa Riccardi**[1]

Received: 31 July 2020 / Revised: 18 June 2021 / Accepted: 1 September 2021

Abstract

Reinforcement learning entails many intuitive and useful approaches to solving various problems. Its main premise is to learn how to complete tasks by interacting with the environment and observing which actions are more optimal with respect to a reward signal. Methods from reinforcement learning have long been applied in aerospace and have more recently seen renewed interest in space applications. Problems in spacecraft control can benefit from the use of intelligent techniques when faced with significant uncertainties—as is common for space environments. Solving these control problems using reinforcement learning remains a challenge partly due to long training times and sensitivity in performance to hyperparameters which require careful tuning. In this work we seek to address both issues for a sample spacecraft control problem. To reduce training times compared to other approaches, we simplify the problem by discretising the action space and use a data-efficient algorithm to train the agent. Furthermore, we employ an automated approach to hyperparameter selection which optimises for a specified performance metric. Our approach is tested on a 3-DOF powered descent problem with uncertainties in the initial conditions. We run experiments with two different problem formulations—using a 'shaped' state representation to guide the agent and also a 'raw' state representation with unprocessed values of position, velocity and mass. The results show that an agent can learn a near-optimal policy efficiently by appropriately defining the action-space and state-space. Using the raw state representation led to 'reward-hacking' and poor performance, which highlights the importance of the problem and state-space formulation in successfully training reinforcement learning agents. In addition, we show that the optimal hyperparameters can vary significantly based on the choice of loss function. Using two sets of hyperparameters optimised for different loss functions, we demonstrate that in both cases the agent can find near-optimal policies with comparable performance to previously applied methods.

Keywords Intelligent Control · Reinforcement Learning · Spacecraft Powered Descent

Extended author information available on the last page of the article

Published online: 04 October 2021

1 Introduction

Space missions are becoming more ambitious in scope and will require new technologies and methods to fulfil increasingly stringent demands. One important aspect of these missions is the Guidance, Navigation, and Control (GNC) subsystem which must be able to safely transport payloads and possibly humans in very uncertain environments. This motivates the use of methods from Intelligent Control (IC) which can cope with these uncertainties while achieving near-optimal performance. Classical optimisation methods for control systems can fall apart when uncertainties in the environment are present (Gaudet et al. 2020b). In the context of landing on extra-terrestrial bodies, these uncertainties include atmospheric models, spacecraft aerodynamic coefficients, and initial conditions, among others (Quadrelli et al. 2015).

IC methods are designed to deal with substantial uncertainties where traditional methods cannot. Combining the fields of automatic control, artificial intelligence, and operational research (Saridis 1979), these methods can adapt online using Artificial Intelligence (AI) techniques such as fuzzy logic, machine learning, and evolutionary computing (Wilson et al. 2019). There are also classes of control methods which use AI methods that learn how to control a system offline and do not update in deployment. While these cannot be considered "intelligent" in the same way online adaptive intelligent controllers are, they are still very useful for uncertain environments. By incorporating uncertainties into the training process, a controller can learn how to handle uncertainties when deployed.

Neural Networks (NNs) are one of the most common AI architectures used in control and have been used for several decades (Hunt et al. 1992). This is largely due to their universal approximation capabilities and adaptability for online adaptive controllers. NNs are incorporated into classical control systems in a variety of ways such as to approximate system uncertainties, model the environment, or directly acting as a controller (Fazlyab et al. 2016; Ichikawa and Sawa 1992; Talebi et al. 2009). An alternative approach to using NNs which relates to optimal control theories is in the machine learning paradigm of Reinforcement Learning (RL) (Sutton and Barto 1998). In general, these methods use NNs to learn the optimality (specifically the expected return) of a control policy and improve this incrementally through interaction with the environment. This simple idea can be incredibly powerful; state-of-the-art methods now compete with the performance of humans in certain tasks which are considered to require human intelligence (Mnih et al. 2015; Silver et al. 2016).

As with many machine learning problems, one bottleneck to the use of RL in practice is training times (Duan et al. 2016b). Especially in environments with high-dimensional state or action spaces, the 'curse-of-dimensionality' results in an exponential increase in the size of the problem for the agent with increases in dimensionality. While recent works have shown success in learning from pixel level observations (Lillicrap et al. 2015), these systems require notably long training times. In these cases, as with many state-of-the-art RL applications, the agent operates in a discrete action space. Other works have also extended RL methods

to continuous action spaces (Mnih et al. 2016; Schulman et al. 2015; Van de Wiele et al. 2020). While this results in longer training times, many continuous action problems can instead be simplified by appropriately discretising the action space. Another limitation of certain RL algorithms is their sensitivity to hyperparameters (Henderson et al. 2018). Since some of the update rules used in RL are inherently unstable, this means their hyperparameters require careful selection to avoid converging on poor solutions. This is commonly done using rules of thumb or trial and error (Bergstra et al. 2011). Alternatively, optimising hyperparameters more methodically can improve the resulting performance. This is not a trivial task due to the high dimensionality of hyperparameter search spaces but is not intractable with current computational methods and capabilities.

The problem of learning speed in RL is well studied with various proposed improvements. Algorithms such as NFQ (Riedmiller 2005) and PILCO (Deisenroth and Rasmussen 2011) have been developed to improve the data efficiency of RL. More recently the concept of "meta-learning" has become more popular. This is where a learning algorithm is used to train a learning algorithm which then trains a policy. In doing this the higher-level learning algorithm trains over a long period such that the lower-level algorithm learns to be quick and responsive (Wang et al. 2017; Duan et al. 2016a). This approach has also recently been applied to a spacecraft guidance problem (Gaudet et al. 2020a). Automated hyperparameter selection is a very similar problem to meta-learning since it often uses a higher level learning procedure to "train" the hyperparameters of the lower level algorithm. These automated methods use a variety of intelligent approaches such as evolutionary computation (Schweighofer and Doya 2003; Young et al. 2015) and Bayesian optimisation methods (Barsce et al. 2017; Bergstra et al. 2011).

This work aims to find ways of speeding up learning times in a sample spacecraft control problem. This builds on work previously carried out by Gaudet and Furfaro (2014), Gaudet et al. (2020b) which considers the problem of a fuel-efficient pinpoint Mars landing. The nature of this problem lends itself well to the RL paradigm of learning through interaction and observing rewards. As a result, there is also such an environment in the OpenAI Gym suite of RL benchmarks (Brockman et al. 2016). The size of the problem in this benchmark environment is not representative of reality and so the authors in Gaudet et al. (2020b) propose a more realistic problem with substantial uncertainties in initial conditions. In their approach, they use Proximal Policy Optimisation (PPO)—a RL algorithm capable of operating in continuous action spaces (Schulman et al. 2017). This has good stability but requires optimisation online over trajectories which is not as data efficient as many alternative offline approaches. Here we aim to use the popular Q-learning algorithm (Watkins and Dayan 1992) to find near-optimal solutions to the problem in a discrete-action form. Furthermore, we use an automated sequential method to select hyperparameters according to different criteria and observe the effect this has on learning performance.

This paper is structured as follows. Below we give an overview of related work for the problem of powered descent guidance. Section 2 presents some fundamental concepts relating to RL and introduces the Q-learning algorithm which will be used to solve the lander problem. The problem of a lander powered descent is

then described in Sect. 3. Here we also describe the approach used to transform the action space from continuous to discrete. In Sect. 4 we present the results of the hyperparameter tuning, action space tuning, and from training the agent in the environment. This considers how efficiently the agent learns in comparison to previous approaches and looks at how well it achieves the goal in the presence of uncertainties. Section 5 gives conclusions and future research directions.

1.1 Related work

Autonomous GNC for extra-terrestrial landing has been a well studied problem dating back to the Apollo lunar missions (Song et al. 2020). These missions had tight constraints on computing power which necessitated simple and robust control algorithms to generate feasible trajectories online (Klumpp 1974). Following the success of the Apollo missions, there have been great advancements both in the computational capabilities onboard spacecraft and the methods for autonmous GNC. Most newer methods applied to this problem come under the field of Optimal Control, where a control system seeks to maximise or minimise a characteristic of a dynamical system (Bellman 1966). More recently, Intelligent Control methods have also gained interest for aerospace applications (Riccardi et al. 2018).

With renewed interest in missions to Mars, and its potential for future manned space missions, there has been more research in the past twenty years focusing on entry, descent, and landing for Mars missions (Braun et al. 2006). Limitations on spacecraft resources and increasingly stringent requirements on control system performance motivates the use of optimal control theories. These methods mostly minimise fuel consumption (or control effort), subject to constraints on the lander's final position and other aspects of the trajectory and actions (Meditch 1964). Certain traditional optimisation approaches, such as general nonlinear programming methods, can generate offline trajectories, but are unsuitable for use online due to the lack of guarantees on computational time needed (Mao et al. 2016). Instead, various approaches exist to transform the optimisation problem such that it can be solved using convex programming methods (Acikmese et al. 2013; Acikmese and Ploen 2007; Blackmore et al. 2010). This allows optimal trajectory generation online in bounded computational time.

Compared to optimal control, IC methods will make fewer assumptions on the system's dynamics, which allows them to consider greater uncertainties. This comes at the expense of removing the analytical guarantees on performance in optimal control. Instead, control methods using AI will most often give statistical guarantees on performance when subject to specified uncertainties. One example of RL applied to powered descent is shown in Jiang et al. (2019), which also incorporates pseudospectral optimal control methods. In this case, the RL controller is used to determine the 'handover' point at which the controller switches from entry-phase to powered descent-phase. Similarly, in Furfaro and Linares (2017) the authors combine RL with a ZEM/ZEV controller, which is another analytical method for descent trajectory generation (Guo et al. 2013). Their approach uses an RL agent to select waypoints for the ZEM/ZEV guidance.

Previous work has also investigated the possibility of using RL to fully derive the integrated GNC system for powered descent (Gaudet and Furfaro 2014; Gaudet et al. 2020b). As stated previously, the RL method employed in these works is PPO which uses continuous action spaces. Q-Learning is another RL method which has gained more attention recently due to the Deep Q-Networks (DQN) algorithm, which demonstrated human-level performance in the benchmark Atari game environments (Mnih et al. 2015). Here we investigate if the methods employed in DQN can also perform well in the powered descent task. To the authors' knowledge, this paper is the first application of this algorithm to the powered descent problem on this scale. We use the same problem formulation as in Gaudet et al. (2020b), which has a 'shaped' reward function designed to guide the agent towards the goal state (Ng 2003). These reward functions are susceptible to 'reward-hacking' where an agent converges on undesirable behaviour by exploiting an aspect of the reward function (Ng et al. 2000). Here we further demonstrate that, without using an appropriate corresponding state representation for the shaped reward, an agent can 'hack' the reward such that it seems to improve its reward, but does not learn a desirable policy. Unlike previous approaches to this problem, we also discretise the action domain to make it suitable for Q-Learning. Operating in discrete action spaces has its own set of challenges compared to continuous domains, especially where the action space is high dimensional with many discrete actions (Dulac-Arnold et al. 2015). We aim to simplify the problem complexity using few discrete actions and examine whether this is able to accelerate learning time while maintaining near-optimal performance.

