Using Intrinsically Motivated Reinforcement Learning for Manipulation Robotics

Student Name: Lauren Stumpf
Supervisor Name: Professor Magnus Bordewich
Submitted as part of the degree of BSc Natural Sciences to the
Board of Examiners in the Department of Computer Sciences, Durham University

Abstract—**Context**: Manipulation robots are an integral part of manufacturing. However, the current generation of manipulation robots, struggle to adapt to changes in their environment because they have been explicitly programmed. If instead we used a reinforcement learning approach, the robot would be more capable of operating in less structured environments and would indeed be able to adapt to changes in their environment. However it is very difficult to engineer a well-shaped and dense reward signal to allow the agent to learn a manipulation task. Therefore, the rewards that the robot learns from are usually infrequent (sparse) and uninformative.
Aims: To produce a reinforcement learning agent capable of completing various manipulation tasks in spite of the sparse rewards. Specifically, we hope to produce a skillful agent, through combining and researching state of the art practices and architecture. This agent should be able to generalise across environments.
Method: We supplement the sparse task reward with a dense task-independent intrinsic reward. We then research and combine the most appropriate recent state of the art namely ensemble learning, hindsight experience replay, D2RL, R2D2 with some novel practices and architecture such as a new data augmentation scheme and dropout layers.
Results: We produce a Deep Ensembled Truncated Quantile Critic with Recurrent Networks which we define in this paper. The final agent is extremely competent and surpasses the performance of all surveyed approaches. It is able to complete the three manipulation tasks to varying degrees of success. It is also able to generalise across environments. We also demonstrate some of the best practices to exploit a predominately intrinsic reward signal.
Conclusion: We found that all included components bolster the performance of the agent with the exception of recurrent networks which are harmful to the generalisation power of the agent. The novel use of dropout and its impact on generalisation power is particularly promising. We also gain some valuable insights into how best exploit intrinsic rewards, namely that intrinsic rewards are best maximised by behaving less greedily and that stochastic agents work best. Overall, the generalisation power of our agent shows the value that reinforcement learning has in producing a flexible agent that does not put a precise requirements on the environment it is operating in.

Index Terms—Deep Learning, Manipulators, Reinforcement Learning, Robotics

1 INTRODUCTION

REINFORCEMENT learning (RL) has demonstrated an impressive ability to autonomously acquire complex and rewarding behaviours well suited to a chosen task. The ability of an RL agent to learn complex behavior without expert supervision, makes it well-tailored as a framework to address difficult problems where a solution is hard to manually engineer. One relevant domain where explicitly programming a solution can be challenging is robotics. This is a particularly intriguing avenue to explore as we draw closer to connecting directly to how humans learn - as an embodied agent in the real world.

With the continuous developments in mechanics, sensing technology, intelligent control and other modern technologies, robots have improved autonomy capabilities and are more dexterous. As a result of this evolving competence and the accuracy of these solutions, automation is an integral component of society and the economy. Conventionally, these robots are built upon rigid feedback control algorithm which are hand-engineered and require expert domain knowledge. These solutions place precise specifications on the operating environment as they are not able to account for variability.

An RL solution, on the other hand, provides robot agents with a high-level specification of what to do instead of how to do it. An agent autonomously learns an optimal policy through trial-and-error interaction with its environment. This learnt policy is capable of executing the optimal sequence of commands for accomplishing the task. By not detailing how exactly the task should be achieved, the robot is more capable of operating in less structured environments and is able to generalize to unknown objects. This is advantageous as it removes precise requirements and specifications on the operating environment and calibration of the robot. By removing these precise specifications, the robots become more independent as no workers are required to maintain the rigidity of the environment. This independence improves the improves the resilience of the supply chain with respect to fluctuations in the workforce (as was seen with COVID-19).

1.1 Nature and Challenge of Problem

Robots are often categorized according to their movement, degrees of freedom, and function. One specific type of robot is called an articulated robot also known as a robotic or manipulator arm. These robots are designed to move and manipulate objects using a robotic arm. As they are among the most common and useful robots found in industry, this paper chooses to focus on this type of robot. Instead of training a physical articulated robot to be capable of certain manipulative tasks, which is expensive and takes a long time to train (as it trains in real-time), we look at training a simulated articulated robot (which has been modelled on a real robot). Specifically, we look at the OpenAI Fetch environments [1] which are simulations based upon the physical Fetch articulated robot. These are visualised in Fig 1. The goal of these four environments is for the articulated robot to learn to grasp and move objects in order to achieve a series of manipulation tasks. In the Reach environment the agent is require to move its end-effector to the goal position (seen as a red circle in Fig 1). Whereas in the Slide, Push and PickAndPlace environments the agent is required to move the object to the goal position through either a sliding, pushing or picking behaviour. These kinds of behaviors can be sequentially concatenated to create behaviors with real value to industry and with the added benefit of flexibility. Hence, a solution to these environments is of great practical and corporate relevance.

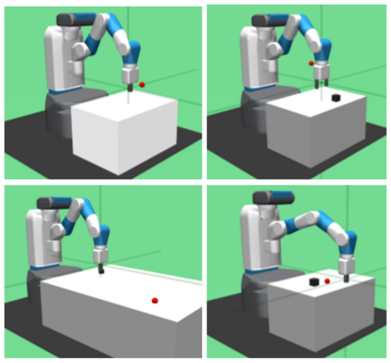


Fig. 1. The four Fetch environments. From left to right they are Reach, PickandPlace, Slide, Push

In RL, state of the art (SOTA) practice is established, generally on environments with dense task rewards such as Atari or OpenAI Mujoco [2] environments. These rewards have been specifically designed to help guide the agent learn to complete the specific task and are received at every step the agent takes (making them dense). However, in robotics it is often too complex to engineer such a reward signal as it requires expert domain knowledge. As a result, rewards in robotics tend to be sparse and binary. For example, in the Fetch environments the agent receives a reward of 1 if the goal (task) has been completed and 0 otherwise. As we show later, SOTA agents suitable for this environment (continuous control agents), really struggle to learn from such sparse and uninformative rewards and are often unable to complete these tasks. Hence applying SOTA practice effectively to these environments is difficult.

1.2 Deliverables and Achievements

Fundamentally, this project aims to create a skillful agent able of completing the Fetch manipulation tasks in spite of these sparse rewards. We place a focus on including practices to make it deployable to industry so the solution has practical value. We aim to produce an agent that outperforms other surveyed approaches and is capable of generalising across environments. We aim to do this by gaining a deep understanding of the most appropriate SOTA practices and architecture and combining these appropriately. We also aim to supplement the sparse task reward with a dense task-independent intrinsic reward and gain an insight into how best to exploit this reward signal. Moreover, we also aim to gain a deep understanding of the resultant agent and the value and compatibility of each component. A full list of the specific deliverables is available in the evaluation (5) section.

The final agent Deep Ensembled Truncated Quantile Critic with Recurrent Networks (DETRN+) out performs all other surveyed approaches and achieves impressive performance in spite of the sparse rewards. Moreover, the agent is capable of generalising to a different task (once it has learnt the new goal space). Our ablation studies demonstrate the value of each component with respect to the performance and generalisation power of the agent. These show that every component has a positive impact of performance expect the use of a recurrent networks which erodes the agent's generalisation power. The use of dropout to increase the generalisation power is particularly promising. We also demonstrate that task-independent intrinsic rewards help the agent learn a task-independent state representation which aids the generalisation power of the agent. Moreover, we gain some valuables insights into how intrinsic task independent rewards differ from extrinsic task dependent rewards and how best to exploit a predominately intrinsic reward signal. Namely task independent rewards are best exploited by acting less greedily and maintaining a consistent signal instead of pursuing large payoffs. We also find that stochastic agents are more able to exploit a predominately intrinsic reward signal in order to accomplish a task. Overall, we demonstrate that RL has real potential and value to manipulation robotics.

