Optimising Worlds to Evaluate and Influence Reinforcement Learning Agents

Paper #105

ABSTRACT

Training reinforcement learning agents on a distribution of procedurally generated environments has become an increasingly common method for obtaining more generalisable agents. However, this makes evaluation challenging as the space of possible environment settings is large, and simply looking at the average performance is insufficient for understanding how well - or how poorly - the agents perform. To address this, we introduce a method for strategically evaluating and influencing the behaviour of reinforcement learning agents. Using deep generative modelling to encode the environment, we propose a World Agent which efficiently generates and optimises worlds (i.e. environment settings) relative to the performance of the agents. Through the use of our method on two distinct environments, we demonstrate the existence of worlds which minimise and maximise agent reward beyond the typically reported average reward. Additionally, we show how our method can also be used to modify the distribution of worlds that agents train on, influencing their emergent behaviour to be more desirable.

KEYWORDS

Reinforcement Learning; Agent Simulation; Evaluation; Training; Procedurally Generated Environments

1 INTRODUCTION

Deep reinforcement learning has achieved many successes in the last few years. Be it human-level performance in the Atari games [26], solving continuous control tasks [24], or interacting in multi-agent settings [23], it is easy to see the progress which has been made. However, in many of these examples, agents are typically trained and evaluated on the same environment. As a result, this raises the concern that agents are overfitting to these environments [35], and therefore minor changes can significantly impact their performance.

One approach for improving the generalisation of agents is to train them on a distribution of procedurally generated environments. In this setup, a new environment setting (which we call *world* throughout this paper) is generated every episode. Such worlds can have different spatial layouts, entity types, and objectives, with the intention of improving the diversity of the training distribution and therefore improving agent performance on other environments.

While training agents on procedurally generated environments has improved their ability to generalise [16, 18, 19, 28], it does make evaluating agents more challenging. This is because the space of possible environment settings (i.e. worlds) is huge, and as such it is unlikely an agent will see the same world twice through random sampling. Additionally, the diversity of worlds may not be uniformly distributed, resulting in many worlds which are similar and/or simplistic. As a training method, this can lead to agents learning undesirable behaviours, while as an evaluation method this can lead to overconfidence in the performance of agents.

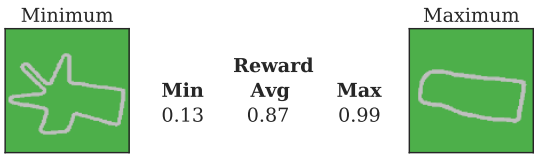


Figure 1: Examples of environment settings (i.e. worlds) which significantly deviate from the average reward. Left: A challenging race track due to its sharp corners, significantly reducing agent reward. Right: A short easy track due to its absence of sharp corners, leading to high agent reward.

For safety-critical applications of reinforcement learning, such as self-driving cars and healthcare, it is vital that we understand where trained agents succeed and where they fail. In these use-cases, training and evaluating agents on randomly sampled worlds is insufficient for understanding how well - or how poorly - they perform, and can also lead to undesirable learned behaviours emerging (such as high aggression). To demonstrate this, in Figure 1 we show the high reported average reward of an agent on our racing environment. Importantly, we also show the discovered worlds which cause the trained agent to perform worst (minimum reward) and best (maximum reward).

To address this, we propose a method for strategically evaluating and influencing reinforcement learning agents. Concretely, we introduce a generative agent – called the *World Agent* – into the training and evaluation process of reinforcement learning agents, giving it the ability to modify the distribution of worlds the agents see. It does this by using deep generative modelling to learn a latent representation of the world which it can then optimise in response to the behaviour and performance of the reinforcement learning agents. As an example, such optimisations could be to maximise the reward achieved by the agent, or minimise the occurrence of undesirable actions.

We demonstrate our method by applying it to two distinct use-cases: a single-agent racing environment and a multi-agent resource gathering environment. For both, we use our method to efficiently and consistently sample worlds which lead to minimal and maximal agent reward. Notably, we find challenging worlds where agents perform poorly, as well as highlight simple worlds where agents perform well. In addition, we include an analysis of two optimisation methods and discuss the trade-off between their performance and sample efficiency. To conclude, we use our method to update

Proc. of the 18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2019), N. Agmon, M. E. Taylor, E. Elkind, M. Veloso (eds.), May 2019, Montreal, Canada

https://doi.org/doi

the training distribution of the agents based on their performance, influencing their learned behaviour to be more desirable.

In summary, our three contributions are as follows:

(1) We introduce a World Agent which can generate worlds (i.e. environment settings) by training a deep generative model to parameterise a procedural environment generator.
(2) We demonstrate a sample efficient method for strategically evaluating agents using our trained World Agent, discovering worlds which lead to minimal and maximal agent reward.
(3) We show that our method can be used to modify the training distribution of agents, leading to different emergent behaviours which can be more desirable.

2 RELATED WORK

As our work draws on a wide body of literature, we broadly group our related work into two categories: (1) reinforcement learning and (2) generative modelling.

2.1 Reinforcement Learning

It has been recently shown that deep reinforcement learning agents can overfit in both single-agent [35] and multi-agent [22] environments, limiting their ability to generalise. One technique to help improve generalisation is to procedurally generate a distribution of worlds (i.e. environments, levels, maps, tasks) for agents to train on. For example, it has been used to prevent memorisation in Sokoban [28], generate challenging terrains and obstacles for continuous-control tasks [16], and form a hand-crafted curriculum in a number of video games [19].