2 Reinforcement learning for optimal control

This section provides the relevant background to RL and the key concepts used in this work. For a more comprehensive introduction into RL, readers are directed to the work of Sutton and Barto (1998).

2.1 RL fundamentals

A RL process consists of an agent which senses an environment in a certain state and carries out actions to maximise future reward. The only feedback the agent receives from the environment is a state signal and reward signal and it can only affect the environment by its actions as shown schematically in Fig. 1. The policy, π followed by the agent dictates the action that the agent will take given a certain state. This policy is updated as the process goes on with the goal of converging on the optimal policy, π^*. Following taking an action, the agent observes a reward, r and the new state. The goal from the agent's perspective is to maximise its return, or specifically its total (cumulative) discounted reward G_t following time-step t, where

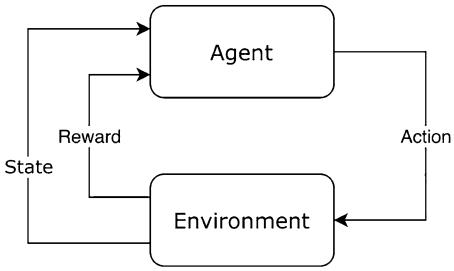


Fig. 1 Agent-environment interaction in reinforcement learning, where the agent observes a state and reward signal from the environment and uses this information to select an action to take

$$\begin{aligned} G_t &= r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots \\ &= \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \end{aligned} \tag{1}$$

The discount factor, γ affects the extent to which long term rewards are considered by the agent and has a value $0 \leq \gamma \leq 1$. As $\gamma \to 1$ longer term rewards are given more significance and for $\gamma = 0$ the only reward considered is that immediately following the current state.

Tasks in RL can be considered episodic or continuing based on the nature of the problem. Episodic tasks have clearly defined stopping criteria usually defined by "terminal" states, which mark the end of an episode once the agent reaches this state. Typically, a terminal state indicates success or failure of a task with a corresponding reward. Continuing tasks cannot be divided into episodes and do not have a stopping criteria as in episodic tasks. These problems require $\gamma < 1$ such that Equation 1 has a finite solution. Here we focus on episodic tasks which are the type of problem to which we apply our agent.

2.1.1 Value functions

Using the definition of total discounted reward from Equation 1, the state-value function, which is the expected return following state s, can then be defined as shown:

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi \left[G_t | s_t = s \right] \\ &= \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s \right] \end{aligned} \tag{2}$$

Similarly, the action-value function, i.e. the value of taking action a in state s, can be defined as

$$\begin{aligned} q_\pi(s,a) &= \mathbb{E}_\pi \left[G_t | s_t = s, a_t = a \right] \\ &= \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, a_t = a \right] \end{aligned} \tag{3}$$

In both Eqs. 2 and 3, $\mathbb{E}_\pi$ denotes the expected value when following policy π.

The policy which will maximise total discounted reward is called the optimal policy, π^*. By definition this is the policy where for all states the value function is greater than or equal to that of any other policy. The state-value function for an optimal policy is as shown:

$$v_*(s) = \max_\pi v_\pi(s) \tag{4}$$

This is called the optimal state-value function. Similarly, the action-value function for an optimal policy, called the optimal action-value function, is defined as:

$$q_*(s,a) = \max_\pi q_\pi(s,a) \tag{5}$$

2.1.2 Markov decision processes

An important concept for RL is that of the Markov property. A system possesses this property if the probability that the next state, s_{t+1} after the current state, s_t is the state s and the next reward to be observed, r_{t+1} is r is only dependent on the action, a_t carried out by the agent and the current state s_t. In symbolic form:

$$Pr\left(s_{t+1} = s, r_{t+1} = r | s_t, a_t\right) \tag{6}$$

This property is crucial as it has significant implications for the value function. For a system with the Markov property, all future states and rewards under a control policy π can be known at any given state. It follows that, for a Markov Decision Process (MDP) in which the Markov property applies, equations 2 and 3 can be solved for any state or state-action pair if the agent has sufficient experience of the environment. For many real systems, the Markov property does not apply but can be assumed such that the agent can obtain a near-optimal solution.

2.1.3 Generalised policy iteration

Three main categories of methods exist for solving MDPs such as this type of control problem: dynamic programming, Monte-Carlo methods, and temporal-difference learning (Sutton and Barto 1998). The common feature underpinning the three types of approach is Generalised Policy Iteration (GPI), where the agent's policy and value estimate for the policy are updated iteratively. This seeks to have the estimated value function of the policy converge to the true value function while also making the policy converge to the optimal policy.

One class of methods for reaching the optimal policy with GPI seeks to find either the optimal value function shown in Eq. 4 or optimal action-value function as in Eq. 5. Once the optimal value function is known, an optimal policy is then to take actions such that, from a given state s, the following state s' has the maximal value of all possible states following s. So to learn the optimal policy we can learn the optimal value function. In practise, this method is impractical due to the requirement of a perfect model which can estimate all possible state transitions. Instead, from the optimal action-value

function we can also derive an optimal policy which selects the action with maximal action-value in a given state. This is referred to as taking *greedy* actions—only choosing actions with the highest expected value.

An alternative to learning value functions is to learn a policy which maps states to actions. Then instead of updating value estimates which affect the policy, the agent directly updates its policy during learning. This policy is generally stochastic and parameterised as a probability distribution over actions for a given state such that the agent optimises the parameters of this distribution. The result is the same as for the value function methods described previously—in both cases the agent tends towards taking actions which maximise expected return.

Methods based on value functions have seen widespread use in recent state-of-the-art applications of RL. One issue with learning action-values is that this usually requires a discrete action space which limits the scope of problems they can solve. Control problems often involve continuous action spaces which can be discretised by specifying discrete magnitudes for each action dimension. This not only allows a greater scope of methods which can be used, but also simplifies the problem by making the number of actions finite. While this is an advantage, in certain classes of problems the true optimal solution requires continuous actions and so discretising the action space causes a loss in optimality. Here we use a discrete action space with the aim of decreasing learning time and observe the effect this has on the performance compared to continuous action algorithms.

2.2 Q-learning

A popular method for finding the optimal action-value function known as one-step Q-learning was devised by Watkins (1989). It is still widely used today and is one of the most significant influences on RL to date.

The update rule for one-step Q-learning is as follows:

$$Q(s_t, a_t) \leftarrow r_{t+1} + \gamma \max_a Q(s_{t+1}, a) \tag{7}$$

In practice, this rule is modified to include a step-size parameter α which controls the rate at which updates are made to action values. This is necessary to prevent action-values from increasing rapidly which would potentially result in over-estimations of action-values. The modified update rule is as shown:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right] \tag{8}$$

Compared to other methods for learning action-values, Q-learning is unique as its updates directly approximate the optimal action-value function through the 'max' operator. This differs from other methods such as SARSA (Sutton 1996) where updates depend on the policy being followed and hence future actions taken. To apply Q-learning in practise, the Q estimates are stored in a table with an entry for each state-action pair. By updating based on the maximum action-value function of s_{t+1} this algorithm can be proven to converge to q_* provided every state-action pair is

updated continuously (Watkins and Dayan 1992). This means any policy which can visit all states throughout its learning period is suitable for applying Q-learning.

2.2.1 Q-networks

Tabular methods of Q-learning suffer from increasingly large tables for larger state spaces—known as the "curse of dimensionality"—and only work in discrete state spaces. These limitations are removed by instead using NNs as function approximators which are more memory efficient for large state spaces and can handle continuous states. The fundamental disadvantage of using NNs within RL is that there is no proof of a guaranteed convergence as there is with tabular methods (Scherffig 2002). We refer to NNs used to approximate action-values as "Q-networks".

NNs are a useful tool capable of approximating any non-linear function (Barto et al. 1983). They have developed significantly over the past century and now employ many new and complex architectures such as the convolutional layers present in deep q-networks (Mnih et al. 2015). When training, a network's parameters, θ are adjusted to minimise a certain loss. For Q-networks, this loss is related to the Temporal Difference (TD)-error shown in Eq. 9, which is equivalent to the bracketed term from Eq. 8. Q_θ denotes the estimated action-value for network parameters θ. The loss is then most often the Mean Squared Error (MSE) across state-action pairs.

$$e = \begin{cases} Q_\theta(s_t, a_t) - r_{t+1}, & \text{if } s_{t+1} \text{ is terminal} \\ Q_\theta(s_t, a_t) - \left(r_{t+1} + \gamma \max_a Q_\theta(s_{t+1}, a)\right) & \text{otherwise} \end{cases} \tag{9}$$

2.3 Applying Q-learning

With the underlying theory established, here we describe how to apply Q-learning to control problems. First we describe the type of policy used to explore the environment while pursuing the optimal policy. Since its creation, Q-learning has also seen several developments and improvements to its data efficiency and stability, some of which we will introduce here.

2.3.1 ϵ-Greedy policies

As discussed in Sect. 2.2, to converge on the optimal policy an agent must visit all states and continuously update all state-action pairs while learning. At the same time, it must take actions already deemed optimal to maximise its return. This leads to an issue commonly referred to as "exploration-vs-exploitation".

In ϵ-greedy policies, the amount of exploration an agent does is controlled by the parameter ϵ, where $0 \leq \epsilon \leq 1$. At each timestep the agent will take a random action with probability ϵ, otherwise it takes a greedy action. Usually, random actions are sampled with equal probability from the action space, but this can be adjusted in cases where random actions could be detrimental to the agent. It is sensible for an

agent to explore more at the start of its training and decrease the amount of exploration as its Q estimates get closer to the optimal action-value function. There are several ways to do this in practise; here we use a simple method of linearly decreasing ϵ from and initial value of ϵ_i to a final value of ϵ_f. Both of these values are hyperparameters which can be tuned to the problem being considered. Additionally, the rate at which exploration decreases is controlled by the value N_ϵ, i.e. the number of episodes over which ϵ varies. The value of ϵ after n_{ep} episodes is then determined as shown in Eq. 10.

$$\epsilon = \max\left\{\epsilon_i - \frac{n_{ep}}{N_\epsilon}\left(\epsilon_i - \epsilon_f\right), \epsilon_f\right\} \tag{10}$$

2.3.2 Experience replay

Q-learning in its original form performs updates on the most recently observed state transition and updates at every timestep. However, since Q-learning is an off-policy update method it does not need to update over whole trajectories taken by the policy, just the individual state transitions. Furthermore, this can be done with multiple state transitions at each timestep. To do this, an agent saves its "experiences" of state transitions into a replay memory and performs updates on minibatches of experience sampled from this memory. This is the idea of experience replay, which has two main benefits. First, the agent's data efficiency is improved by updating on multiple transitions at every timestep. Second, "one-step" Q-learning immediately loses information about a state transition after updating which could be useful—especially for rarely visited states. With experience replay every state transition is likely to be used multiple times to update the network including seldom visited but potentially valuable states.