2 Related Work

Reinforcement learning has been extensively applied to robotics [3]. To place our paper into the wider context of the field we review literature related to the application of RL to manipulation robotics specifically. Manipulation robotics is when a robot applies motions or forces to purposefully change the state of the objects that are being manipulated. We note that we do not limit ourselves to literature on the Fetch environment specifically as there are a variety of benchmarks used and often the most significant solutions are bench-marked on real-world robots. To structure this discussion, we partition the existing literature into model-free and model-based solutions. We then review intrinsic reward literature whereby the reward signal is rich enough (can be calculated for each step the agent takes in the environment) to be applied to the Fetch environments. Again, we do not limit ourselves to the Fetch environments for intrinsic reward literature as often the most significant work is bench-marked on the Atari game Montezuma's Revenge (which has sparse rewards). Moreover, there is a lack of literature applying intrinsic rewards to robotics tasks.

In a model-based RL approach, a model of the system dynamics is learnt by learning a function which predicts state transitions given the current state and chosen action [3]. Often, we then substitute this model, for the real model of system dynamics, and use this to obtain a policy. Model-based approaches are advantageous as it allows the agent to plan and speculate between a range of different action choices. Moreover, by learning a model (as opposed to mathematically formulating a model) it allows the model to be tailored to the local system dynamics. As a result of this, model-based RL has been successfully applied to manipulation robotics ([4], [5], [6] [7], [3]). However, there are certain limitations in model-based RL that make it challenging to apply this approach effectively. We review the most notable literature that have removed the most significant barriers and as a result were able to apply model free RL effectively.

The performance of these approaches are hinged upon the learnt transition model which can either be deterministic or stochastic. Stochastic models result in predictions that are defined by a probability distribution over the future state. Deisenroth et al. [4] argues that whilst using a deterministic function approximator to model the transition function is more stable, the uncertainty of this model is not accounted for as these predictions are often taken in full confidence. They argue that by employing a probabilistic function approximator the uncertainty is more fully expressed. Therefore, they model the transition function probabilistically using non-parametric probabilistic Gaussian processes. The policy is then learnt using this model, via a policy-search based method. The strength of their insight and the resultant approach attains unprecedented levels of sample efficiency and stability on different manipulation benchmark tasks such as Cart-Pole Swing-up.

Furthermore, scaling to a high dimensional space, with model-based approaches, introduces additional complexity into an already complex model and hence has proven very difficult [8], [9]. Kumar et al. [5] was encouraging as they were able to scale their model-based solution to a 100-dimensional continuous state space. Kumar et al. [5] did this by modeling the system dynamics by using a time-varying linear model. This model was learned from on-policy data using linear regression with a Gaussian mixture model. The learnt model is then used to compute the optimal policy by computing the value function and Q-function recursively. The resultant agent is able to complete various dexterous manipulation skills proficiently despite the high dimensional action and state space.

Often model-based approaches are only capable of learning a policy that is capable of control and struggle to learn a policy capable of both perception and control. However, it is preferable to create an end-to-end system capable of both perception and control for convenience and computation expense. Levine et al. [10] was encouraging as it learns a model that can be used to create a policy capable of sensorimotor control by jointly training both the perception and control systems. They model the system dynamics using a linear-Gaussian model and parameterise the policy by a deep convolutional neural network. This approach results in a skillful agent that is able to outperform other surveyed policy search methods in different manipulation tasks that need direct coordination between the control and vision systems such as screwing a lid on a bottle.

One limitation that plagues model-based approaches, is the need for human involvement and feedback at different stages of the pipeline. Finn et al. [6] alleviates this by learning a model in a fully self-supervised manner negating the need for a dense reward signal, human feedback or an image of the goal. Finn et al. [6] trains a deep image predictive model using a large data set of robotic pushing experiences and uses this to endow the agent with visual foresight. The deep image predictive model learns a distribution over a series of future image frames that could happen given a specific action. This model is then utilized to establish the series of actions where the probability of the desired pixels being moved to a desired goal position is a maximum. The final agent is shown to complete a variety of pushing-based manipulation tasks skillfully and can generalise to new objects that were not seen during training.

These pieces of literature demonstrate the value and success of a model-based RL approach. However, one limitation of this framework is that the ground-truth model of the environment is not available to the agent. Consequently, any bias present in the model will be exploited by the agent resulting in an agent that behaves optimally with respect to the learned model however behaves sub-optimally in the real environment. Moreover, in certain unstructured and unregulated environments learning an accurate model of these environments has a higher computational complexity than a model free approach [3]. As the Fetch environments constitute complex dynamics (as discussed in 3.2) it is likely that a model based approach would have an unfeasible complexity given the time and resource constraints of this project.

In contrast, in a model-free approach we chose not to explicitly model the environment but learn an optimal behaviour through interactions with the environment. Generally, a model-free approach is the most common practice when approaching an RL task. This is because, in environments where the dynamics are complicated (as in real world environments) it is preferable to avoid formulating a model and alternatively learn a policy that is implicitly aware of these dynamics without actually modeling them. However these methods often require a prohibitive number of interactions with the environment making them sample inefficient. Moreover, SOTA agents are established on environments that have a well-shaped and engineered reward function specifically designed to help the agent learn the optimal policy. In contrast, in robotics, the agent often receives sparse and binary rewards as it is too difficult to engineer a dense and informative reward function. However, SOTA agents are unable to to learn a reasonable policy from these sparse and binary rewards. Therefore the naive and direct application of these SOTA agents is ineffectual. As a result, there are multiple frameworks introduced in literature that can exploit the benefits of model-free approaches while mitigating the issues. We now review different model-free practices that would allow us to apply a model-free approach.

The issue of sample efficiency can be mitigated by taking a distributed approach. Importance Weighted Actor-Learner Architecture (IMPALA) [11] uses a distributed actor-critic framework to learn a policy and value function by decoupling collecting experience and learning the policy and value function. Multiple distributed actors repeatedly interact with the environment and generate trajectories and a separate learner learns from these trajectories. In IMPALA the magnitude of each update from each actor is independent of the competence of the actor. Hessel et al. [12] builds up the IMPALA framework by introducing a more complex actor-critic update using PopArt normalisation [13] to factor in the competence of an agent when updating the policy. As a result Hado et al. [13] is able to surpass human-level performance on the set of 57 diverse Atari games.

Moreover, to allow the agent to learn from sparse rewards, we can supplement it with additional guidance in the form of an expert. In imitation learning an agent is given access to an expert and then tries to match the behavior of the expert. For example, in a manipulation task the agent could be given a video of a human completing the manipulation task and try to learn to imitate this behaviour. The simplest form of imitation learning is behaviour cloning which casts the imitation learning problem into a supervised learning problem by using the demonstration data as the supervised training data to learn the robot's policy. Ross et al. [14] follows this approach to learn to play Mario Kart by learning from demonstrations of the correct steering (provided by a human expert). However, the sequential nature of the data collected violates the independently and identically distributed assumptions of supervised learning. Moreover, this approach is also disadvantageous as an expert is required to be present across the course of training to provide feedback to the agent.