Most recently, concurrent work to ours has started to investigate how single-agent reinforcement learning can be evaluated in procedurally generated environments. Specifically, worst-case analysis has been performed on a state-of-the-art maze navigation agent [2], and adversarial testing has been used to uncover rare catastrophic failures [1]. In comparison to both of these, we apply our method to an environment populated by multiple agents (Resource Harvest) and additionally investigate using our method to train agents rather than just evaluating them (which we refer to as influencing).

While training on a procedurally generated distribution has started to become more common in single-agent reinforcement learning research, applications to multi-agent settings have received less attention. One recent exception to this is Jadgerberg et al. (2018) [18] who obtained human-level performance in first-person multiplayer games by training agents on a diverse distribution of procedurally generated maps. Our work is an additional contribution to the multi-agent setting, and is an interesting use-case for further research in the area as the environment's settings can have a large influence on the emergent interactions between agents.

Building on the idea that the environment itself can influence multi-agent behaviour, Perolat et al. (2017) [27] demonstrated how – through handcrafting the environment – different social outcomes can emerge, such as conflict and inequality. We build on this work, creating a generative World Agent which can automatically generate such environments rather than requiring them to be handcrafted by a domain expert. Additionally, in our influencing experiments (Section 6.2) we use their proposed conflict and equality metrics to summarise social outcomes and influence agent behaviour towards lower conflict and higher equality.

Another related area to our work is Safe Reinforcement Learning [10]. In comparison to much of this area, we examine how the dynamics of the environment itself can be changed, rather than the learning process of the agents interacting with the environment. Our method can also be used to evaluate the safety of learned policies as our evaluation process makes no assumptions about how the agents were trained.

Outside of reinforcement learning, there is a large body of related literature on Procedural Content Generation (PCG) for game content and understanding how it interacts with the player. For example, weapon design [15], level generation [33], and mechanism generation [37]. Modern machine learning techniques are being increasingly applied to this domain [31], and our work is a further contribution towards this.

2.2 Generative Modelling

Our approach makes use of deep generative modelling to encode and optimise worlds. Such methods have been recently used to train agents inside their own learned world models [13] and generate video game levels from human-designed content [11].

Several works have investigated exploring the latent space of a generative model. For example, Bontrager et al. (2018) [4] propose a system for users to interactively evolve the latent vectors for a GAN towards target images (e.g. shoes). In related concurrent work to ours, latent variable evolution [5] was used to evolve Mario levels in the latent space of a GAN pre-trained on existing game levels [34]. We improve upon these works by considering agents which are able to *learn*, allowing us to evaluate their policies and influence their learning though variations in the world, both of which we investigate throughout this paper.

Others have also had success in using reinforcement learning itself to generate content. Ganin et al. (2018) [9] adversarially trained a reinforcement learning agent to generate images in the space of visual programs, while Zhang et al. (2018) [36] trained an agent to design mazes to be solved by other heuristic- and learning-based agents. In comparison, we focus on efficiently generating worlds for evaluating agents in both single- and multi-agent settings.

Our work also has parallels to mechanism design [7] which looks at how game rules (i.e. *mechanisms*) can be constructed such that desirable agent behaviours emerge, despite the agents' self-interests [29]. Recently, reinforcement learning and deep learning have been used to scale and automate mechanism design [8, 32], however limited work has been done from the perspective of agents interacting in an adaptive spatial environment.

3 PRELIMINARIES

In this section, we provide background information and notation which we use throughout the paper. Specifically, we introduce *deep generative modelling* (to generate worlds), *deep reinforcement learning* (to train agents in the world), and *black-box optimisation* (to optimise worlds).

3.1 Deep Generative Modelling

The aim of probabilistic generative modelling is to recover the true distribution of the input data $\mathbf{X}$. By sampling this distribution, new data points can be generated which resemble the observed data. We focus on the class of models which build approximate data distributions that are conditioned on the latent space $\mathbf{z}$ and a set of parameters $\boldsymbol{\theta}_g$, where we use the subscript 'g' to denote the parameters of the generative model.

In *deep* generative models, these parameters are the weights of deep neural networks. For our work, we use a Variational Auto-Encoder (VAE) [20] which learns an approximate distribution to the true data by maximising the evidence lower bound of the log likelihood of the data. The VAE is composed of an encoder-decoder architecture, where the encoder compresses data, $\mathbf{X}$, into the latent space $\mathbf{z}$, and a corresponding decoder is learnt to recover this data by transforming from the latent space back to the original data. This encoder-decoder structure is shown by the two distributions:

$$\mathbf{X} \sim p(\mathbf{X} \mid \boldsymbol{\theta}_g, \mathbf{z}), \qquad z \sim q(\mathbf{z} \mid \boldsymbol{\theta}_e), \tag{1}$$

where we introduce our encoder $q(\mathbf{z} \mid \boldsymbol{\theta}_e)$ (approximate inference network), with its corresponding parameters $\boldsymbol{\theta}_e$. Once the VAE has been learnt, the separate components can be used independently in order to either generate new data samples or to compress data into the latent space. This ability to use each component on their own is important for our work as will be shown in Section 4.2.

3.2 Deep Reinforcement Learning

As we consider both single- and multi-agent environments, we provide general multi-agent RL notation for N agents. In the single-agent setting, we set $N = 1$.