The agent's memory is denoted $\mathcal{D}$ and contains for each timestep, t, tuples of the form $\left(s_t, a_t, r_{t+1}, s_{t+1}\right)$ which make up an entire state transition. When updating the network, the agent samples a minibatch of k experiences from memory. In our case, experiences are sampled with equal probability. For each state transition in the minibatch, the TD-error is computed using Eq. 9 and the network is updated using the MSE. For problems with longer training periods it is necessary to limit the replay memory size. In this case we use a parameter N_{mem} which is the maximum number of transitions which can be stored. Old transitions are then removed on a first-in-first-out basis.

2.3.3 Target network

It has been noted that Q-networks are inherently unstable and weight updates tend to diverge and cause an "explosion" in their values. This is due to the 'max' operator in Eq. 7 which as noted can result in over-estimations of the action value. One simple method for mitigating this issue is to use a separate network for approximating target action-values—referred to as a "target network". The target network is initialised identical to the main Q-network but does not update its own weights. Instead, the

weights of the main Q-network are periodically copied to the target network. This allows the targets to also converge on the optimal action-value function while decoupling the weight updates from this prediction.

The parameters of the target network are denoted θ^-. The TD-error used to update the Q-network from Eq. 9 can then be written as shown for non-terminal s_{t+1}:

$$e = Q_\theta(s_t, a_t) - \left(r_{t+1} + \gamma \max_a Q_{\theta^-}(s_{t+1}, a)\right) \tag{11}$$

2.3.4 Algorithm for Q-learning

Algorithm 1 shows the procedure we use to apply Q-learning with the components described previously. This is the same as the DQN algorithm (Mnih et al. 2015) with the only difference being the agent's NN architecture. In the problem considered here, features are already extracted from the environment and so shallower NNs with only fully connected layers instead of convolutional layers suffice to solve the problem.

Algorithm 1 Q-Network algorithm

initialise network with random weights
initialise $\epsilon \leftarrow \epsilon_i$
$\delta_\epsilon \leftarrow \frac{(\epsilon_i - \epsilon_f)}{N_\epsilon}$
for episode= 1 to N_{ep} **do**
  initialise state $s_t \leftarrow s_0$
  while state s_t is non-terminal **do**
    select action a_t according to policy π
    execute action a_t and observe r_{t+1}, s_{t+1}
    update memory $\mathcal{D}$ with $(s_t, a_t, r_{t+1}, s_{t+1})$
    select random minibatch of k experiences $(s_j, a_j, r_{j+1}, s_{j+1})$ from $\mathcal{D}$
    $e_j = \begin{cases} Q_\theta(s_j, a_j) - r_{j+1} & \text{if } s_{j+1} \text{ is terminal} \\ Q_\theta(s_j, a_j) - \left(r_{j+1} + \gamma \max_a Q_{\theta^-}(s_{j+1}, a)\right) & \text{otherwise} \end{cases}$
    update network using the error e_j for each transition in the minibatch
    after C time-steps set $\theta^- \leftarrow \theta$
  end while
  $\epsilon \leftarrow \max\{\epsilon - \delta_\epsilon, \epsilon_f\}$
end for

3 Powered descent problem

We use a 3-Degrees of Freedom (DOF) lander problem obtained from Gaudet et al. (2020b). This section gives an overview of the environment properties; for a full description we direct readers to the original paper.

3.1 Environment description

In the powered descent problem, an agent—in this case the lander—starts from one of a range of possible initial conditions a certain altitude and lateral distance from a landing site. The goal of the agent is to make a soft pinpoint landing at the desired location while minimising fuel consumption. There are three dimensions in the action space which represent each of the lander's body frame axes along which it may produce thrust. In this 3-DOF problem, the lander's motion is only translational with no rotational components. The action space, here denoted $\mathbf{T}$, is the commanded thrust for each thruster. The body frame force acting on the lander, $\mathbf{F}^B$ is then given as shown:

$$\mathbf{F}^B = \mathbf{T} = \begin{Bmatrix} T^x_{cmd} \\ T^y_{cmd} \\ T^z_{cmd} \end{Bmatrix} \tag{12}$$

In this 3-DOF case the inertial frame is equivalent to the body frame and so $\mathbf{F}^B = \mathbf{F}^N$. The equations of motion are shown below:

$$\dot{\mathbf{r}} = \mathbf{v} \tag{13}$$

$$\dot{\mathbf{v}} = \frac{\mathbf{F}^N}{m} + \mathbf{g} \tag{14}$$

$$\dot{m} = -\frac{\Sigma_{i=1}^{3} F_i^B}{I_{sp} g_{ref}} \tag{15}$$

where m is the lander's mass, $\mathbf{r}$ is the position, and $\mathbf{v}$ is the velocity. Constant values in these equations are $\mathbf{g} = [0\ 0\ -3.7114]^T N/kg$ (gravitational field strength on Mars), $g_{ref} = 9.81$ N/kg, and $I_{sp} = 210$ s. The sampling time used for transmitting data to the agent is $0.2s$ which is the frequency at which the agent receives state observations.

One of the key problems with achieving a pinpoint landing is the large range of possible initial conditions in which the lander can be located. The implemented environment incorporates this uncertainty in the training process by sampling over a wide range of initial conditions in training episodes. The range of possible conditions for each relevant state value are shown in Table 1.

3.2 Shaped reward function

This problem can be readily considered as a sparse reward problem. This is where all state transitions except to a terminal state yield zero reward and the terminal reward indicates either success or failure of the task. These problems are particularly challenging for RL agents since they have to explore the environment to 'find' the

Table 1 Range of initial conditions in the lander environment

Parameter	Min. Value	Max. Value
Downrange position	0	2000
Crossrange position	− 1000	1000
Elevation position	2300	2400
Downrange velocity	− 70	− 10
Crossrange velocity	− 30	30
Elevation velocity	− 90	− 70

Position values in m and velocity values in m/s

correct terminal state which can take many trials. On the other hand, minimising fuel consumption can easily be defined as a reward at every step which is inversely proportional to the amount of fuel consumed. Combining these two aspects together yields a situation where the agent is effectively discouraged from exploring by the negative rewards received from consuming fuel. This obviously makes the agent less likely to reach the target terminal state.

One way of dealing with this problem as presented in Gaudet et al. (2020b) is to use a shaped reward function which guides the learning towards the goal. Although this can cause a loss in optimality with respect to fuel consumption, this vastly speeds up the learning by avoiding excessively long periods of exploration. The reward function used in this environment is a weighted sum of different terms which either penalise fuel consumption or constraint violations, or motivate the agent towards the target landing zone. The rationale behind each term is discussed in the original work and stated here:

$$\begin{aligned} r = \alpha\left\|\mathbf{v} - \mathbf{v}_{targ}\right\| + \beta\left\|\mathbf{F}^{B}\right\| + \eta \\ + \kappa\left(r_z < 0 and \|\mathbf{r}\| < r_{lim} and \|\mathbf{v}\| < v_{lim} and gs < gs_{lim}\right) \end{aligned} \tag{16}$$

where the following quantities are defined:

$$\mathbf{v}_{targ} = -v_0\left(\frac{\hat{\mathbf{r}}}{\|\hat{\mathbf{r}}\|}\right)\left(1 - exp\left(-\frac{t_{go}}{\tau}\right)\right) \tag{17}$$

$$v_0 = \left\|\mathbf{v}_0\right\| \tag{18}$$

$$t_{go} = \frac{\|\hat{\mathbf{r}}\|}{\|\hat{\mathbf{v}}\|} \tag{19}$$

$$\hat{\mathbf{r}} = \begin{cases} \mathbf{r} - [0\ 0\ 15], & \text{if } r_z > 15 \\ [0\ 0\ r_z], & \text{otherwise} \end{cases} \tag{20}$$

$$\hat{\mathbf{v}} = \begin{cases} \mathbf{v} - [0\ 0\ -2], \text{ if } r_z > 15 \\ \mathbf{v} - [0\ 0\ -1], \text{ otherwise} \end{cases} \tag{21}$$

$$\tau = \begin{cases} \tau_1, \text{ if } r_z > 15 \\ \tau_2, \text{ otherwise} \end{cases} \tag{22}$$

Equation 17 can be imagined as a velocity field pointing towards the target landing location. The magnitude of this velocity decreases exponentially as the position becomes closer to the target and from 15m above the landing location the target velocity is straight downwards with the aim of creating a soft, vertical landing. Per the first term in Eq. 16, the agent is encouraged to follow this velocity. Then the second term motivates minimising the fuel consumption and the third term, η is a positive constant which motivates the agent to continue through the environment towards the goal. The final term rewards a soft landing within state limits of position r_{lim}, velocity v_{lim}, and glideslope gs_{lim}.

The constants α, β, η, and κ are selected to weight each of the terms in the reward function. We use the same values as in Gaudet et al. (2020b) of $\alpha = -0.01$, $\beta = -0.05$, $\eta = 0.01$, and $\kappa = 10$. In addition, τ_1 and τ_2 can be selected to tune the magnitude of the velocity field. Again we use the same values from Gaudet et al. (2020b) of $\tau_1 = 20$ s and $\tau_2 = 100$ s. The final state limits are specified as $r_{lim} = 5$ m, $v_{lim} = 2$ m/s, and $gs_{lim} = 79°$.

Like the reward function, careful selection of the state representation can assist the agent in learning an effective policy. For such classes of landing problems, common state representations for control include quaternion or cartesian position (and body-frame angles for 6-dof problems) (Battin 1999). In this case where the landing target has a fixed position, cartesian state representation is inefficient since it does not exploit the rotational symmetry of the problem.

The state representation used in this environment is closely related to the shaped reward function and is shown in Eq. 23. Its main component is the error between the spacecraft's velocity and the target velocity for each component. It also includes the parameter t_{go} from Eq. 19 which gives a crude estimate of the remaining time before landing based on the position and target velocity. The final component of the state representation is the altitude, r_z, which is the most useful distance measure for deciding actions.

$$s = \left[v^x - v^x_{targ},\ v^y - v^y_{targ},\ v^z - v^z_{targ},\ t_{go},\ r_z\right] \tag{23}$$

In addition to this shaped state representation, we also compare the results when training an agent using a 'raw' state representation as shown in Eq. 24. Instead of using the velocity error, the agent uses as input the position, velocity, and mass of the spacecraft at the current timestep.

$$s = \left[r_x,\ r_y,\ r_z,\ v_x,\ v_y,\ v_z,\ m\right] \tag{24}$$

3.3 Discretised action space

As discussed previously, while Q-learning can operate in continuous state spaces with more advanced implementations (Van de Wiele et al. 2020), in its most common form it requires a discrete action space for maximising across actions. The action space for the lander is the commanded thrust for each engine which can be continuous or discretised with respect to a maximum thrust magnitude. In Gaudet et al. (2020b) the agent uses a continuous action space with limits on the total thrust from all of the engines. Using a discrete action space requires sensible maximum values for each thruster to be defined as will be discussed in Sect. 4.