Imitation learning can also be achieved through inverse reinforcement learning whereby an agent tries to learn the reward signal of the environment. It specifically tries to learn the reward signal under which the demonstration data is the optimal policy. It then uses this learnt reward signal to learn the optimal policy. Syed et al. [15] and [16] take a game-theoretic view of inverse RL and reduce the inverse RL problem into a zero-sum two-player game where the first player chooses policies and the second chooses reward functions. However, unfortunately these approaches are not able to scale well to high-dimensional continuous inputs. Guided Cost Learning (GCL) [17] and Generative Adversarial Imitation Learning (GAIL) [18] are both able to scale to high-dimensional continuous inputs. The algorithmic structure of both algorithms are very similar. They both simultaneously learn the reward signal and policy that mimics the expert. At each step, trajectories are sampled from both the current and expert policy and are then used to formulate a reward function. This reward is then optimized to generate an updated policy and this procedure is iterated until the desired level of convergence. Specifically, in GAIL the reward is obtained from a discriminator network that is trained to discern between trajectories generated using the expert policy and trajectories generated using a generator (the learnt policy). The policy is trained using TRPO [19]. Conversely, in GCL, the reward is obtained by minimizing the Maximum Entropy IRL cost [20] which can be optimized by any model-free algorithm. However, despite the success of these imitation learning approaches, we do not have access to the physical environments to provide demonstration data for the agent. Moreover, these approaches also require an expert to be able to provide feedback across the course of training which is also infeasible for this project.

An alternative strategy to help the agent learn from sparse rewards, is to supplement the sparse task-dependent reward with a dense task-independent intrinsic reward. Intrinsic rewards are computed by the agent itself and help the agent develop useful behaviours that help the agent complete the task. Intrinsic rewards can be partitioned according by the behaviors that it causes the agent to display, namely knowledge acquisition or skill learning [21]. Knowledge acquisition describes an intrinsic reward signal that motivates the agent to accumulate knowledge of its environment such as the things it can and cannot control, novel areas or a sense of proximity. Conversely, skill learning describes an intrinsic reward signal that motivates the agent to develop the ability to competently learn new skills. We note that this boundary is not strict and sometimes an intrinsic reward signal can cause the agent to display both of these. A review of skill learning literature ([22], [23], [21]appropriate for tasks that can be deconstructed into a series of difficult skills. As the tasks in each Fetch environment can only be deconstructed into one or two skills these methods are not appropriate. To discuss knowledge acquisition literature, it is best to section them by the type of knowledge it helps the agent accumulate. Specifically, it can help accumulate knowledge that describes the environment or knowledge about how best to control the environment.

An intrinsic reward that causes an agent to acquire knowledge of its environment is generally formulated in one of two way, either as the novelty of the state or the prediction error. When representing the intrinsic reward as the prediction error, literature follows a similar framework whereby the intrinsic reward is given by $R(s_t, s_{t+1}) = ||g(s_{t+1}) - F(g(s_t), a_t)||$ where $g(s)$ is a function that projects the state space to a latent feature space and $F(s, a)$ is a model of the system dynamics which predicts the next latent state of the environment given the current latent state (which is found by transforming the current state of the environment by $g(s)$) and chosen action. This reward rewards the agent for pursuing states that, when projected into a latent feature space, result in the transition function predicting a different latent state than otherwise encountered. This error indicates that predicting these states are difficult which are heuristically valuable states to explore. Usually F is approximated by a neural network that predicts the next latent state given the action and current latent state. Both $F(s, a)$ and g are pre-trained separately from (before) the agent. This ensures the intrinsic reward is consistent. As we have seen earlier, learning a model of the state transitions can be very difficult. To tackle this, literature pursues making g a function that is able to produce a simple feature space thereby reducing the burden on F. Burda et al. [24] first proved that taking g as the identity mapping is ineffective when the state space is large and conversely recommended taking g as a convolutional network, fixed after random initialization. Conversely, [25] takes g to be the compressing component of a dynamic auto-encoder. Stadie et al. [25] shows that this method outperforms other surveyed work in terms of exploration (using an exploration metric they introduce in the paper) on the Atari Learning domain. However, both these methods struggle to account for the stochasticity of the environment and confuse this with an error in the model.

This results in the white noise problem whereby the intrinsic reward signal encourages agents to pursue random noise in the environment. As these parts of the state space cannot be modeled (as they are random) this results in a large prediction error which causes the agent to keep pursuing these uninformative states.

The intrinsic curiosity module (ICM) [26] mitigates this issue by putting an extra constraint on g. g is approximated by a neural network with two sub-modules. The first projects a state s into a latent feature space through a transformation denoted by $\phi(s)$. The second module takes two subsequent states s_t and s_{t+1}, projects both of these using the first sub-module $\phi(s)$ and then predicts the action that caused the transition from the first state to the second. The neural network is trained to minimize the loss between the predicted action and the actual action. For environmental features that are not effected by the agent's actions this error will give a low intrinsic reward (as error includes ability to predict a_t) thereby dissuading the agent to pursue states that are inherently unpredictable making it robust to the white-noise problem. They demonstrate that the resultant agent is able to significantly outperform the baseline (that is not given any intrinsic reward) in the Super Mario Bros.

Alternatively, exploration with mutual information (EMI) [27] mitigates the white noise problem by also tackling the construction of the latent space. EMI learns the function g for both states and actions via variational divergence estimation of mutual information. Specifically it learns compact representations by minimizing the uncertainty for the action a_t given s_t, s_{t+1} and the uncertainty for s_{t+1} given a_t, s_t. EMI makes the impactful choice to constrain F to be represented by a linear model which is endowed with a model error (which accounts for the error arising from approximating the system dynamics with a linear model). This choice of a linear model is significant as it substantially transfers the complexity of the modeling burden onto g. This transfer manages to construct a latent space free from the white-noise problem. The resultant agent is able to outperform four other surveyed approaches (incluing ICM) on two challenging locomotion environments.

However in general, prediction error methods (both ICM and EMI) struggle to encode features in the latent space g that evolve overtime. As result, these methods are not appropriate for the Fetch environments as in manipulation tasks objects positions evolve overtime. An alternative intrinsic reward signal that encourages the agent to acquire meaningful knowledge of its environment is formulating the reward according to the novelty of the state transitioned in. In a discrete action space, this can be formulated in a count-based method [28] whereby $R(t) = \frac{1}{N(s_t)}$ where $N(s_t)$ refers to how many times that state has been visited. Evidently, for a continuous or large discrete action space, this translates to an unfeasible space complexity. Consequently, a pseudo-count is often computed and so the intrinsic reward is given by $R(t) = \frac{1}{\hat{N}(s_t)}$ where $\hat{N} = \frac{\rho(s)(1-\rho'(s))}{\rho'(s)-\rho(s)}$. Here $\rho(s)$ is a density model which literature approximates with different models.

Bellemare et al. [29] creates a latent feature space of the state space and constructs a density model over this feature space. It does this through mapping a series of feature vectors (a series of consecutive states that have been mapped to the feature space) to a probability distribution and constructs a density model using different independent feature distributions. This density model assigns a larger count to feature vectors that have similar features with regularly visited states. Ostrovski et al. [30] alternatively takes the density model as the SOTA neural density model PixelCNN [31]. However, computing these pseudo-counts for both these methods is still very computationally expensive. ϕ-EB [32] decreases the computational expense of a pseudo count by modeling the density model on the feature space already created by the computation of $V(s)$. ϕ-EB achieves impressive results on Montezuma's revenge when factoring in the reduced computational cost.