A stochastic game for N agents is defined by a set of states $\mathcal{S}$, an observation function $O : \mathcal{S} \times \{1, .., N\} \rightarrow \mathbb{R}^d$ specifying each agent i's d-dimensional private observation, and a set of actions for each agent $\mathcal{A}^i$. Agents choose their actions by sampling from a stochastic policy $\pi^i : O^i \times \mathcal{A}^i \rightarrow [0, 1]$ which leads to a state based on the state transition function $T : \mathcal{S} \times \mathcal{A}^1 \times ... \times \mathcal{A}^N \rightarrow \Delta(\mathcal{S})$ (where $\Delta(\mathcal{S})$ denotes the set of discrete probability distributions over $\mathcal{S}$). Each agent i receives reward defined as $r^i : \mathcal{S} \times \mathcal{A}^1 \times ... \times \mathcal{A}^N \rightarrow \mathbb{R}$ and tries to maximise its own total expected future reward $R^i = \sum_{t=0}^{\infty} \gamma^t r_t^i$, where $\gamma \in [0, 1)$ is the temporal discount factor and r_t^i is the reward received by agent i at time t.

The goal of reinforcement learning is to find the optimal policy π^* which achieves the maximum expected return from all states. In *deep* reinforcement learning, this policy is approximated with a deep neural network [26].

3.3 Black-Box Optimisation

If we want to optimise an objective function, but are only able to query the value $f(x)$ for a point $x \in \mathcal{X}$, then the problem setup is known as black-box optimisation. The key challenge is that $f(x)$ is not available in a simple closed form, leading to an optimisation task:

$$\mathbf{x}^* = \underset{\mathbf{x} \in \mathcal{X}}{\operatorname{argmax}}\, f(x), \tag{2}$$

where gradient information about the black-box objective function $f(x)$ is unavailable.

To solve such tasks, *Evolution Strategies* (ES) [25] approach the problem by evaluating the fitness of a batch of solutions, after which the best solutions are kept while the others are discarded. Survivors then procreate (by slightly mutating all of their genes) in order to produce the next generation of solutions. In this work, we use Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [14] which adaptively changes the size of the search space each generation. Another common approach is *Bayesian Optimisation* (BO) [30] which defines a prior distribution over the objective function (e.g. a Gaussian process) and selects new samples according to an acquisition function. The acquisition function defines a trade-off between exploration and exploitation when querying the objective function. We perform a comparison between CMA-ES and BO in Section 6.1.3.

4 METHODS

In this section, we introduce our method for strategically training and evaluating agents.

4.1 Overview

We propose the introduction of a generative *World Agent* into the training and evaluation process of reinforcement learning agents. By encoding the environment using a deep generative model and then searching in the model's latent space, our World Agent is able to efficiently adapt the distribution of worlds based on the performance of the reinforcement learning agents.

To help explain our method, we separate it into three phases which are visualised in Figure 2.

(1) **Generate Worlds:** Sample worlds using our pre-trained deep generative model.
(2) **Train Agents:** Train reinforcement learning agent(s) on the sampled set of generated worlds.
(3) **Optimise Worlds:** Iteratively generate and optimise worlds to maximise a given agent-based metric.

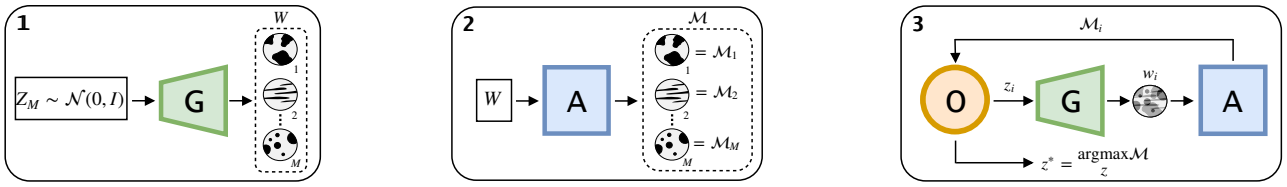
In this work, we consider both one iteration (for evaluating agents, see Section 6.1) and two iterations (for influencing agents via retraining, see Section 6.2) through these phases.

4.2 Generating Worlds

Rather than individually optimising every aspect of a world, for example at an individual pixel level, we use a deep generative model to compress the complex distribution over the world space $\mathbf{W}$ into a tractable distribution over the latent space $\mathbf{z}$. This allows us to efficiently optimise the environment by searching within this lower dimensionality latent space (Section 4.4). In Equation 3 we show how a world $\mathbf{w}_i$ can be sampled from the latent space by passing a sample $\mathbf{z}_i$ to the generator $\mathbf{G}$, where $\boldsymbol{\theta}_g$ are the generator's weights:

$$\mathbf{w}_i = \mathbf{G}(\mathbf{z}_i; \boldsymbol{\theta}_g), \qquad \mathbf{z}_i \sim q(\mathbf{z} \mid \boldsymbol{\theta}_e) \tag{3}$$

To learn $\boldsymbol{\theta}_g$ and $\boldsymbol{\theta}_e$, we train our VAE on a dataset of worlds created by a handcrafted procedural generator. As such procedural generators are typically rule-based or involve few parameters, they are challenging to optimise. To address this problem, we use a VAE which compresses the complex parameter set of the procedural generator into a tractable distribution over the latent space, making it possible to optimise worlds efficiently.



(a) Generate worlds via sampling. **(b) Train agents on sampled worlds.** **(c) Optimise worlds given trained agents.**

Figure 2: The three phases of our method. (a) Generate Worlds: sample latent vectors (z) from the latent space as input to the generator [G] to produce a set of worlds ($\mathbf{w}_1, ..., \mathbf{w}_M$) from the world distribution, (b) Train Agents: use sampled set of worlds (W) to train reinforcement learning agents [A], (c) Optimise Worlds: using our optimiser [O], iteratively sample the latent space to find z^*, where z^* is an optimal point in the latent space that maximises the World Agent's objective. Notation: $\mathbf{z}_i$ corresponds to the latent vector of world $\mathbf{w}_i$; the World Agent's evaluation of $\mathbf{w}_i$ is given by the metric $\mathcal{M}_i$.