In this environment the thrusters are oriented along each axis of the lander's body frame and each thruster can provide a force in the positive or negative direction of its axis. Since it is usually redundant to produce a force in the negative z-direction—a force already provided by gravity—here we constrain the agent to only choose positive thrusts for the z-thruster. The range of possible thrust commands the agent can give is then specified as shown:

$$\mathbf{T} = \begin{Bmatrix} [-1,1] \cdot T^x_{max} \\ [-1,1] \cdot T^y_{max} \\ [0,1] \cdot T^z_{max} \end{Bmatrix} \tag{25}$$

where T^i_{max} is the maximum possible thrust in the i-direction. The simplest way to discretise this action space is to allow possible actions of $\{-1, 0, 1\} \cdot T^i_{max}$ in the x- and y-direction and $\{0, 1\} \cdot T^z_{max}$ in the z-direction. Initial experiments showed it was beneficial to include an intermediate action in the z-direction instead of this on/off action and so for the simplest case we use $\{0, 0.5, 1\} \cdot T^z_{max}$ as the possible actions for this. 3 dimensions with 3 possible actions gives an action space size of $3^3 = 27$, where we assume each engine is controlled independently of the others.

4 Results of Q-learning applied to powered descent problem

Here we present the results obtained from applying the Q-Learning algorithm to the previously described lander problem. These are presented in four main parts. First, we show the procedure used for optimising hyperparameters with two different loss functions. We discuss the differences in the values obtained for each and will later compare their performance. Second, we look at the action size selection, i.e. selecting an appropriate value for T^i_{max} and observe the effect its value has on certain performance measures. Third, we compare our results to those obtained using PPO in a continuous action space. Finally, we show the results from applying the method with a raw state representation. Experiments were run on an Ubuntu 18.04 computer with a 3.6 GHz Intel i7-4790 CPU and 8 GB RAM.

4.1 Hyperparameter optimisation

The main problem when optimising hyperparameters is the search space, which can contain mixtures of real, integer, and possibly categorical values. In addition, the number of hyperparameters can be large resulting in a significantly high-dimensional search space which is difficult to optimise. One approach to the problem of automatically selecting hyperparameters is to perform a random search and select the best configuration, which can be surprisingly effective (Bergstra and Bengio 2012). Improvements to the random search include Hyperband, which uses a bandit based approach to speed up evaluation (Li et al. 2017). This method assumes that an algorithm's performance in early training epochs can be used to indicate later training performance, which does not hold well when training RL agents. Other automated hyperparameter search methods incorporate Bayesian optimisation methods, such as Spearmint (Snoek et al. 2012), Tree of Parsen Estimators (TPE) (Bergstra et al. 2011), and Sequential Model-based Algorithm Configuration (SMAC) (Hutter et al. 2011). Here we use TPE since this method has been applied successfully in computer vision tasks which possess the problems previously described. The implementation used here is from the Python library "hyperopt", which is designed for applying TPE to machine learning models (Bergstra et al. 2013).

We fix the number of hidden layers in the Q-network as 3 and optimise the number of hidden units in each layer, denoted $\tilde{N}_i$ for the ith hidden layer. The other hyperparameters related to the Q-network are the learning rate, α and target network update steps, C. In this case the Q-learning algorithm has 4 additional hyperparameters: initial exploration probability, ϵ_i; number of episodes to decrease ϵ, N_ϵ; discount factor, γ; and minibatch size, k. The final exploration probability ϵ_f is fixed as 0. To select the number of episodes over which to train agents in all experiments, we first trained an agent for 10,000 episodes. This used an untuned, but stable, hyperparameter configuration as follows: $\tilde{N}_1 = 100$, $\tilde{N}_2 = 150$, $\tilde{N}_3 = 100$, $\alpha = 2e-5$, $C = 65$, $\epsilon_i = 0.5$, $N_\epsilon = 2000$, $\gamma = 0.95$, $k = 100$. The resulting learning curve is shown in Fig. 2. Over the first 1000 episodes the reward oscillates before rapidly increasing. After approximately 1500 episodes this rate of increase declines and remains very gradual for the rest of the episodes. To strike a balance between maximising performance and minimising training duration, we chose 4000 episodes as the duration for training—both for hyperparameter optimisation and training with the selected hyperparameters.

In the context of RL problems there are several ways to define optimisation criteria for hyperparameters. Since the agent's main goal is to maximise its total reward, one possible criterion is the total reward at the end of training. Here we also aim to learn a near-optimal policy in the shortest number of training episodes possible. This can effectively be achieved by minimising the area under the learning curve—where the cumulative rewards are negative.

From here we present results from two separate hyperparameter optimisation studies: reward-optimised and area-optimised. Due to the stochastic nature of the environment and agent, each evaluation takes the average over 8 independent runs. The loss value is then the upper 95% confidence interval in the average across runs. This is to give an indication of worst case average performance such that evaluations

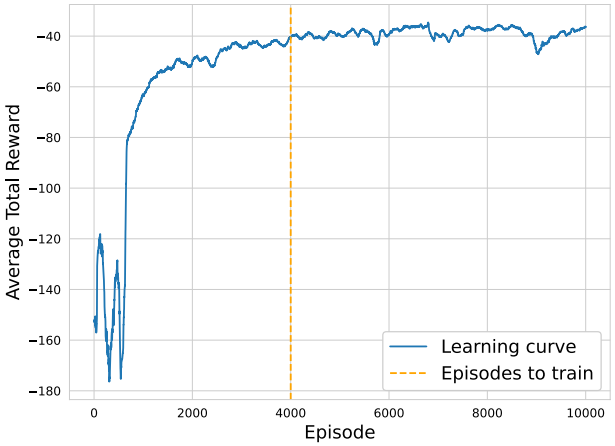


Fig. 2 Initial training with untuned hyperparameters to select number of training episodes. Uniformly filtered over 120 episodes for clarity

with low mean and high variance have a higher loss than those with lower variance. The maximum magnitudes of the x-, y-, and z-direction thrusters can also be treated as tunable hyperparameters. To minimise the dimensionality of the hyperparameter search space, we set $T^x_{max} = T^y_{max}$. Furthermore, the intermediate action magnitude in the z-direction is always $0.5 \cdot T^z_{max}$ during hyperparameter optimisation. The effect of changing this relative magnitude is discussed later.

Table 2 shows the selected hyperparameters based on the area-optimised and reward-optimised losses. Figures 3 and 4 show the results of the hyperparameter optimisation as parallel plots. Each line in the plot represents one evaluation in the optimisation and the magnitude on each axis shows the value for its respective hyperparameter. Considering first the NN structure, we see that in both cases the optimised networks take on an autoencoder structure (Hinton 1990) with a sparse layer of few hidden nodes located between two larger hidden layers. The parallel plots also clearly show this tendency towards such a structure, with many darker

Table 2 Selected Q-Learning hyperparameters when optimising for area under learning curve and final reward

Parameter	Area-optimised	Reward-optimised
$\tilde{N}_1$	212	150
$\tilde{N}_2$	80	65
$\tilde{N}_3$	218	165
α	3.96×10^{-5}	$2.08 \times 10^{-}5$
C	70	65
ϵ_i	0.367	0.269
N_ϵ	300	2700
γ	0.914	0.926
k	58	104

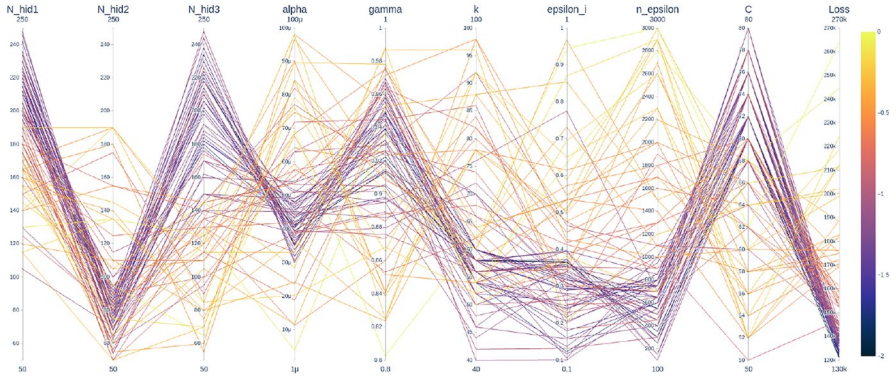


Fig. 3 Parallel plots showing hyperparameter optimisation evaluations for area. Colour indicates loss, normalised on a log scale for clarity

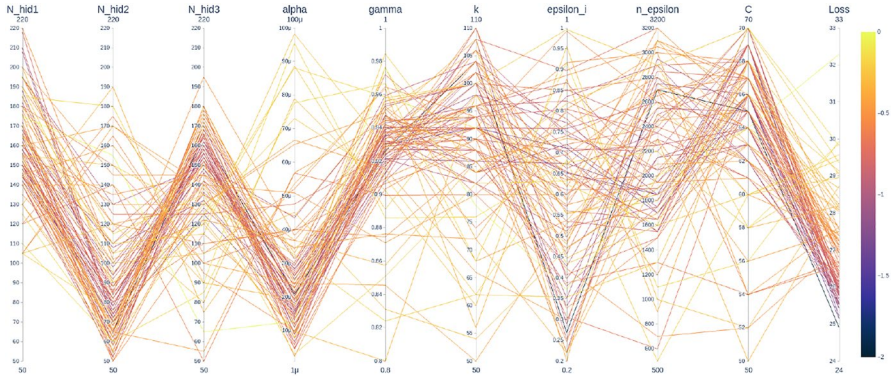


Fig. 4 Parallel plots showing hyperparameter optimisation for reward. Colour indicates loss, normalised on a log scale for clarity

lines in a 'V' shape for both optimisations. This was not biased by the initial limits selection which were over a wide range for each layer and gradually adjusted as the optimisation progressed, and so it is interesting to see this structure emerge purely from automated selection of the number of hidden nodes. The area-optimised network is also larger with every layer containing more hidden nodes than the corresponding reward-optimised layer.