Whilst formulating intrinsic reward like this is effective in accumulating knowledge regarding the structure and content of an environment, it is often more useful to learn how best to exploit the dynamics of the environment. To do this, we can look at intrinsic rewards that encourage the agent to accumulate knowledge of how best to manipulate the environment. This form of knowledge is the most useful form of knowledge for the Fetch environments as they consist of a series of manipulation tasks. Formally, the process of an agent accumulating knowledge related to how best to manipulate the environment is called empowerment. Empowerment is denoted by Σ and is mathematically formulated as the mutual information between an action at time t and state at time t. A high amount of mutual information between an action and a state indicates the action is highly aligned with the state and the action taken is impactful on the state (indicating a high degree of control). Formally it is given by

$$\Sigma(s_t) = max_n I(a_t^n; s_{t+n}) \tag{1}$$

where I represents the mutual information and $a_t^n = (a_t, a_{t+1}, .., a_{t+n})$ which corresponds the actions executed by the agent from time t to time $t + n$. By formulating the intrinsic reward signal as Σ the agent is encouraged to maximise its empowerment. However, equation 1 is computationally expensive to compute and as a result it is usually approximated. Mohamed and Rezende [33] approximate equation 1 by computing a lower bound (that is easier to compute) for the mutual information and then using this lower boud to compute the empowerment. To do this they learn an approximator q_ξ in a supervised fashion from environment transitions using maximum likelihood estimation. Mohamed and Rezende demonstrate the value of their approach by outperforming all surveyed intrinsic rewards on both static and dynamic environments. Alternatively, [23] alternatively approximates 1 using the policy and the observation history. They demonstrate that the intrinsic rewards formulated in this way show a high degree of alignment with maximising extrinsic rewards.

Our paper proceeds to build on the reviewed literature. We choose to take a model-free approach. This is because learning a model would be computationally more expensive for the Fetch environments than a model-free approach. Moreover, the learnt model would be likely to incorporate bias that would be passed to the agent and so the agent would not be capable of acting optimally in the real environment. Model-free literature is rarely applied successfully to the Fetch environments because model-free RL is rarely used in robotics. In our paper, we combine SOTA practice in model-free RL in a way that no other paper has done before. We build on the reviewed model-free literature as we choose to do this in a more feasible way than for example a distributed approach across multiple GPUs or requiring an expert signal. We also supplement this agent with an intrinsic reward signal that aims to empower the agent and outperforms the surveyed empowerment intrinsic rewards. Whilst the intrinsic reward signal we choose to use has been applied to the Fetch environments we build upon this framework as we identify certain areas for improvement in the way it has been applied which we aim to change.

3 Solution

3.1 Background Theory - Markov Decision Process and Goal-Conditioned Reinforcement Learning

A Markov decision process (MDP) [34] is a mathematical framework for modelling decision-making in an environment. It is characterised by the tuple (S, A, T, R) where S is the finite set of possible states in the environment; A is the finite set of feasible actions that the agent can take; T is the (stochastic) transition function $T : S \times A \rightarrow S$ that returns the next state given the chosen action and the current state; and R is the (stochastic) reward function $R : S \times A \rightarrow \mathbb{R}$ that given the current action and current state returns a reward $r \in \mathbb{R}$. This framework can be extended to model decision-making in environments where we have partial observability. Here we model the process as a Partially Observable Markov Decision Process (POMDP). In a POMDP the true state of the environment is hidden from the agent and instead the agent receives observations (as opposed to the state). An observation is an incomplete description of the true state of the environment. A POMDP is more appropriate to model a decision-making process in robotics (than a MDP) as for physical robots the true state of the agent will rarely be available. For example, information regarding the mechanical wear or heating of system will be hidden from the agent. Therefore the decision making process will have to revolve around partial information. Formally, a POMDP extends the tuple used in a MDP. A POMDP is described as a 6-tuple (S, A, T, R, Ω, O) where Ω denotes the finite set of valid observations and O is the (stochastic) observation function $O : S \rightarrow \Omega$ that maps the underlying true environment state to an observation. An observation can be denoted by o or s, we have chosen o and retain the notation o' to denote the next observation. We also note that we follow convention and use capitalised letters to describe random variables and lower case letters to denote the actual value of the random variable.

Reinforcement learning is one of three machine learning paradigms. It aims to train an agent that learns a policy which maximises the expected reward. Conventionally it is modelled by a framework set out by [35] whereby the agent interacts with the environment in discrete time steps $t = 0, 1, 2...T$. At each time step the agent takes an action a_t as advised by the policy μ in response to the observation $o_t \in \Omega$. The environment then returns a reward $r_t \sim R(s_t, a_t)$, transitions from s_t to $s_{t+1} \sim T(s_t, a_t)$ and provides a new observation $o_{t+1} \sim O(s_t)$ to the agent. This process is iterated for a set number of time steps and the policy is optimised in someway to maximise the expected reward $\sum_{t=0}^{T} \gamma^t r_t$ or to maximise the average reward $\frac{1}{T} \sum_{t=0}^{T} \gamma^t r_t$ where γ is the discount factor $\gamma \in [0, 1]$. The framework that we use in this paper to optimise the policy are different forms of Q-Learning. In Q-Learning a Q-function $Q^\pi(s, a)$, also known as the action-value function, is learnt which tells us how valuable state action pairs are under the policy π. We can then use this Q-function to assess and optimise the policy.

The framework set out above produces an RL agent that can only achieve the task specified by its reward function. In situations where an agent needs to perform a diverse set of tasks (such as moving objects to different locations) this framework does not scale well. Under this framework we would need to learn different policies concurrently, each with a unique reward function engineered to achieve each unique task. However, this is computationally inefficient as it requires a different policy for each goal (which is infeasible when the goal space is large) and does not leverage the shared structure between different goals. Alternatively, goal-conditioned reinforcement learning (GCRL) [36] [37] is an extension of the RL framework whereby a policy is learnt that can generalise across goals as well as states.

Formally, GCRL extends the framework of a MDP (or a POMDP) with an additional term G where G denotes the goal space. It is common for a goal to be certain state and so often the goal space is a subset of the state space. This is preferable as then the goal space inherits the structure of the state space. The reward function $R : S \times A \times G \rightarrow \mathbb{R}$ is slightly different as it is now a mapping from a state $s \in S$, action $a \in A$ and goal $g \in G$ to a reward $r \in \mathbb{R}$. In robotics, the reward function is often taken to be binary (1 if the goal is achieved and 0 otherwise). This is advantageous as it does not require any expert domain knowledge and is simple to engineer. The policy π becomes goal conditioned such that $\pi : S \times G \rightarrow A$. It aims to maximise the expected reward $\sum_{t=0}^{T} R s_t, a_t, g_t)$ or the average expected reward $\frac{1}{T} \sum_{t=0}^{T} R(s_t, a_t, g_t)$ (where g_t is the same goal for the whole trajectory). This produces an agent that is capable of achieving any arbitrary goal $g \in G$.

3.2 Fetch Environments

As previously mentioned, in this paper we focus on solving the Fetch environments [1]. In order to do this we follow a GCRL framework as in the Fetch environments the agent is required be able to generalise across goals (where the goal space is subset of the state space). The Fetch environments consists of four separate environments namely FetchReach, FetchPush, FetchPickAndPlace and FetchSlide. In all these environments the agent is required to complete a specific manipulation task using an articulated robotic arm that is based on the 7 degrees of freedom Fetch robotics arm which has a two-fingered parallel gripper. Three of these manipulation tasks (FetchPush, FetchSlide and FetchPickAndPlace) require the robotic arm to move an object to a 3-dimensional goal position. In the FetchPush environment, the robot is required to move a box on a table to a goal position on the table. The gripper's fingers are locked which prevents the agent learning a grasping behaviour. This task can be achieved through pushing - although sometimes an agent can learn a rolling behaviour. In the FetchSlide environment, a puck is placed on a table and the coefficient of friction between the puck and the table is deliberately set to be very low. As the target position is placed outside of the robot's reach it is required to hit the puck in such a way that it slides and arrives at the target location. In FetchPickAndPlace, a box is placed on the table and the agent is required to move it to a selected position. This position may be on the table or in the air and as a result requires the agent to learn a grasping behaviour. The fourth, FetchReach, is the simplest of these tasks and simply requires the end-effector to move to a 3-dimensional desired position.