After training, we use the learned decoder to form our generator [G]. To ensure that all generated worlds are valid for our environments, we include an additional processing step (described in Section 5.2). In Figure 3 below, we visualise an example of this pipeline for training the VAE and forming the the generator.

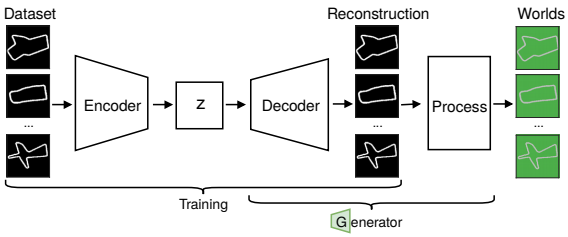


Figure 3: Setup of our VAE. (1) Training: Use procedurally generated dataset to train encoder and decoder, minimising error between input and reconstruction. (2) Generator: Use the trained decoder, and additional processing step, to form our World Agent's generator.

4.3 Training Agents

The next phase of our method is the training of the reinforcement learning agents[1] which interact in and with the worlds produced by our World Agent. Importantly, we assume that this is the only phase where the reinforcement learning agents learn.

Each agent tries to learn a policy which maximises its own expected return across the provided distribution of worlds. Importantly, agent reward functions do not have to align with our World Agent's objective.

For a given world $\mathbf{w}_i$ and its latent representation $\mathbf{z}_i$, the behaviour of the agents is summarised in their corresponding set of trajectories $\mathcal{T}_i$. These can then be quantified into various metrics $\mathcal{M}$ which we introduce in the next section.

4.4 Optimising Worlds

To search the space of worlds, we use an optimiser to sample from the latent space of the generator. Samples are selected with the goal of maximising the World Agent's objective function (i.e. metric $\mathcal{M}$), where the optimisation task:

$$\mathbf{z}^* = \underset{\mathbf{z}}{\text{argmax}}\ \mathcal{M}\left(\mathbf{G}(\mathbf{z}; \theta_g), \mathcal{T}(\mathbf{z})\right), \tag{4}$$

is over the latent space. This objective function depends on the behaviour of the agents (the trajectories $\mathcal{T}$) and the generated worlds $\{\mathbf{G}(\mathbf{z}_i; \theta_\mathbf{g})\}_{i=1}^{M}$, both of which are functions of the latent space.

For this optimisation task, there exists a trade-off between sample efficiency and performance. If sampling is expensive, then we would like to minimise the number of samples required. This is important if it takes a significant amount of time or resources to run an episode. On the other hand, if sampling is fast and cheap, then we would prefer a method which finds more optimal points even if it took longer. To this end, we compare two different types of optimisation methods described in Section 6.1.3 - an Evolution Strategy (ES) and Bayesian Optimisation (BO).

Metrics. The main metric we consider and optimise for using our World Agent is the reward the reinforcement learning agents receive. For our single-agent environment, this is simply the agent's total reward received in an episode, while in our multi-agent environment this is the sum total of rewards received by all agents (i.e. total group reward). We also consider several domain-specific metrics for both environments, however we defer their explanations until after we have introduced the environments (Section 6.2).

5 EXPERIMENTAL SETUP

In this section, we provide details on our environments, our World Agent which generates and optimises worlds for these environments, and the reinforcement learning agents which are then trained and evaluated in these generated worlds. In all of our experiments, we consider worlds (i.e. environment settings) to be the spatial layout of the environment, and therefore optimising worlds refers to the World Agent modifying the environment's spatial layout.

5.1 Environments

5.1.1 Particle Racing. Our first environment, shown in Figure 4, is a single-agent particle racing game based on the OpenAI Gym Car Racing environment [6]. The objective for the agent is to complete one loop of the track as quickly as possible, receiving -0.2 reward

[1]While we use reinforcement learning in this work, our approach also works for rule-based and pre-trained agents, as well as any other agent-based training method.

per step and a positive reward proportional to their speed along the track. In addition, if the agent leaves the track (i.e. crashes), it receives −300 reward and the episode is terminated. The episode also terminates after 300 steps if no crash occurs.

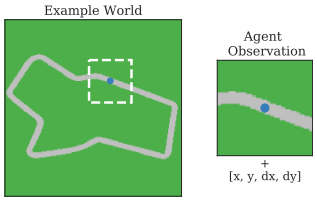


Figure 4: Example world (left) and agent observation (right) for the Particle Racing environment. The blue circle is the agent, grey is the track which the agent can move on, and green is the grass upon which the agent crashes. The agent's observation window is represented by the white dashed line.

The agent observes its surrounding area, as well as its current position and directional speed. At each step, the agent can apply a unit of force in one of four directions, increasing its velocity in that direction which is then gradually decreased over time due to friction from the track.

For this environment, our world agent can optimise the layout of the track, ranging it from a trivial oval to a challenging star with many sharp corners.

5.1.2 Resource Harvest. Our second environment, shown in Figure 5, is a multi-agent resource gathering game which has been recently explored in the multi-agent reinforcement learning literature [17, 23, 27]. Inspired by laboratory experiments from behavioural game theory, the harvest game is a sequential social dilemma for studying the emergent behaviour of groups in a spatial and temporal setting.