When tuning learning rates, it is reasonable to assume that optimising for area yields a higher learning rate to converge more quickly on a solution. This is reflected in the optimised values for α which is nearly 2 times larger in the area-optimised case. Again there are more darker lines higher up this axis in the area-optimised parallel plot compared to that of the reward-optimised. Another key difference in the hyperparameters comes in those affecting exploration: the value of N_ϵ for area-optimised is 9 times smaller than that of reward optimised. This is also reasonable since the area-optimised agent seeks to exploit actions as quickly as possible. While

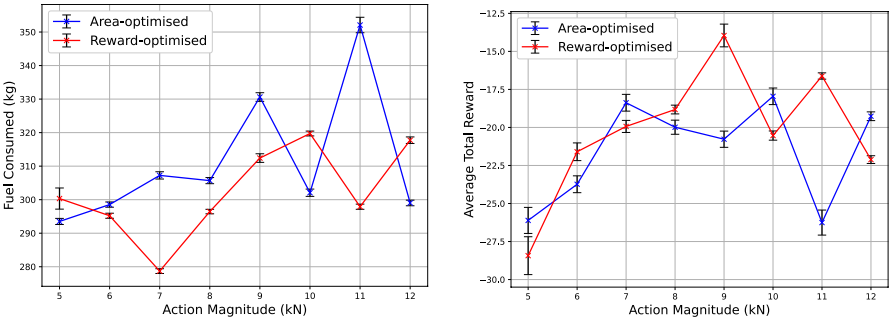
the initial exploration probability ϵ_i is slightly higher in the area-optimised case, its exploration probability will quickly decrease to less than the reward-optimised agent. Looking at the parallel plots, we can also see that the best performances in terms of reward had a wide range of values for ϵ_i, however the best for the area optimisation are clustered around lower values. This is also the case for N_ϵ, where the best values for the reward optimisation tend to occupy a higher range of values. The final notable difference is in the minibatch sizes: 58 and 104 for area-optimised and reward-optimised respectively. The reason for this is unclear but, as can be seen from the parallel plots, this behaviour was shown across all optimisation runs with final reward favouring larger minibatch sizes and area under learning curve showing a peak in performance at and near $k = 58$. In the remainder of this section we use both the area-optimised and reward-optimised hyperparameters to compare their results.

4.2 Action size selection

In addition to the agent's hyperparameters, there are several environmental parameters which can be tuned. For most of these we use default values as presented in Gaudet et al. (2020b). The only parameters which we tune here are the action magnitudes. As discussed previously, this was fixed in the z-direction as $T^z_{max} = 12$ kN. Initial tests showed that this gave suitable performance and did not need further tuning. On the other hand, the action magnitudes in the x- and y-direction were found to affect the performance both in terms of landing precision and fuel consumption. This motivates a methodical approach to their selection.

Given the rotational symmetry of the problem about the z-axis, we can simplify the selection of action magnitudes by letting $T^x_{max} = T^y_{max}$ such that we only need to find a single value. We explore the effect changing this magnitude has on performance by testing a trained agent over 500 episodes. This is done for values of action magnitude in the range (5 kN,12 kN) in steps of 1 kN. As was done for the hyperparameter optimisation, agents are trained for 4000 episodes. In the testing episodes we examine the reward received by the agent and the fuel consumption. It is expected that the optimal hyperparameters vary for every action magnitude, however due to time constraints and the time required to optimise the hyperparameters it was not possible to do this for the full range of action magnitudes. Despite this, we can still use the optimised values to obtain favourable performance across the range as shown here.

The results of this study are shown in Fig. 5. One important observation is that the trend in fuel consumption does not track the total reward; contrary to what might be expected given fuel consumption is part of the reward function. This is because of the landing bonus and velocity tracking terms in the reward function which can be more fuel intensive to achieve but produce larger rewards. Both fuel consumption and average total reward vary considerably across action magnitudes and occasionally have large peaks and troughs, for example when the magnitude is 11 kN for the area-optimised agent. Considering first the reward-optimised agent, its best performance in terms of reward is at 9 kN, but this also has the third highest fuel

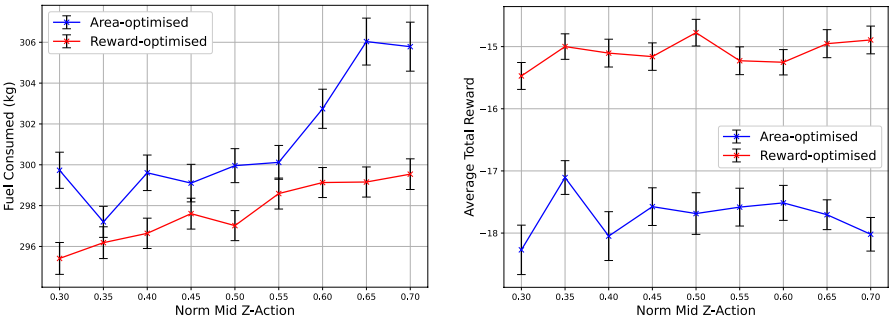


(a) Fuel consumption (b) Average total reward

Fig. 5 Change in performance with varying values of action magnitude for the powered descent problem. The respective goals are to minimise the fuel consumption and maximise the reward. Error-bars denote one standard error

consumption across its range. Furthermore, the reward shows higher variance at 9 kN relative to other values. We select the action magnitude for the reward optimised agent as 11 kN which gives the second highest reward and median fuel consumption. This differs significantly for the area-optimised agent for which 11 kN is notably the worst performing value. Peak reward for this agent is at 10 kN which does also show favourable fuel consumption. However, as before we consider the point which has lower variance in reward to be more beneficial and so the action magnitude for the area-optimised agent is selected as 12 kN.

Initially the midpoint thrust in the z-direction was fixed at half the maximum thrust, i.e. 6 kN. Here we also test the effect of varying this magnitude on the performance of a trained agent. Using the x and y action magnitudes specified previously, we vary the midpoint of the z-direction action between 0.3 and 0.7 of the maximum thrust and observe the changes in fuel consumption and average total reward. The results of this are shown in Fig. 6. Both metrics occupy a much narrower range of values compared to those in Fig. 5. In particular, the average total reward for each



(a) Fuel consumption (b) Average total reward

Fig. 6 Change in performance with varying mid-points of z-direction action for the powered descent problem. Error-bars denote one standard error

set of optimised hyperparameters remains nearly constant across the values of the midpoint thrust. As expected, the fuel consumption increases with the midpoint thrust, especially for the area-optimised agents. However, values below 0.5 again show very little improvement in the fuel consumption for both agents. As with the previous tests, these varying midpoint actions would likely have different associated hyperparameters. Furthermore, in this case all of these agents were trained using the same midpoint action of 0.5 which was only varied in testing. Despite these limitations, the results shown here suggest tuning this action has little effect on the resulting performance, and so this midpoint action was kept fixed at half the maximum magnitude.

4.3 Training and testing agents

With the hyperparameters and action sizes defined, here we show the results obtained from applying these agents to the environment. As with the hyperparameter optimisation, the stochastic nature of the environment and agents means it is necessary to run them multiple times to determine their range of performance. We do 50 training runs of 4000 episodes for each agent and observe their learning curves. The results of this are shown in Fig. 7—averaged across runs with standard deviation shown.

Both sets of hyperparameters show very similar trends in the learning curves which show a sharp increase in average reward over the first 300 episodes before gradually plateauing at their maximum average reward. As would be expected, the area-optimised curves reach their maximum more quickly than the reward-optimised. Considering the standard deviation, we see little variance in performance over most of the training period. In both cases the highest variance occurs in the

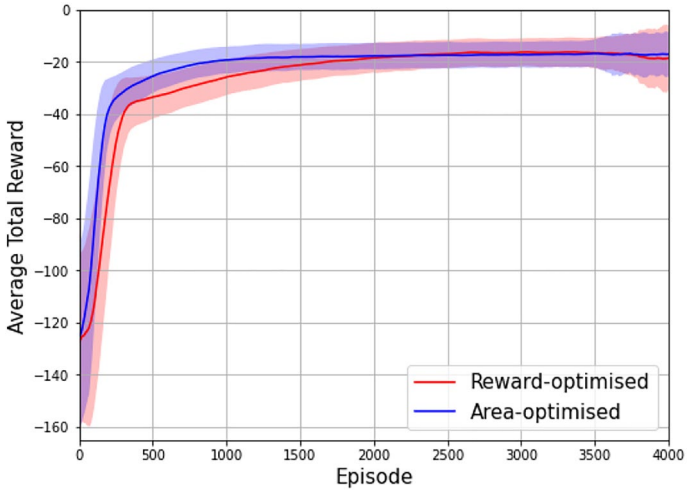


Fig. 7 Learning curves averaged over 50 runs for each agent. Shaded area indicates +/- one standard deviation. Uniformly filtered (average) over 120 episodes for clarity

initial episodes when more random actions are taken. This variance also increases slightly towards the end of training—notably in the reward-optimised curves. This is highlighted in Fig. 8 which shows the reward optimised run which performs poorly towards the end of training, causing the standard deviation of reward-optimised runs to increase. Up to the final 500 episodes, this curve tracks the mean very closely. In the final 300 episodes the average total reward decreases sharply—mirroring the rapid increase in early training. This behaviour can be due to the inherently unstable nature of Q-Learning (Van Hasselt et al. 2016). This instability is partly mitigated by the use of the target network, as discussed previously, and by careful tuning of hyperparameters. While we aimed to optimise the hyperparameters using a metric which encouraged robustness in the solution, achieving a more robust solution less susceptible to diverging performance would require more training runs per evaluation. However, most of the trained agents display favourable learning performance and can be successfully applied to the problem.

The results which follow use the optimal trained agents from each of the hyperparameter runs; i.e. the area-optimised agent with smallest area under the learning curve and the reward-optimised agent with highest average reward at the end of training. These individual learning curves along with the number of steps per episode are shown in Fig. 10. For comparison, a learning curve obtained from training an agent using PPO is shown in Fig. 9. Note that the y-scales are identical in both Figures, however the x-scale shows the difference in training times between Q-learning and PPO. The number of training episodes used to train the agent with Q-learning is an order of magnitude less than that of PPO. The policy trained using PPO does converge on a higher average reward than either of the Q-learning agents, which could be a result of the discretised action space causing a loss in optimality. This suggests a necessary trade-off between performance and learning time when choosing a RL algorithm to train an agent. While the average total reward is different

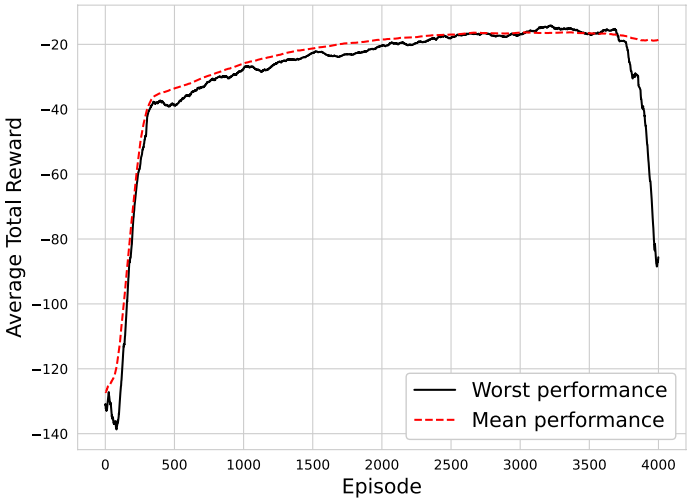


Fig. 8 Mean and worst performing learning curves for the reward-optimised agent

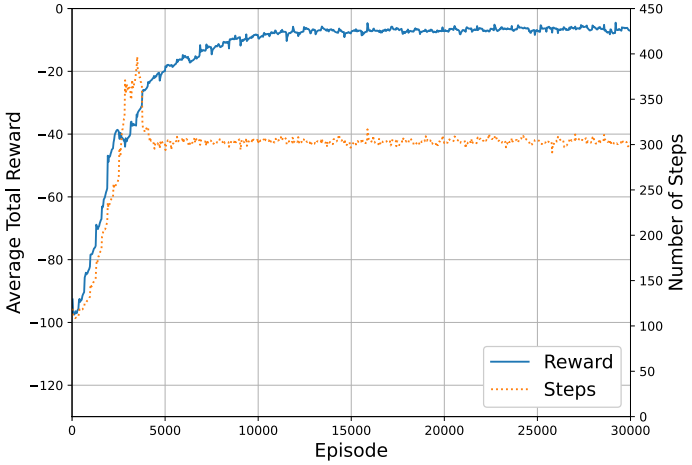


Fig. 9 Average reward and steps per episode over a training run of the PPO agent. Data from Gaudet et al. (2020b)