In all of these environments, a reward of one is given if the object (end-effector for FetchReach) is with 5cm of the target location and zero otherwise. The action space is four-dimensional with 3 dimensions describing the movement of the gripper-hand in Cartesian coordinates and the fourth dimension controlling whether the gripper is open or closed. Once the agent chooses an action, this action is repeated for 20 simulator steps each executed 0.002 seconds after each other, before control is returned to the agent. The observation an agent receives consists of the 3D position of the gripper (described in Cartesian coordinates) and its linear velocity. If the environment contains an object (all environments except FetchReach) the observation also includes the object's 3D position (described in Cartesian coordinates), its rotation (using Euler angles) and its angular and linear velocities.

3.3 Intrinsic Rewards

As the reward signal in the Fetch environments is very sparse and uninformative, it is extremely difficult to learn from. This is because if the agent does successfully complete the task it struggles to attribute credit to the actions and states that were helpful in achieving this. This is called the credit assignment problem. Furthermore, the agent is not well guided by the reward signal as there is no intermediate feedback provided. Moreover because the reward is the same magnitude regardless of how the task is completed the agent struggles to learn the most efficient way to complete the task. To mitigate these issues associated with a sparse uninformative task reward signal, in this paper we supplement it with a task-independent dense intrinsic reward which can be computed for every step the agent takes.

In the framework set out in the previous section (see section 3.1) the reward r is an external reward supplied by the external physical environment (outside of the agent). This external reward can be seen as a form of extrinsic motivation as it motivates the agent to complete a specific task and comes from an external source (to the agent). Alternatively, when a behaviour is acquired without any direct feedback from the environment but rather by an internal intrinsic reward, this reward signal is referred to as intrinsic motivation [38]. Intrinsic rewards are more general and do not have to be redesigned for different tasks. This type of motivation has roots in developmental learning which describes the trend of babies to learn by exploring and interacting with their environment. Here the babies are moved to explore the environment because it is inherently enjoyable not because of any specific rewarding outcome supplied by the environment.

To formalise this idea, we divide the environment (using the boundary of the physical agent) into the agent and the environment external to the agent. For example, for a robot the agent refers to everything inside the casing of the robot and the external environment refers to everything outside of the casing of the robot. The final reward signal is now computed by the agent which produces a reward by combining the task reward r_e (supplied by the external environment) and any intrinsic reward r_i (computed in the internal environment). This framework is shown in Fig 3, with the conventional framework being shown in Fig2. As we have separated the environment this is also reflected when the agent is given observations of the environment. Specifically, the state of the environment is broken down into the agent state (denoted by s^a) which describes the state of the agent and the external environment state (denoted by s^s) which describes the state of the environment outside of the agent. For example, in the Fetch environments the agent state s^a is the information describing the robotic arm (for example its position). Conversely the environment state consists of the information related to the external environment (for example the position of the object). The agent then receives both of these states as observations (as they have been transformed by the observation mapping O).

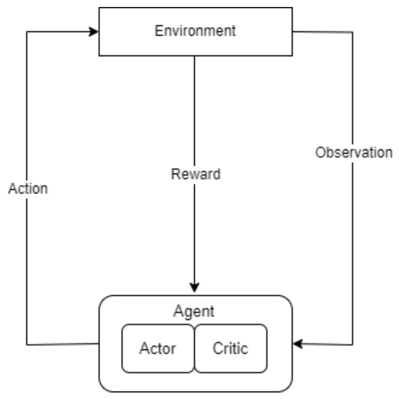


Fig. 2. Framework with only extrinsic task rewards

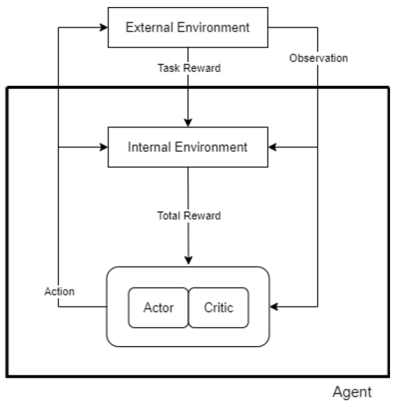


Fig. 3. Framework with both intrinsic task-independent rewards and extrinsic task rewards

As discussed in the related work section 2, there are multiple ways to formulate this intrinsic reward. In this paper we choose to formulate the reward as the mutual information between the agent and the external environment as introduced in [39]. Formally the intrinsic reward is given by $r = I(S^a; S^s)$ where I represents the mutual information, S^a represents the random variable for the state variable of the agent and S^s represents the random variable for the state variable of the external environment. By maximising the mutual information between the agent's state and the external environment's state, it maximises the control the agent has over its environment as a high mutual information indicates that the agent state is impact on the environment state. This is specifically appropriate for the Fetch environments as if the agent has control over the environment then it able to manipulate the environment.

The mutual information I between the two random variables S^a and S^s measures their mutual dependence. Specifically, it is indicative of the amount of information we gain about variable S^a while observing variable S^s. However in practice calculating $I(S^a; S^s)$ is difficult as both S^a and S^s are continuous variables and exact computation of the quantity $I(X; Y)$ is only tractable for discrete variables (or a small family of problems where the probability distributions are known). To be able to feasibly compute $I(S^a; S^s)$ we use a neural network (which is called a Mutual Information Neural Estimator) as a function approximator in a technique introduced in [40]. We approximate $I(S^a; S^s)$ by the learnt model $I_\Phi(S^s; S^a)$ where Φ are the parameters for the neural network. The Mutual Information Neural Estimator is trained using stochastic gradient descent using samples from interactions with the environment.

In order to combine the two separate rewards we use a weighted sum $R = \theta r_i + (1 - \theta) r_e$ where θ is a hyperparameter which we take to be 0.2. In our initial first agents we found that just adding the rewards together (without the parameter θ) meant that the extrinsic task reward was overwhelmed (as it was so infrequent compared to the intrinsic reward). Which we solved by scaling down the magnitude of the intrinsic rewards relative to the extrinsic rewards.

3.4 Architectural Overview and Design Choices

Having set up the environment and reward signal we now focus on describing the RL agent we have implemented. These design choices are a result of surveying literature and early experimentation. All components are included to help the performance and generalisation power of the agent.

3.4.1 Off Policy Agents

The way the policy is learnt or optimised depends on the specific agent chosen. State of the art RL agents have been shown to be difficult to reproduce [41], brittle [41], and very sensitive to the random seed [41] and implementation [42]. This conjecture is best exemplified with the standard state of the art Soft Actor Critic (SAC) [43] and Twin Deep Deterministic Policy Gradient (TD3) [44] both presenting to beat the other in the OpenAI gym continuous control tasks [2] in their respective papers. This conjecture makes it difficult to confidently assert which state of the art is most appropriate for the Fetch environments. Additionally, because the reward signal is predominately an intrinsic reward (as the extrinsic task reward is rarely issued) it is not immediately obvious which agent would learn best from this kind of signal. We cannot immediately assume an agent capable of learning an optimal policy from a task-dependent extrinsic reward is capable of learning an optimal policy from a task-independent intrinsic reward and that maximising the intrinsic reward in the same way would lead to task success. For example, in general a task-dependent reward might have a higher variability as there is a large payoff for achieving the task and low reward otherwise. Whereas as an intrinsic reward does not aim to achieved a specific task (it is just encouraging general helpful behaviour) it is likely to be a flatter reward signal as it does not strongly emphasis or strongly reward any specific task. Therefore, we might expect the Q-function learnt for an intrinsic reward to be a lot flatter (as the state action pairs are all more similar in value) than the Q-function for an extrinsic reward. This would mean that it is less consequential or likely for the agent to get stuck in a local minimum when choosing an action with a flatter Q-function. Therefore it might be less suited to the stochastic nature of SAC or TQC where their stochastic nature helps prevent the agent from converging in a local minimum. Conversely, the stochastic nature of SAC or TQC maybe be advantageous to help generalise to the more diverse or random intrinsic reward signal (as they are not engineered to towards a specific goal) which would lead to a bumpier optimal Q-function. Therefore it is not immediately obvious which agent is most suited to exploit a predominately intrinisic reward function.