The environment itself consists of four self-interested agents who individually receive +1 reward for harvesting a resource (performed by moving onto it), and are therefore motivated to harvest the resources as fast as possible before the other agents are able to do so. Notably, resources recover based on the amount of nearby resources, and therefore leaving several resources untouched leads to faster recovery and more to collect in the long run.

Each agent has an orientation and observes the area it is facing. At each step, agents can stand still, move either forwards, backwards, left, or right, as well as rotate left or right.

For this environment, our World Agent can optimise the location of resources and walls, allowing for the isolation of agents and the privatisation of resources. Importantly, the agent spawn locations are fixed to the four corners.

5.2 World Agent

5.2.1 Generator. To encode the space of worlds, we train a Variational Auto-Encoder (VAE) on a procedurally generated dataset of worlds, using the learned decoder as the World Agent's generator. Both the encoder and decoder of the VAE consist of two fully connected hidden layers, whereby the encoder has a 1024–512 structure and the decoder is the transpose. The dimensionality of the latent space $\mathbf{z}$ is 10.

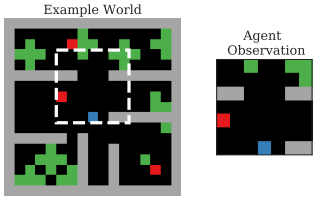


Figure 5: Example world (left) and agent observation (right) for the Resource Harvest environment. The red blocks are the agents (with the blue block representing the observing agent), the green blocks are resources (i.e. apples) which can be collected, and the grey blocks are walls. The agent's observation window is represented by the white dashed line.

For the Particle Racing environment, we use the procedural generator from the Car Racing environment [6] to generate a dataset of 10,000 tracks which we compress to 64×64 for training our generator. To ensure generated tracks are valid, we then map each reconstructed track to its most similar procedurally generated track.

As a procedural generator does not exist for the Resource Harvest environment, we constructed our own and generated a dataset of 10,000 worlds for training our generator. To ensure consistency across generated worlds, we process the reconstruction such that the number of resources is the same across all worlds.

Our handcrafted procedural generator works as follows: (1) create 0-4 regions of random sizes grown from each corner of the grid world, (2) merge overlapping regions, (3) add a bordering wall to each region with an optional entrance, (4) add clusters of resources to each room.

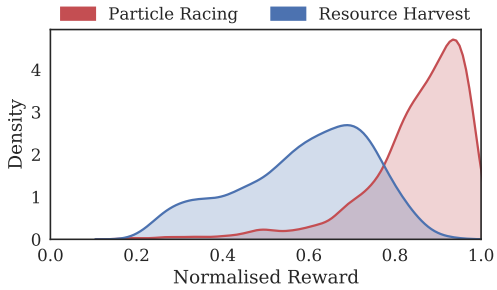
5.2.2 Optimiser. We use and compare two different optimisation algorithms for finding worlds. The first optimiser, and main one we use throughout, is Covariance Matrix Adaptation Evolution Strategy (CMA-ES). For this, we use the implementation of Ha (2017) [12] with a population size of 21. The second optimiser we investigate is Bayesian Optimisation (BO) which we implement using GPyOpt [3] and compare to CMA-ES in Section 6.1.3. For both optimisers, we evaluate each world 8 times and take the average of the returned metrics.

5.3 Reinforcement Learning Agents

We use two different deep reinforcement learning algorithms for the agents depending on whether the environment is single- or multi-agent. For the single-agent Particle Racing environment, we use an existing PyTorch implementation of the ACKTR algorithm [21] with its default settings, training for 10,000 episodes. For the multi-agent Resource Harvest environment, we use independent learning with each agent separately learning using its own Deep Q-Network [26]. We use the same settings as existing related literature on this environment [27] and train for 5,000 episodes.

6 EXPERIMENTS

In this section, we present the key results from our experiments. First, that we can efficiently find worlds which lead to significant differences in the performance of agents (Section 6.1). Next, that



(a) Distribution of rewards from randomly sampled worlds.

(b) Optimised worlds which minimise and maximise agent reward.

Figure 6: Results from evaluating agents on both environments. (a) Standard method where randomly sampled worlds are used to assess an agent's performance. Particle Racing yields a consistently high reward, while Resource Harvest has more variance but is still typically high. (b) Our strategic evaluation which efficiently discovers worlds where the performance of agents is minimised and maximised. Notably, our method is able to find worlds where the performance of the agents significantly deviates from the average.

this information can be used to modify the training distribution of agents, influencing their emergent behaviour (Section 6.2).

Note, we normalise all reported metrics to be between 0 and 1 to improve readability.

6.1 Evaluating Agents

To begin our experiments, we evaluate agents trained on a distribution of procedurally generated environments. First, we analyse them on randomly sampled worlds, similar to how they were trained. Next, we use our method to demonstrate the possible variances in agent performance on procedurally generated environments. To conclude, we then show the sample efficiency of our method.

6.1.1 Random Evaluation. To demonstrate the potential issues with evaluating agents via random sampling, we first evaluate agents on 1,000 randomly sampled worlds for both environments, visualising the resulting reward distribution in Figure 6a.

As can be seen, for Particle Racing a vast majority of the sampled worlds return a high reward, giving an overall average reward of 0.87. From this, we would typically conclude that the agent performs well and also generalises well to other tracks as we trained it on a diverse range of tracks. For Resource Harvest, the distribution of rewards has a higher variance due to the higher stochasticity in the environment, however most of its mass is still between 0.3 and 0.8 with an average of 0.61.