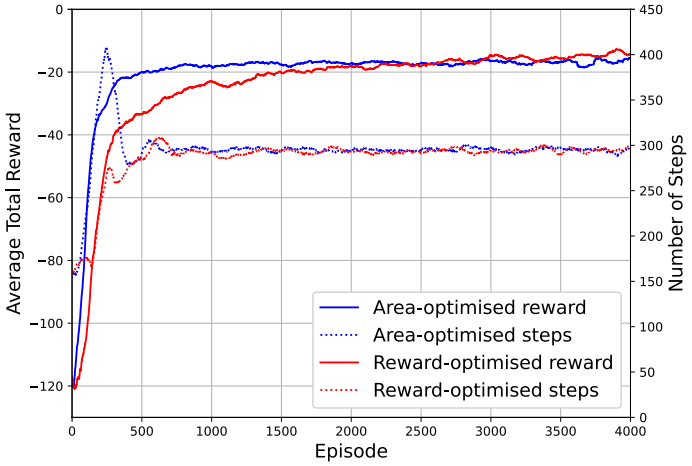


Fig. 10 Average reward and steps per episode over a training run of Q-learning for two different sets of hyperparameters. Uniformly filtered (average) over 120 episodes for clarity

at the end of training, the number of steps per episode of every trained agent is consistently close to 300. This is likely a result of the shaped reward function which effectively specifies the velocity at all states and therefore will tend to cause similar durations across episodes (Fig. 10).

Figures 11 and 12 show an example trajectory of the reward-optimised and area-optimised agents respectively. These figures show the position and velocity of a lander over the course of one episode starting from initial conditions close to the edge of its range experienced in training. In both cases the position and velocity

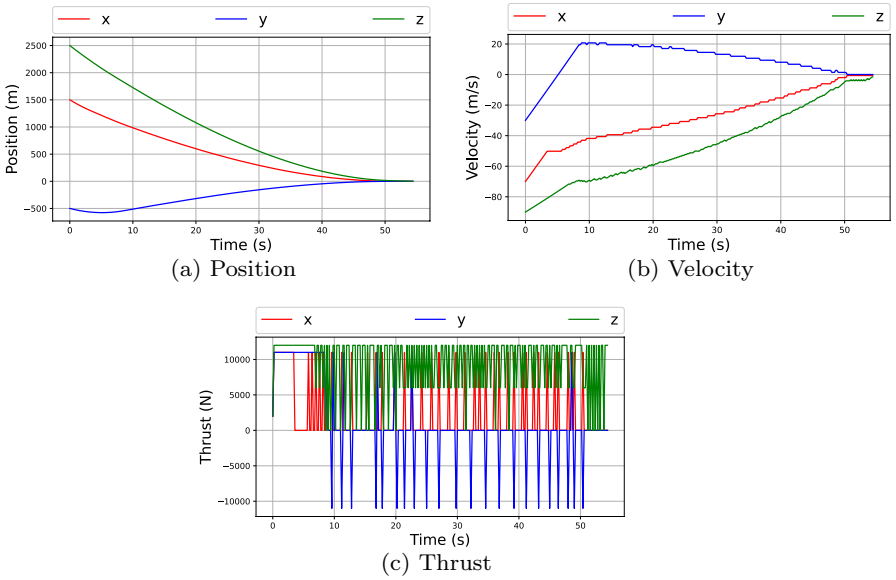


Fig. 11 Trajectory of the reward-optimised agent over a sample episode with commanded thrusts

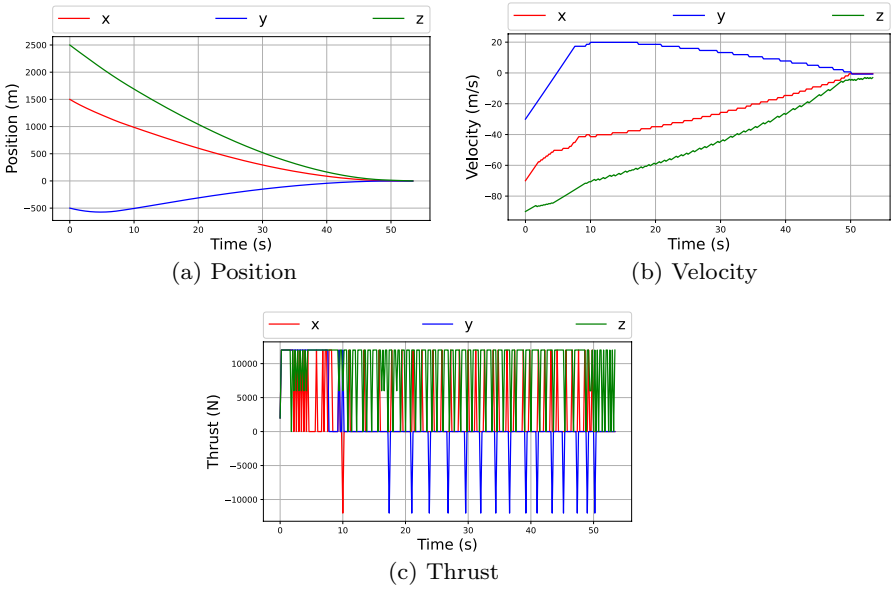


Fig. 12 Trajectory of the area-optimised agent over a sample episode with commanded thrusts

show an expected tendency towards zero with a very low velocity in the z-direction over the final seconds of the episode. The reward-optimised agent appears to achieve a softer landing with lower vertical velocity than that of the area-optimised agent.

As will be shown, this was common across testing episodes. The figures also show the commanded thrusts, i.e. the agent's actions over the course of the trajectory. It is worth noting that although the thrust oscillates between magnitudes very often, if we assume a minimum duration of 0.01 s (Kienitz and Bals 2005), given the action sampling time used here of $0.2s$, these control actions can provide a physical solution. For both agents the thrusts in the x- and y-directions show similar behaviour with a small difference in their magnitude as discussed previously. On the other hand, while the reward-optimised agent frequently selects the $T^z = 0.5 \cdot T^z_{max}$ action in the z-direction, the area-optimised agent resembles more a 'bang-bang' controller which switches between zero and its maximum amplitude. It is possible that this represents a local minimum solution which the area-optimised agent quickly converged on, but with worse performance than the policy found after more episodes by the reward-optimised agent.

Finally we test each agent's ability to handle the varying initial conditions by running many Monte Carlo simulations of an episode for a trained agent. The distributions for these are the same as for training and are simulated 5000 times to test over a broad range of initial conditions. Figure 13 and Table 3 show the distributions of

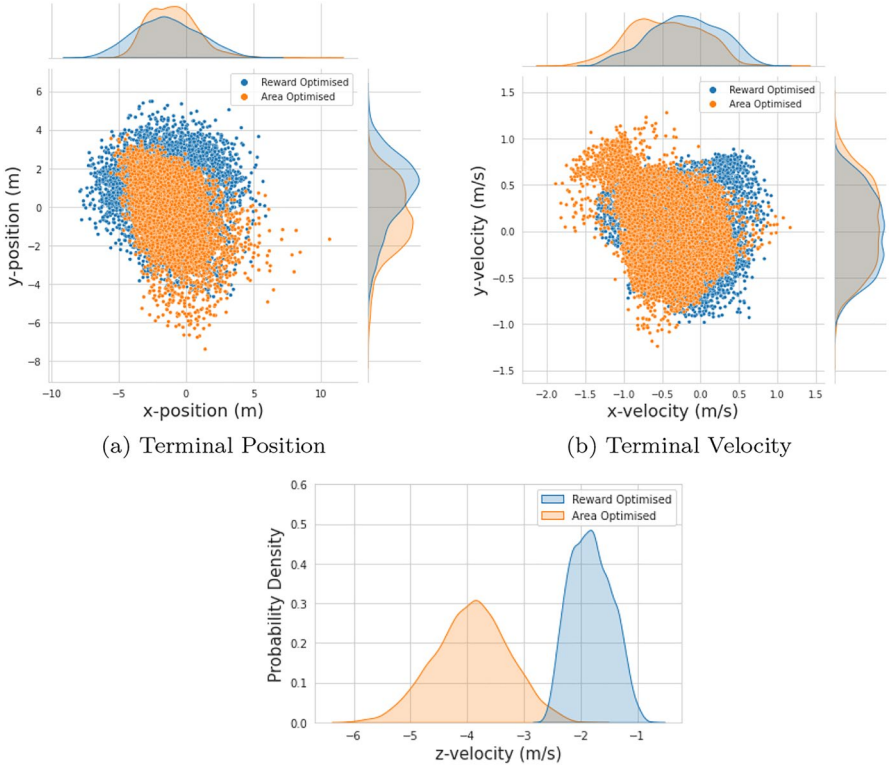


Fig. 13 Scatter and KDE plots showing distributions of terminal landing states for each agent

Table 3 Comparison of test results from both agents trained with Q-Learning

	Area				Reward			
	Mean	Min.	Max.	STD	Mean	Min.	Max.	STD
xy-position (m)	2.501	0.032	10.798	1.228	3.09	0.082	7.918	1.422
xy-velocity (m/s)	0.709	0.007	1.958	0.373	0.591	0.011	1.393	0.267
z-velocity (m/s)	− 3.927	− 6.029	− 1.865	0.659	− 1.827	− 2.647	− 0.714	0.361
Fuel (kg)	295.5	254.9	408.8	18.1	306.4	267.4	376.1	17.3

Statistics shown for terminal state over 5000 test episodes

terminal velocity and position for both the reward-optimised and area-optimised, as well as the fuel consumption. Both agents consistently achieve x- and y-velocities within the desired range, however the area-optimised agent has slightly higher variance in the x-velocity. The x- and y-positions show a greater spread than the respective velocities for both agents. While they mostly reach the desired landing region within a $5m$ radius, there are several points for both agents which do not satisfy this requirement. The most striking difference in performance between the agents is in the terminal z-velocity. The mean z-velocity of the reward-optimised agent is -1.827 m/s whereas for the area-optimised agent it is -3.927 m/s. This means the area-optimised agent is consistently unable to achieve the soft landing requirement, but the reward-optimised agent shows much better performance in achieving a soft pinpoint landing.