Thus as part of our solution we implement four different off-policy agents to investigate and gain an insight into which components of these agents are most appropriate to learn from the predominately intrinsic reward signal. Moreover, our conclusions will be more confident as we will have tested different competitive alternatives. We implement four off-policy agents to compare, one previous SOTA (Deep Determinisic Policy Gradient [45]), two current SOTA (SAC and TD3) and one very recent SOTA (Truncated Quantile Critic [46]) and will evaluate the performance in practice. A full description of each of these algorithms is outside the scope of this paper however we briefly contrast these algorithms so we can discuss and attribute success to specific components of each algorithm.

All of these algorithms use an actor-critic framework. In this framework we use two separate structures to represent the policy and the estimated value function. The policy structure is known as the actor which selects the action. The estimated value function structure is known as the critic as it critiques the action chosen by the actor by assessing how good the action was and how it could improve. The critic is often taken to be a Q-function $Q^\pi(s, a)$ which returns the value of state s, action a pairs under the policy π.

Deep Determinisic Policy Gradient (DDPG) [45] interleaves learning an approximator to $Q^{*(s,a)}$ (the optimal critic) with learning an approximator to $a^{*(s)}$ (the optimal actor). Lillicrap et al. [45] demonstrates that the naive application of neural networks as an approximator to either function is unstable. Therefore to be able to use neural networks that produce stable values they compute target values by creating a copy of both the actor and critic networks. They necessitate that the target neural networks lag behind the normal networks by updating the weights of these target networks infrequently using polyak averaging $\theta' \leftarrow \tau\theta + (1-\tau)\theta'$ where τ is a hyper-parameter taken to be less than 1, θ' are the target network parameters and θ are the normal network parameters. These target values are more stable as they are constrained to update slowly. To help exploration they add Ornstein-Uhlenbeck noise to the chosen action while training.

TD3 makes three critical method changes to DDPG. Firstly, it uses clipped double-Q learning whereby two Q-functions are learnt and the smaller Q-value is used as the target. The use of the smaller Q-value as the target, causes the Q-value to regress towards this smaller value which helps to mitigate overestimation of the Q-function. Secondly, it updates the Q-function more frequently than the policy. This delayed policy update helps reduce the reactivity present in DDPG which stabilises learning. Finally, noise is added to the target action to help smooth the target policy. Target policy smoothing addresses a behaviour of DDPG observed in practice whereby the Q-function erroneously develops a sharp peak for some actions which the policy proceeds to exploit. As this sharp peak is not a reflection of the value of these actions but rather an error that has been exaggerated, this results in brittle and disadvantageous behaviour. By smoothing out the target policy with the addition of noise, the Q-function is smoothed over similar actions.

SAC, published roughly concurrently, is similar to TD3 in the respect that it incorporates clipped double-Q learning and the stochasticity of the policy in SAC give an effect similar to target policy smoothing. However, the principle attribute of SAC is entropy regularization. In SAC, the policy tries to maximise both the expected cumulative return and the entropy of the policy. This is advantageous as it, in practice, helps stop the agent from converging to unsatisfactory local optimum. Moreover, it helps strike a suitable balance between exploration-exploitation whereby the exploration (caused by entropy regularisation) does not necessarily come at the cost of exploitation.

Truncated Quantile Critic (TQC) combines SAC's entropy regularisation framework and a distributional approach. It approximates the return random variable $Z^{\pi}(s,a) = \sum_{t=0}^{\infty} R(s_t, a_t)\gamma^t$ in contrast to approximating the expectation of this return (which is the Q-function). It approximates this random variable $Z^{\pi}(s,a)$ by using N (a hyper-parameter to be set) Dirac distributions with each distribution having a support of m (a hyper-parameter to be set) atoms. For each distribution, the set of atoms is truncated at the right tail by removing atoms with the largest locations and $Z^{\pi}(s,a)$ is approximated by averaging across the location of the remaining atoms. This truncation is introduced to help stop overestimation. This distributional approach is advantageous as it is more conducive to learning the intrinsic randomness (aleatoric uncertainty) of the environment and policy. Moreover, we can exploit the granularity of distributional representation to help control overestimation with greater precision.

3.4.2 *Ensemble Learning*

The transition function used by the physics simulator Mujoco (which the Fetch environments are based upon) is stochastic. This allows it to account for the effects of mechanical variability that would likely occur in a physical robot. However, it makes the environment stochastic which is a source of action dependent random noise. This noise can be very harmful to off-policy algorithms as it is one of the main causes of overestimation bias. Overestimation bias refers to when the value of certain state actions pairs are overestimated by the Q-function. The impacts of overestimation are particularly exacerbated in off-policy learning grounded in temporal difference learning as the errors are propagated backward through episodes and subsequently accumulate over the learning process. Kumar et al. [47] showed that this error propagation and the resultant overestimation of the Q-function can cause unstable convergence and inconsistency resulting in an a brittle and unstable solution.

In initial experimentation we observed that some of the agents we first trialed were often pursing non-rewarding environments states (specifically the corners of the environment when we played the video). On further analysis, we were able to attribute this to the overestimation of the Q-function. Here homogeneous states often had varying Q-values (where we expected similar Q-values because of the homogeneity) associated with them that were often too high with respect to rewarding states (e.g. states near the goal state).

To address this shortfall, we decided to use ensemble methods in our solution. Ensemble methods combine multiple critics and/or actors in a special way to mitigate the overestimation. For example, the use of double Q-learning (with two critics), in TD3, is an example of an ensemble method however we consider a more extensive framework. In our solution we implement 5 actor-critic pairs. The off-policy algorithms we consider (TQC, SAC, TD3, DDPG) are all grounded in temporal difference (TD) learning. This means that the updates for the actor and critic are based upon the TD error. The TD error is given by the difference between the ultimate correct reward that we would obtain following the policy π from the current state s_t and the current prediction (made by the critic) of this reward. We use the critic target network(s) to approximate the ultimate correct reward and the normal critic network(s) for the current prediction. Mathematically this is usually formulated as

$$TD_{error} = Q^{\pi}_{\theta}(s_t, a_t) - r_t - \gamma \overline{V}^{\pi}(s_{t+1}) \tag{2}$$

where θ denotes the parameters of the normal critic network and $\overline{V}^{\pi}(s_{t+1})$ denotes the ultimate correct reward that we would receive following policy π starting from s_{t+1} (as calculated by the target critic). We note that we can compute the value function from the Q-function as $V^{\pi}(s) = Q^{\pi}(s, \pi(s))$. When updating each critic we multiple 2 (the TD error) by $w(s,a) = \sigma(-\overline{Q}_{std}(s,a) \times T) + 0.5$ as introduced in [48]. Where $\overline{Q}_{std}(s,a)$ denotes the standard deviation of all 5 target Q-values. This helps stabilise learning and mitigate the overestimation bias as it make the updates, where the target Q-functions have a high variance, smaller. Conversely, the 5 actors do not communicate, even when updated, however when the agent is required to choose an action the action is taken to be the average of all of the 5 actors chosen action as introduced in [49]. We have visualised this set up in Fig 4. We have chosen to shown two actor critic pairs instead of five to make the diagram clearer.