6.1.2 Strategic Evaluation. Next, we evaluate the agents using our method to *efficiently* find worlds where they perform worst (minimum reward) and perform best (maximum reward), yielding surprising results compared to our average analysis.

For Particle Racing (Figure 6b, top), our method finds a rare world which consistently causes the agent to crash, resulting in a minimum reward of 0.13 (an 85% reduction in reward from the reported average of 0.87). Notably, the environment has a high number of sharp and unexpected corners. In contrast, the world the agent performs best on – obtaining a reward of 0.99 – has no surprising corners, and is instead a simple rectangle-like shape. As a result, the agent never crashes.

For Resource Harvest (Figure 6b, bottom), our method finds a spatial arrangement of resources and walls such that the reward of the agents is heavily diminished - from the average of 0.61 down to a minimum of 0.05 (a 91.2% reduction). This was achieved by spreading out the resources so that they do not gain the recovery bonus from nearby resources. Conversely, reward is maximised (0.92, up from the average of 0.61) by removing walls and grouping resources so that they all benefit from the recovery bonus and therefore recover as quickly as possible.

In summary, we find that our method is able to evaluate where agents perform well and where their performance is significantly reduced. Next, we will demonstrate that this process is significantly more sample efficient than random sampling.

6.1.3 Analysis of Optimisation Methods. A key component of our World Agent is the optimisation process by which worlds are optimised based on the performance of the agents. To analyse the effectiveness of our optimisers, Figure 7 presents the distribution of rewards from worlds sampled by each optimiser with the World Agent's objective set to finding worlds which *minimise* the agent's reward.

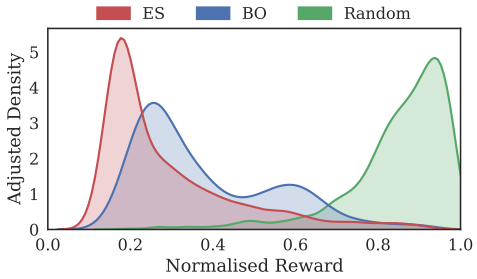
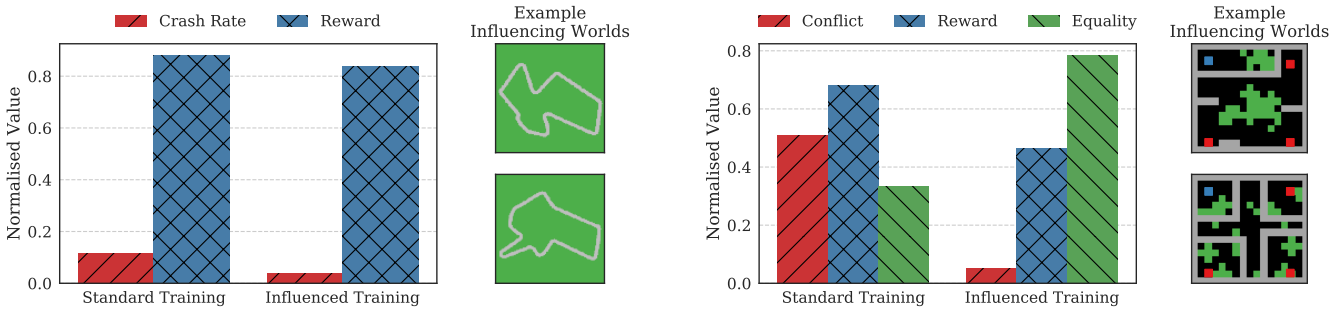


Figure 7: Reward distribution of each optimiser (ES & BO) optimising for *minimal* reward on the Particle Racing environment, and the Random distribution from Figure 6a.

As can be seen, both the evolution strategy (ES) and Bayesian optimisation (BO) are able to find a large number of worlds which



(a) Particle Racing. **(b) Resource Harvest.**

Figure 8: Comparison between training methods for (a) Particle Racing and (b) Resource Harvest. Default Training is where agents are trained on randomly sampled worlds. Influenced Training is where agents are trained on our modified training distribution. We also include two example worlds which are included in the new training distribution. Additional Metrics: *Crash Rate* is the probability of an agent crashing in an episode, *Conflict* is the average number of steps in an episode where agents are tagged out, and *Equality* is the distribution of rewards across agents calculated as 1-Gini Coefficient.

lead to significantly lower reward than average. Interestingly, we see that ES performs the best (i.e. more mass on lower rewards), however of note is that BO is more sample efficient, achieving a high reward on average but requiring 5× fewer samples during the optimisation process. This result indicates that there is a useful trade-off between performance and sample-efficiency which can be made based on the cost of sampling.

6.2 Influencing Agents

In our next set of experiments, we use our method to adjust the training distribution of agents by sampling worlds which maximise a given agent-related property. Concretely, we create a new training distribution consisting only of sampled worlds which have the property we desire (specific to each environment), and then train new agents on this distribution. These trained agents are then evaluated on the same random distribution of worlds as the previous experiment, demonstrating that learned behaviours transfer. We refer to this process as *influencing* agents as our method influences the learned emergent behaviour of the agents.

As each environment is different with their own unique metrics and results, we investigate them separately.

6.2.1 Particle Racing. In this environment, we observe that there exists many possible worlds which causes agents to crash. For safety-critical applications like self-driving cars, this is undesirable. To this end, we sample worlds where agents are likely to crash (i.e. high crash rate, > 0.5) to form the new training distribution. As shown in Figure 8a (right), these worlds tend to be challenging with at least one sharp corner.