We compare the results from the Q-Learning agents to those obtained from the agent trained using PPO. The results from running the same 5000 test episodes for the PPO agent as for Q-Learning are shown in Table 4. From this, we see that the agent trained with PPO achieves a more precise landing with a maximum distance from the landing position of only 1.2 m, compared to 7.918 m for the reward-optimised agent and 10.798 for the area optimised. Furthermore, the velocities are generally lower for the PPO agent. These differences can be attributed to the coarse discretisation for the Q-learning agents which limits their ability for finer control. This is particularly important for achieving the pinpoint landing over the final few metres of descent, and so training a separate controller for this final phase could improve the overall performance. In addition, increasing the action space size with more discrete magnitudes could improve the agent's landing precision, but at the expense of increased problem complexity. Finally comparing the fuel consumption, the

Table 4 Test results for agent trained with PPO. Statistics shown for terminal state over 5000 test episodes

	PPO			
	Mean	Min.	Max.	STD
xy-position (m)	0.744	0.318	1.2	0.16
xy-velocity (m/s)	0.254	0.228	0.295	0.008
z-velocity (m/s)	− 0.719	− 0.913	− 0.107	0.12
Fuel (kg)	291.1	262.1	353.6	14.4

averages for each agent are very similar: 291.1 kg for PPO, 295.5 for area-optimised Q-Learning, and 306.4 for reward-optimised Q-Learning. The area-optimised agent has both the highest maximum fuel consumption at 408.8 kg and the lowest minimum with 254.9, however this is lilkely due to this agent rarely achieving a 'soft' landing and therefore requiring less fuel to decelerate. Although the reward-optimised agent consumes more fuel than the area-optimised, it clearly shows superior landing performance and achieves comparable fuel consumption to the PPO agent.

4.4 Training with raw state representation

Applying the same methodology as for the results above, we also train an agent using Q-Learning with a raw state representation, as given by Eq. 24. Again we tune hyperparameters for this problem formulation using the two area- and reward-based optimisation criteria. These gave the optimised hyperparameters as shown in Table 5. The action magnitudes used were the same as for the shaped state, with $T^x_{max} = T^y_{max} = 11$ kN for the reward-optimised agent and $T^x_{max} = T^y_{max} = 12$ kN for the area-optimised agent. These hyperparameters appear different to those of the shaped state problem (Table 2). In particular the NN structures do not show the clear autoencoder structure. Furthermore, the parameters ϵ_i and N_ϵ show the opposite behaviour of what is expected, with reward optimised favouring lower values for both compared to the area optimised.

Figure 14 shows the learning curves for the agents trained using each set of hyperparameters with the raw state representation. We can clearly see that it does not converge on as high a reward as the previous agents, only achieving an average cumulative reward of around −40 per episode. Nevertheless, both agents appear to 'learn' as their learning curves increase sharply throughout the early episodes. Testing these agents as before reveals their performance in terms of achieving a pinpoint soft landing is, however, very poor. Figure 15 shows the final positions and velocities for these agents over 5000 test episodes, which show it has a much wider spread of horizontal positions and velocities in the x- and y-directions than the agents using the shaped state. More importantly, the z-direction velocity is always very high with both agents always having terminal

Table 5 Selected Q-Learning hyperparameters when optimising for area under learning curve and final reward in the 'raw state' configuration

Parameter	Area-optimised	Reward-optimised
$\tilde{N}_1$	105	120
$\tilde{N}_2$	150	200
$\tilde{N}_3$	165	130
α	4.67×10^{-5}	3.89×10^{-5}
C	65	65
ϵ_i	0.441	0.316
N_ϵ	1850	900
γ	0.941	0.923
k	95	120

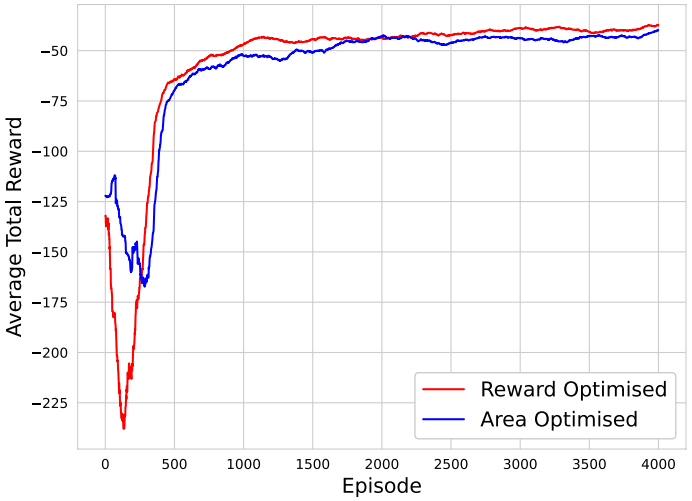


Fig. 14 Learning curves for agents trained with a raw state representation

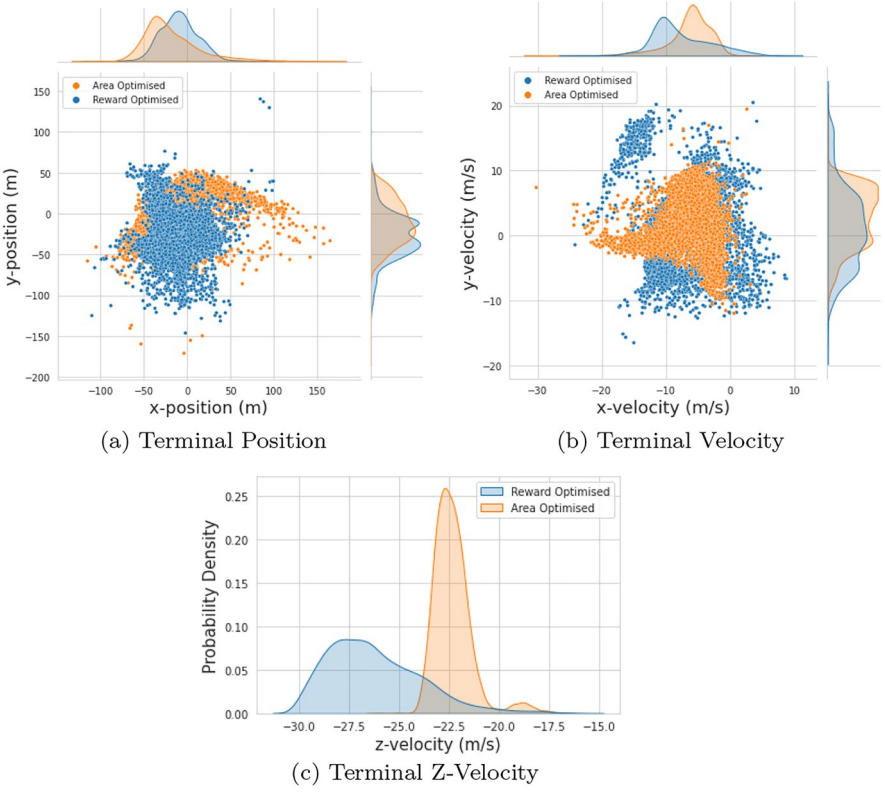


Fig. 15 Scatter and KDE plots showing distributions of terminal landing states for each agent trained with a raw state representation

z-velocities of magnitude greater than 15 m/s. This is likely due to the nature of the reward function, which rewards the agent for following a target velocity. While the agent has learned to follow this velocity over the start of the trajectory, to avoid receiving the negative rewards over the final descent, where the target velocity decreases more sharply, it accelerates towards the end instead of decelerating. This shows the importance of properly defining the RL problem, including the state representation, and suggests further research is needed into how to avoid such reward-hacking behaviour when using fixed state representations.

5 Conclusions

We have demonstrated that widely used RL techniques are applicable to the difficult control problem of powered descent. With respect to the goal of improving the learning time, it is clear that this has been achieved by simplifying the problem to a discrete action space and using a data-efficient algorithm. While this approach does not yield the same level of optimality as a continuous action agent, it is still capable of achieving near-optimal solutions which can cope with uncertainties. Using this approach required tuning of hyperparameters and careful selection of action space parameters. Both of these factors were shown to have a significant effect on the performance of an agent. In particular, hyperparameter tuning for different loss functions gave contrasting values for the hyperparameters with corresponding variations in performance of the trained agents. In this case, optimising for mean reward at the end of training produced a more successful agent than optimising for area under the learning curve.

One of the limitations of the methodology presented here is the separate tuning of hyperparameters and action space. Ideally these would either be optimised in parallel with the action space magnitude as a variable, or optimised over a range of action magnitudes to allow a trade-off between solutions which could also consider hardware limitations. Other parameters relating to the reward function were also kept at their default values for the purpose of this study. Optimising these could also give improvements in performance.

The idea of creating simpler problems to improve the learning time has uses in many other environments. This could also be developed to incorporate transfer learning (Taylor and Stone 2009), such that agents trained quickly on simple problems can be transferred to more complex environments using knowledge acquired from a simpler problem. Attempting to minimise training times eventually leads to the problem of "one-shot-learning"—having an agent learn to solve a problem online in a single episode. This will likely involve meta-learning procedures which can optimise various aspects of a learning agent beyond what we have considered here. Future work will look into exploiting environment models for faster training times in similar spacecraft control problems, with the long term goal of achieving online adaptive agents capable of one-shot-learning.

References

Acikmese B, Ploen SR (2007) Convex programming approach to powered descent guidance for mars landing. J Guid Control Dyn 30(5):1353–1366. https://doi.org/10.2514/1.27553

Acikmese B, Carson JM, Blackmore L (2013) Lossless convexification of nonconvex control bound and pointing constraints of the soft landing optimal control problem. IEEE Trans Control Syst Technol 21(6):2104–2113. https://doi.org/10.1109/TCST.2012.2237346

Barsce JC, Palombarini JA, Martinez EC: Towards autonomous reinforcement learning: Automatic setting of hyper-parameters using Bayesian optimization. In: 2017 43rd Latin American Computer Conference, CLEI 2017, vol 2017. Institute of Electrical and Electronics Engineers Inc, pp 1–9 (2017)

Barto AG, Sutton RS, Anderson CW (1983) Neuronlike adaptive elements that can solve difficult learning control problems. IEEE Trans Syst Cybernet SMC–13(5):834–846

Battin RH (1999) An introduction to the mathematics and methods of astrodynamics, Revised Edition. American Institute of Aeronautics and Astronautics. https://doi.org/10.2514/4.861543

Bellman R (1966) Dynamic programming. Science 153(3731):34–37

Bergstra J, Bengio Y (2012) Random search for hyper-parameter optimization. J Mach Learn Res 13(2):281–305

Bergstra J, Bardenet R, Bengio Y, Kégl B (2011) Algorithms for hyper-parameter optimization. Adv Neural Inf Process Syst 24:2546–2554

Bergstra J, Yamins D, Cox DD (2013) Making a science of model search: hyperparameter optimization in hundreds of dimensions for vision architectures. In: 30th international conference on machine learning, ICML 2013, vol 28, pp 115–123

Blackmore L, Açikmeşe B, Scharf DP (2010) Minimum-landing-error powered-descent guidance for mars landing using convex optimization. J Guid Control Dyn 33(4):1161–1171. https://doi.org/10.2514/1.47202