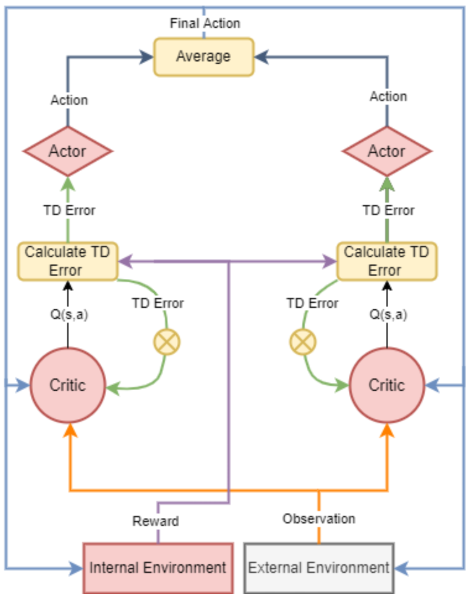


Fig. 4. Figure visualising how the actor and critic are updated. We use red to highlight the parts that are part of the agent and grey to denote the external environment. The reward, observation, action and final actions are highlighted by using purple, orange, dark blue and light blue. The calculations are made clear by using the brown yellow colour. The circle with a cross through it denotes the calculation and multiplication of the TD Error by $w(s, a)$

3.4.3 Neural Network Architecture

Both the actor and critic are approximated by neural networks. We now discuss the structure, size, type and initialisation of the neural networks we have chosen to use to do this. The final architecture of the critic can be seen in Fig 5 on page 11. The actor is very similar except it is only fed the state and goal and outputs an action instead of a Q-value.

We use a wide neural architecture as it helps the neural network learn more complex relationships [50] [51] thereby producing a more skilful agent. Specifically, we tie the width of the network to the complexity of the environment, reserving the widest neural architecture for PickAndPlace (512) and Slide (512) because these are the most complex environments. By using a wide network for the most complex environments, the neural network is able to learn a rich function approximation. Moreover, by using a less wide neural network for the less complex environment Push (256) it helps prevent the agent from over-fitting. We also necessitate that the width of the value network is double the width of the policy network in line with conventional practice found in code bases [52].

In order to increase the depth of the neural network while avoiding the issues such as a loss of mutual information [53] or vanishing gradients [54] (that are associated with deeper neural architecture) we use skip connections [55]. We extend [55] for GCRL and concatenate the state-action-goal instead of just the state-action to each intermediate layer.

Inspired by this application of a recent advancement in deep learning to RL, we also added a dropout layer to the actor and critic. This is intended to help increase the generalisation power through preventing the neural networks from over-fitting.

Whilst multi-layer perceptrons (MLPs) are standard practice in fully observable environments, in a partially observed environment it is more advantageous to have a state representation that encodes its state-action trajectory in addition to its current observation (which an MLP struggles to do). To endow the agent with this temporal ability we use a Long Short Term Memory (LSTM) network in the neural network for the actor and critic. This helps the agent learn a meaningful state representation that can integrate information through time - a crucial property that overcomes partial observability by reasoning, using previous experience, about the missing information.

To the best of our knowledge, there is no literature whereby an LSTM has been applied in the continuous control environment (only the discrete domain [56] [57]). This leaves ambiguity relating to the best way to update the hidden state of the LSTM. Therefore, we implemented four different update strategies and look at finding the best for a continuous control environment. Namely bootstrapped sequential updates [56], random updates [56], burn-in strategy [57] and a stored-state strategy [57].

Bootstrapped sequential updates propagates the hidden state to the end of the episode which is disadvantageous as it destabilises training because of the higher variance of the network updates brought about by the highly correlated nature of the states in a trajectory. Conversely, random updates randomly zeros the LSTM hidden state at some point in the episode which produces more de-correlated sampling which is imperative for robust optimization of neural networks. However the recurrent state is mismatched to the experience which inhibits the LSTM's ability to exploit long temporal correlations. Both these strategies select a random hidden start state to initialise the network. Conversely in the burn-in strategy a portion of the replay experience is used to recover a more meaningful initial state which, once found, is used to initialise the network and then propagated to the end of the episode. In a stored-state strategy the recurrent state is stored in the experience replay and then used to initialise the hidden state and then propagated through the episode. Storing the recurrent state makes the recurrent state more representational than a random start state. This helps the network learn long temporal correlations as the LSTM does not have to learn to recover the hidden state. Moreover, by using relatively short sequences, this helps ensure robust optimization of the neural network (something the burn-in strategy is not immune to because of its longer sequences). However, [57] notes that the stored strategy may be victim to 'representation drift' which leads to 'recurrent state staleness' as the older experiences will store older recurrent states leading to a destructive recurrent state.

To stabilise training, the last layer in the MLP network parameterising the actor was initialized with smaller weights (than the other layers) which signifies the action distribution being centered around zero with a small standard deviation. This has been shown to have quite a significant and unexpected impact on performance [58].

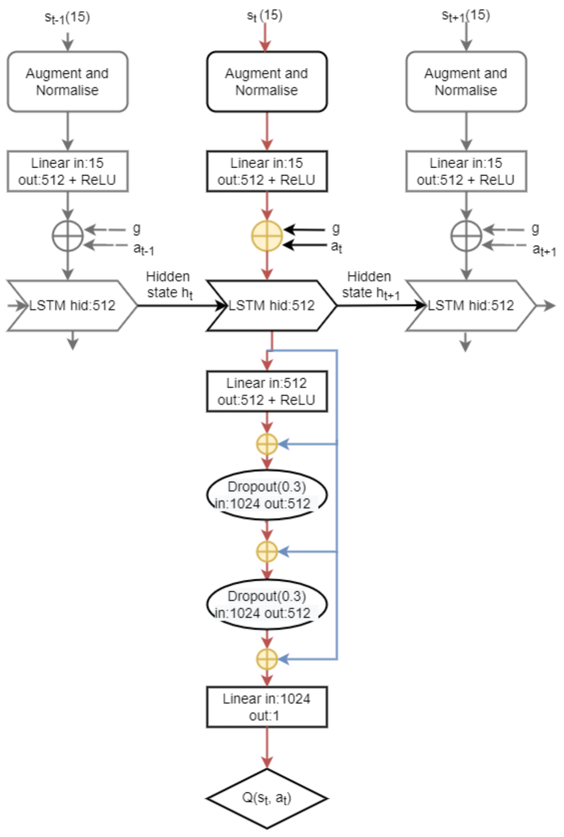


Fig. 5. Architecture of the critic. The orange denotes the tensor concatenation operation. Skip connections are shown in blue. The input state s_t, action a_t, goal g are all sampled from the Hindsight Experience Reply Buffer (see 3.5.2). in purple,

3.5 General Practice

We now outline the different general state of the art RL techniques that are not specifically related to the agent. These have been included to help create a more proficient agent.