In Figure 8a (left) we present the results of our influenced training regime. Notably, crash rate is significantly reduced, dropping from 0.11 to 0.04. This comes at the cost of a 5% reduction in reward due to the agent driving slower and therefore taking longer to complete the track. In the context of safety, our re-trained agent's behaviour is more desirable as it crashes less often.

6.2.2 Resource Harvest. For this environment, we consider the situation where one agent in a multi-agent system has more power than the others. Specifically, we allow this agent to perform the `TAG` action which fires a 3-block wide beam forward from the agent, removing any other agent hit from the episode for 25 steps. To measure the social consequences of this, we use two metrics introduced in Perolat et al. (2017) [27]:

(1) Conflict: The average number of steps in an episode where agents are tagged out.
(2) Equality: The distribution of rewards across agents, defined using the Gini coefficient as follows:

$$\text{Equality} = 1 - \frac{\sum_{i=1}^{N}\sum_{j=1}^{N}|R^i - R^j|}{2N\sum_{i=1}^{N}R^i}$$

Typically in this situation, the more powerful agent would tag other agents as much as possible (high conflict), reducing competition and therefore privatising the resources (high inequality). In many scenarios, this can be an undesirable behaviour which we would like to minimise. For example, if a powerful AI is created, we would like it to fairly share resources with human users rather than exploit them to maximise its own reward.

To counter this undesirable behaviour, we sample worlds which minimise conflict to form the new training distribution. Specifically, we sample worlds where no agents are tagged out in an entire episode, and then train fresh agents on this new distribution. As shown in Figure 8b (right), these no-conflict worlds are typically ones where the tagging agent (shown as blue in the top left of the world) is isolated from the other agents through the use of walls.

We present the impact of this training method in Figure 8b (left). As can be seen, our influenced training regime results in significantly lower conflict (from 0.51 to 0.05) and higher equality (from 0.33 to 0.78). This arises as the tagging agent never learns to associate its aggressive `TAG` action with increased reward, and therefore reduces the probability of its use. This comes at the cost of reduced group reward as the resources are less likely to be privatised.

7 CONCLUSIONS AND FUTURE WORK

In this work, we introduced a method for strategically evaluating and influencing reinforcement learning agents. By encoding the

environment using a deep generative model and then optimising in the latent space based on agent performance, we showed that our method can efficiently and consistently discover worlds which minimise and maximise agent performance. We further demonstrated that we can influence the emergent behaviour of agents by adapting the training distribution of worlds, producing agents whose behaviour is more desirable.

For safety-critical applications of reinforcement learning and multi-agent systems, our method provides a way of analysing and understanding where trained agents perform well and where they fail. For example, it can be used to efficiently discover simulated environment settings which cause a self-driving car to crash, as well as train an agent to interact more cooperatively in a multi-agent setting.

There are a number of natural extensions to explore in future work. First, we will further investigate how our method can be used in the training of agents, such as through additional iterations of our three-phase method or in an online manner where agents are continuously evaluated as they train. Next, for our generative World Agent we will look into other generative models (such as GANs), using reinforcement learning [9], as well as additional optimisation techniques that perform well while maintaining sample efficiency.

REFERENCES

[1] Anonymous. 2019. Rigorous Agent Evaluation: An Adversarial Approach to Uncover Catastrophic Failures. In *Submitted to International Conference on Learning Representations*. https://openreview.net/forum?id=B1xhQhRcK7 under review.

[2] Anonymous. 2019. Uncovering Surprising Behaviors in Reinforcement Learning via Worst-case Analysis. In *Submitted to International Conference on Learning Representations*. https://openreview.net/forum?id=SkgZNnR5tX under review.

[3] The GPyOpt authors. 2016. GPyOpt: A Bayesian Optimization framework in python. http://github.com/SheffieldML/GPyOpt. (2016).

[4] Philip Bontrager, Wending Lin, Julian Togelius, and Sebastian Risi. 2018. Deep Interactive Evolution. In *International Conference on Computational Intelligence in Music, Sound, Art and Design*. Springer, 267–282.

[5] Philip Bontrager, Julian Togelius, and Nasir Memon. 2017. DeepMasterPrint: Generating Fingerprints for Presentation Attacks. *arXiv preprint arXiv:1705.07386* (2017).

[6] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. Openai gym. *arXiv preprint arXiv:1606.01540* (2016).

[7] Vincent Conitzer and Tuomas Sandholm. 2002. Complexity of mechanism design. In *Proceedings of the Eighteenth conference on Uncertainty in artificial intelligence*. Morgan Kaufmann Publishers Inc., 103–110.

[8] Paul Dütting, Zhe Feng, Harikrishna Narasimhan, and David C Parkes. 2017. Optimal auctions through deep learning. *arXiv preprint arXiv:1706.03459* (2017).

[9] Yaroslav Ganin, Tejas Kulkarni, Igor Babuschkin, SM Eslami, and Oriol Vinyals. 2018. Synthesizing Programs for Images using Reinforced Adversarial Learning. *arXiv preprint arXiv:1804.01118* (2018).

[10] Javier Garcıa and Fernando Fernández. 2015. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research* 16, 1 (2015), 1437–1480.

[11] Edoardo Giacomello, Pier Luca Lanzi, and Daniele Loiacono. 2018. DOOM Level Generation using Generative Adversarial Networks. *arXiv preprint arXiv:1804.09154* (2018).

[12] David Ha. 2017. Evolution Strategies Tool. https://github.com/hardmaru/estool. (2017).