Braun R, Manning R, Braun R, Manning R (2006) Mars exploration entry, descent and landing challenges. In: 2006 IEEE aerospace conference. IEEE, Big Sky, MT, USA, pp 1–18. https://doi.org/10.1109/AERO.2006.1655790

Brockman G, Cheung V, Pettersson L, Schneider J, Schulman J, Tang J, Zaremba W (2016) OpenAI Gym. arXiv preprint arXiv:1606.01540

Deisenroth MP, Rasmussen CE (2011) PILCO: a model-based and data-efficient approach to policy search. In: Proceedings of the 28th international conference on machine learning. https://doi.org/10.1055/s-2002-35543

Duan Y, Chen X, Schulman J, Abbeel P (2016a) Benchmarking deep reinforcement learning for continuous control. arXiv 48:14. https://doi.org/10.1109/CVPR.2014.180

Duan Y, Schulman J, Chen X, Bartlett PL, Sutskever I, Abbeel P (2016b) RL^2: Fast Reinforcement learning via slow reinforcement learning. arXiv preprint arXiv:1611.02779

Dulac-Arnold G, Evans R, van Hasselt H, Sunehag P, Lillicrap T, Hunt J, Mann T, Weber T, Degris T, Coppin B (2015) Deep reinforcement learning in large discrete action spaces. arXiv preprint arXiv:1512.07679

Fazlyab AR, Fani Saberi F, Kabganian M (2016) Adaptive attitude controller for a satellite based on neural network in the presence of unknown external disturbances and actuator faults. Adv Space Res 57(1):367–377. https://doi.org/10.1016/j.asr.2015.10.026

Furfaro R, Linares R (2017) Waypoint-based generalized ZEM/ZEV feedback guidance for planetary landing via a reinforcement learning approach. In: 3rd international academy of astronautics conference on dynamics and control of space systems, DyCoSS, pp 401–416

Gaudet B, Furfaro R (2014) Adaptive pinpoint and fuel efficient mars landing using reinforcement learning. IEEE/CAA J Automatica Sinica 1(4):397–411. https://doi.org/10.1109/JAS.2014.7004667

Gaudet B, Furfaro R, Linares R (2020a) Reinforcement learning for angle-only intercept guidance of maneuvering targets. Aerospace Sci Technol. https://doi.org/10.1016/j.ast.2020.105746

Gaudet B, Linares R, Furfaro R (2020b) Deep reinforcement learning for six degree-of-freedom planetary landing. Adv Space Res 65(7):1723–1741. https://doi.org/10.1016/j.asr.2019.12.030

Guo Y, Hawkins M, Wie B (2013) Applications of generalized zero-effort-miss/zero-effort-velocity feedback guidance algorithm. J Guid Control Dyn 36(3):810–820

Henderson P, Islam R, Bachman P, Pineau J, Precup D, Meger D (2018) Deep reinforcement learning that matters. In: The thirty-second AAAI conference on artificial intelligence, pp 3207–3214

Hinton GE (1990) Connectionist learning procedures. In: Machine learning. Elsevier, vol 3, pp 555–610. https://doi.org/10.1016/b978-0-08-051055-2.50029-8

Hunt KJ, Sbarbaro D, Zbikowski R, Gawthrop PJ (1992) Neural networks for control systems-a survey. Automatica 28(6):1083–1112. https://doi.org/10.1016/0005-1098(92)90053-I

Hutter F, Hoos HH, Leyton-Brown K (2011) Sequential model-based optimization for general algorithm configuration. In: International conference on learning and intelligent optimization. Springer, pp 507–523

Ichikawa Y, Sawa T (1992) Neural network application for direct feedback controllers. IEEE Trans Neural Netw 3(2):224–231. https://doi.org/10.1109/72.125863

Jiang X, Li S, Furfaro R (2019) Integrated guidance for Mars entry and powered descent using reinforcement learning and pseudospectral method. Acta Astronautica 163:114–129. https://doi.org/10.1016/j.actaastro.2018.12.033

Kienitz KH, Bals J (2005) Pulse modulation for attitude control with thrusters subject to switching restrictions. Aerospace Sci Technol 9(7):635–640. https://doi.org/10.1016/j.ast.2005.06.006

Klumpp AR (1974) Apollo lunar descent guidance. Automatica 10(2):133–146. https://doi.org/10.1016/0005-1098(74)90019-3

Li L, Jamieson K, DeSalvo G, Rostamizadeh A, Talwalkar A (2017) Hyperband: a novel bandit-based approach to hyperparameter optimization. J Mach Learn Res 18(1):6765–6816

Lillicrap TP, Hunt JJ, Pritzel A, Heess N, Erez T, Tassa Y, Silver D, Wierstra D (2015) Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971

Mao Y, Szmuk M, Acikmese B (2016) Successive convexification of non-convex optimal control problems and its convergence properties. In: 2016 IEEE 55th conference on decision and control (CDC). IEEE, Las Vegas, NV, USA, pp 3636–3641. https://doi.org/10.1109/CDC.2016.7798816

Meditch J (1964) On the problem of optimal thrust programming for a lunar soft landing. IEEE Trans Autom Control 9(4):477–484

Mnih V, Kavukcuoglu K, Silver D, Rusu AA, Veness J, Bellemare MG, Graves A, Riedmiller M, Fidjeland AK, Ostrovski G, Petersen S, Beattie C, Sadik A, Antonoglou I, King H, Kumaran D, Wierstra D, Legg S, Hassabis D (2015) Human-level control through deep reinforcement learning. Nature. https://doi.org/10.1038/nature14236

Mnih V, Puigdomènech Badia A, Mirza M, Graves A, Harley T, Lillicrap TP, Silver D, Kavukcuoglu K (2016) Asynchronous methods for deep reinforcement learning. In: International conference on machine learning

Ng AY (2003) Shaping and policy search in reinforcement learning. University of California, Berkeley

Ng AY, Russell SJ et al (2000) Algorithms for inverse reinforcement learning. In: ICML, vol 1, p 2

Quadrelli MB, Wood LJ, Riedel JE, McHenry MC, Aung MM, Cangahuala LA, Volpe RA, Beauchamp PM, Cutts JA (2015) Guidance, navigation, and control technology assessment for future planetary science missions. J Guid Control Dyn 38(7):1165–1186. https://doi.org/10.2514/1.G000525

Riccardi A, Minisci E, Di Carlo M, Wilson C, Marchetti F (2018) Assessment of intelligent control techniques for space applications. Technical report, European Space Agency

Riedmiller M (2005) Neural fitted Q iteration—first experiences with a data efficient neural reinforcement learning learning method. In: 16th European conference on machine learning. Porto, Portugal. https://doi.org/10.1007/11564096_32

Saridis GN (1979) Toward the realization of intelligent controls. Proc IEEE 67(8):1115–1133. https://doi.org/10.1109/PROC.1979.11407

Scherffig L (2002) Reinforcement learning in motor control. Ph.D. thesis, University of Osnabruck

Schulman J, Levine S, Moritz P, Jordan M, Abbeel P (2015) Trust region policy optimization. In: 32nd international conference on machine learning, ICML 2015, vol 3, pp 1889–1897
Schulman J, Wolski F, Dhariwal P, Radford A, Klimov O (2017) Proximal Policy Optimization Algorithms. arXiv preprint arXiv:1707.06347. http://arxiv.org/abs/1707.06347
Schweighofer N, Doya K (2003) Meta-learning in reinforcement learning. Neural Netw 16(1):5–9. https://doi.org/10.1016/S0893-6080(02)00228-9
Silver D, Huang A, Maddison CJ, Guez A, Sifre L, Van Den Driessche G, Schrittwieser J, Antonoglou I, Panneershelvam V, Lanctot M, Dieleman S, Grewe D, Nham J, Kalchbrenner N, Sutskever I, Lillicrap T, Leach M, Kavukcuoglu K, Graepel T, Hassabis D (2016) Mastering the game of Go with deep neural networks and tree search. Nature. https://doi.org/10.1038/nature16961
Snoek J, Larochelle H, Adams RP (2012) Practical Bayesian optimization of machine learning algorithms. In: Proceedings of the 25th international conference on neural information processing systems-Volume 2, pp 2951–2959
Song Zy, Wang C, Theil S, Seelbinder D, Sagliano M, Liu Xf, Shao Zj (2020) Survey of autonomous guidance methods for powered planetary landing. Front Inf Technol Electron Eng 21(5):652–674 (2020). https://doi.org/10.1631/FITEE.1900458
Sutton RS (1996) Generalization in reinforcement learning: successful examples using sparse coarse coding. Adv Neural Inf Process Syst 8:1038–1044
Sutton RS, Barto AG (1998) Reinforcement learning: an introduction. MIT Press, Cambridge. https://doi.org/10.1109/MED.2013.6608833
Talebi HA, Khorasani K, Tafazoli S (2009) A recurrent neural-network-based sensor and actuator fault detection and isolation for nonlinear systems with application to the satellite's attitude control subsystem. IEEE Trans Neural Netw 20(1):45–60. https://doi.org/10.1109/TNN.2008.2004373
Taylor ME, Stone P (2009) Transfer learning for reinforcement learning domains: a survey. J Mach Learn Res 10:1633–1685
Van de Wiele T, Warde-Farley D, Mnih A, Mnih V (2020) Q-Learning in enormous action spaces via amortized approximate maximization. arXiv preprint arXiv:2001.08116
Van Hasselt H, Guez A, Silver D (2016) Deep reinforcement learning with double Q-learning. In: Proceedings of the 30th AAAI conference on artificial intelligence, pp 2094–2100
Wang JX, Kurth-Nelson Z, Tirumala D, Soyer H, Leibo JZ, Munos R, Blundell C, Kumaran D, Botvinick M (2017) Learning to reinforcement learn. arXiv preprint arXiv:1611.05763. https://doi.org/10.1039/c004615a
Watkins CJCH (1989) Learning from Delayed Rewards. Ph.D. thesis, King's College
Watkins CJCH, Dayan P (1992) Q-learning. Mach Learn 8:279–292
Wilson C, Marchetti F, Carlo MD, Riccardi A, Minisci E (2019) Intelligent control: a taxonomy. In: 2019 8th international conference on systems and control, ICSC 2019, pp 333–339. Institute of Electrical and Electronics Engineers Inc. https://doi.org/10.1109/ICSC47195.2019.8950603
Young SR, Rose DC, Karnowski TP, Lim SH, Patton RM (2015) Optimizing deep learning hyper-parameters through an evolutionary algorithm. In: Proceedings of MLHPC 2015: machine learning in high-performance computing environments—held in conjunction with SC 2015: the international conference for high performance computing, networking, storage and analysis. https://doi.org/10.1145/2834892.2834896

Authors and Affiliations

Callum Wilson[1] · **Annalisa Riccardi**[1]

✉ Callum Wilson
callum.j.wilson@strath.ac.uk

[1] Mechanical and Aerospace Engineering, University of Strathclyde, 75 Montrose Street, Glasgow G1 1XQ, UK