[13] David Ha and Jürgen Schmidhuber. 2018. World Models. *arXiv preprint arXiv:1803.10122* (2018).

[14] Nikolaus Hansen and Andreas Ostermeier. 2001. Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation* 9, 2 (2001), 159–195.

[15] Erin Jonathan Hastings, Ratan K Guha, and Kenneth O Stanley. 2009. Automatic content generation in the galactic arms race video game. *IEEE Transactions on Computational Intelligence and AI in Games* 1, 4 (2009), 245–263.

[16] Nicolas Heess, Srinivasan Sriram, Jay Lemmon, Josh Merel, Greg Wayne, Yuval Tassa, Tom Erez, Ziyu Wang, Ali Eslami, Martin Riedmiller, et al. 2017. Emergence of locomotion behaviours in rich environments. *arXiv preprint arXiv:1707.02286* (2017).

[17] Edward Hughes, Joel Z Leibo, Matthew G Philips, Karl Tuyls, Edgar A Duéñez-Guzmán, Antonio García Castañeda, Iain Dunning, Tina Zhu, Kevin R McKee, Raphael Koster, et al. 2018. Inequity aversion resolves intertemporal social dilemmas. *arXiv preprint arXiv:1803.08884* (2018).

[18] Max Jaderberg, Wojciech M Czarnecki, Iain Dunning, Luke Marris, Guy Lever, Antonio Garcia Castaneda, Charles Beattie, Neil C Rabinowitz, Ari S Morcos, Avraham Ruderman, et al. 2018. Human-level performance in first-person multiplayer games with population-based deep reinforcement learning. *arXiv preprint arXiv:1807.01281* (2018).

[19] Niels Justesen, Ruben Rodriguez Torrado, Philip Bontrager, Ahmed Khalifa, Julian Togelius, and Sebastian Risi. 2018. Procedural Level Generation Improves Generality of Deep Reinforcement Learning. *arXiv preprint arXiv:1806.10729* (2018).

[20] Diederik P Kingma and Max Welling. 2013. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114* (2013).

[21] Ilya Kostrikov. 2018. PyTorch Implementations of Reinforcement Learning Algorithms. https://github.com/ikostrikov/pytorch-a2c-ppo-acktr. (2018).

[22] Marc Lanctot, Vinicius Zambaldi, Audrunas Gruslys, Angeliki Lazaridou, Julien Perolat, David Silver, Thore Graepel, et al. 2017. A unified game-theoretic approach to multiagent reinforcement learning. In *Advances in Neural Information Processing Systems*. 4190–4203.

[23] Joel Z Leibo, Vinicius Zambaldi, Marc Lanctot, Janusz Marecki, and Thore Graepel. 2017. Multi-agent Reinforcement Learning in Sequential Social Dilemmas. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, 464–473.

[24] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2015. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971* (2015).

[25] Zbigniew Michalewicz. 1996. Evolution strategies and other methods. In *Genetic algorithms+ data structures= evolution programs*. Springer, 159–177.

[26] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529.

[27] Julien Perolat, Joel Z Leibo, Vinicius Zambaldi, Charles Beattie, Karl Tuyls, and Thore Graepel. 2017. A multi-agent reinforcement learning model of common-pool resource appropriation. In *Advances in Neural Information Processing Systems*. 3646–3655.

[28] Sébastien Racanière, Théophane Weber, David Reichert, Lars Buesing, Arthur Guez, Danilo Jimenez Rezende, Adrià Puigdomènech Badia, Oriol Vinyals, Nicolas Heess, Yujia Li, et al. 2017. Imagination-augmented agents for deep reinforcement learning. In *Advances in Neural Information Processing Systems*. 5690–5701.

[29] Tuomas Sandholm. 2003. Automated mechanism design: A new application area for search algorithms. In *International Conference on Principles and Practice of Constraint Programming*. Springer, 19–36.

[30] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. 2012. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*. 2951–2959.

[31] Adam Summerville, Sam Snodgrass, Matthew Guzdial, Christoffer Holmgård, Amy K Hoover, Aaron Isaksen, Andy Nealen, and Julian Togelius. 2017. Procedural content generation via machine learning (PCGML). *arXiv preprint arXiv:1702.00539* (2017).

[32] Pingzhong Tang. 2017. Reinforcement mechanism design. In *Early Carrer Highlights at Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI, pages 5146–5150*.

[33] Roland van der Linden, Ricardo Lopes, and Rafael Bidarra. 2014. Procedural generation of dungeons. *IEEE Transactions on Computational Intelligence and AI in Games* 6, 1 (2014), 78–89.

[34] Vanessa Volz, Jacob Schrum, Jialin Liu, Simon M Lucas, Adam Smith, and Sebastian Risi. 2018. Evolving Mario Levels in the Latent Space of a Deep Convolutional Generative Adversarial Network. *arXiv preprint arXiv:1805.00728* (2018).

[35] Chiyuan Zhang, Oriol Vinyals, Remi Munos, and Samy Bengio. 2018. A Study on Overfitting in Deep Reinforcement Learning. *arXiv preprint arXiv:1804.06893* (2018).

[36] Haifeng Zhang, Jun Wang, Zhiming Zhou, Weinan Zhang, Yin Wen, Yong Yu, and Wenxin Li. 2018. Learning to Design Games: Strategic Environments in Reinforcement Learning.. In *IJCAI*. 3068–3074.

[37] Alexander Zook and Mark O Riedl. 2014. Automatic Game Design via Mechanic Generation.. In *AAAI*. 530–537.