3.5.1 Data Augmentation

Data augmentation (DA) is the process of applying random (but realistic) transformations to increase the diversity of the data set. It increases both the sample-efficiency (as we re-use the same experience but slightly transformed) and the generalisation power of the RL agent (as it learns on a more diverse data set). In our solution, we transform the states (both the agent and external environment state) that the agent is fed in four different ways (two additive and two detractive ways). We first apply random amplitude scaling [59] where the states are multiplied by $z \sim Uni[\alpha, \beta]$ (for $\alpha > 1, \beta > 1$) and additive Gaussian noise [59] where $z \sim N(0, 1)$ is added to the states. We also propose two novel detractive state augmentations (akin to cropping in image augmentations). Namely, random amplitude division whereby the states are divided by $z \sim Uni[\rho, \lambda]$ (for $\rho > 1, \lambda > 1$) and subtractive Gaussian noise where $z \sim N(0, 1)$ is subtracted from the states. These two novel augmentations are important for different reasons. Firstly, a more diverse augmentation pipeline is important to account for and represent the sensor noise the physical robot will experience which can be both additive and reductive due to effects like dampening. Therefore, these augmentations help with respect to a goal of this project which is to create a solution that is deployable to a physical robot. Moreover, in computer vision a maximally diverse augmentation pipeline has been shown to be the best to prevent overfitting [60] and it is likely that the same applies to RL.

We apply these augmentations in a novel way by selecting one at random and applying it to both the agent and environment state with a probability specified by a hyper-parameter we call $\delta \in [0, 1]$ that increases across the course of training to help prevent over-fitting. We found in our initial agents that if δ was high at the beginning of training the agent was not yet competent enough to learn from this signal and so struggles to learn. The agent and environment state is then mapped to an observation using the environment mapping O. We note that when trialing this idea we found that applying more than one augmentation or always applying an augmentation (so $\delta = 1$) distorted the observation too much and the agent would struggle to learn.

3.5.2 Experience Replay Buffer

An experience replay buffer is a finite sized cache that stores transitions and uses a first in first out strategy. Using a replay buffer is advantageous for two main reasons. Firstly, optimisation algorithms usually assume that samples adhere to the independently and identically distributed principle. This assumption is broken when samples are generated from sequentially exploring an environment. A replay buffer breaks these temporal correlations by blending older and recent experience for the networks to update. Secondly, a replay buffer helps ensure that rare experiences are repeatedly seen by reusing an experience multiple times.

In order to increase the sample efficiency of the agent, we use a Hindsight Experience Replay (HER) [37] buffer which aims to maximize the amount the agent learns from each experience by re-framing each experience with respect to different goals. Intuitively, HER substitutes the intended goal for an arbitrary goal in the replay buffer and then recomputes the reward to make a new sample. This means that the RL algorithm has a consistent learning signal. This allows the agent to learn how to achieve arbitrary goals, which includes the goals we are interested in.

3.5.3 Normalisation and Clipping

We now discuss different normalisation and clipping techniques we have implemented. We highlight these choices as they can have a significant impact on performance [58] and are rarely discussed in literature (which contributes to the lack of reproducibility in RL).

To increase the robustness and stability of our solution, we normalise the observations before they are fed to the agent. To do this the empirical mean μ and standard deviation o_ρ of the (not normalised) observations are calculated. We then normalise the observations by subtracting this empirical mean and dividing by max(o_ρ, 10^6). This is important as it means the inputs fed to the neural networks have a standard deviation approximately equal to 1 and a mean approximately equal to 0. This helps the optimization of neural networks since the hidden activation functions do not saturate too fast and so do not give near zero gradients early on in learning.

We also clip the observations. Clipping observations, whilst having no effect on performance, is a valuable practice as there is a risk of mechanical malfunction (for a physical robot) which could be manifested in observations with an unusually high magnitude. By clipping the observations it helps prevent the agent learning from this erroneous signal. Moreover, there will likely be variance between different physical robots (even of the same type) therefore this helps standardise the observations so the agent can generalise to different physical robots. Therefore, this technique is a practical inclusion for deployment in industry.

We also clip the gradient in order to stabilise learning. Gradient clipping helps stabilise the learning process as it forces the optimizer to make only small changes in the backward step. Alternatively, we tried using a smaller learning rate. However, a smaller learning rate decreased the pace of learning whilst gradient clipping allows the agent to learn as usual but protects it from large detours.

Moreover, to limit the scale of the error derivatives we transform the rewards. Rewards are commonly clipped $[-1, 1]$ which makes learning more stable however it also changes the goal of the agent as it cannot differentiate between rewards of different magnitude. An alternative strategy, which the technique we have chosen to implement, is to re-scale the reward to be between $[-1, 0]$ (by normalising and then subtracting one). Scaling the reward to be between $[-1, 0]$ instead of $[0, 1]$ heuristically improves the stability of the solution [1].

3.6 Tools, Testing, Verification and Validation

The OpenAI Fetch environments are based upon the Mujoco physics simulator which took us several weeks to install on the NCC because of difficulties relating to a lack of documentation on system requirements and dependencies. However since this, Mujoco has been acquired by DeepMind and there has been an increase in support and documentation which will help users installing this in the future. We use Mujoco 2.10.

We decided to use Tensorflow 1.14 which is optimised and well supported. This is because the implementation of the Mutual Information Neural Estimator (which we built the rest of our code on) was implemented in Tensorflow 1. We spent several weeks trying to migrate this code to Tensorflow 2 however we found that there was a lack of backward compatibility and often crucial features become deprecated (such as tf.placeholder) that were very useful in the project. To perform stochastic gradient descent we used the Adam optimization algorithm which updates the network's parameters using back propagation. When updating the normal or target networks we kept the other network parameters fixed by halting gradient flow.

In order to tune the hyper-parameters, a parameter sweep was conducted using the Python library Optuna [61]. Optuna is an automatic hyper-parameter optimising framework that uses a Bayesian optimization algorithm called Tree-structured Parzen Estimator Agents to identify the most promising hyper-parameters. 250 trials were conducted for each environment. Moreover, when testing the different proposed model components, all improvements were verified by comparing the performance across different random seeds for the same number of epochs. To compare the performance we looked at the maximum mean score (across 100 episodes with the agent in evaluation mode so that it was acting optimally) at each epoch. To verify that the agents were indeed improving, we plotted the training to verify that the gradient was non zero and compared it to an agent that acted randomly. To validate the improvements of our components the agent's performance was compared to an agent trained on extrinsic rewards and a baseline off-policy agent trained on the same intrinsic rewards to verify its any improvements made.

4 Results

Our final agent was evaluated across all four Fetch environments. We only choose to use the Reach task when evaluating generalisation power as it is a very easy benchmark to solve [37] and so not significant in terms of performance. In line with our deliverables, we class the results into results relating to performance, generalisation power and insights into intrinsic rewards behaviour. In the performance and generalisation power sections we perform ablation studies to meet the knowledge and understanding deliverables. We present both graphical (which can be found on page 16 and 17) and numerical results. For each experiment, the necessary data was gathered using a sequential-sampling strategy. The experiment was first run on two random seeds that we fixed across all experiments (23, 687). If there was a large disparity between these two runs, then the experiment was run again on two more seeds (38, 493). However, for almost all of the experiments there was very little disparity so the runs were very rarely re-ran. We then took the mean of all of the runs taken. We distributed the training across 32 NVIDIA CPUs using the distributional library MPI to distribute the training. None of this research would have been possible without the NCC as we clocked up ≈ 6100 CPU hours (the majority of which can be attributed to parameter sweeps).

To graphically evaluate the agent we have plotted the mean task success rate against the number of epochs. To do this, at each epoch we run 100 episodes with the agent set in evaluation mode. In evaluation mode the agent is set to act as optimally as possible which means that no noise is added to any of the chosen actions. To calculate the mean task success rate we then sum the number of episodes where the task has been successfully completed (the goal has been met) and divide by 100. For all tasks we have chosen to train the agent for 50 epochs and evaluate performance at every epoch. Across all tasks, this was the best number of epochs to strike a balance between over-fitting and under-fitting whilst ensuring the agent has converged. This can be seen in the figures (on page 16 and 17) as across environments the gradient is largely flat by the 50^{th} epoch. In order to be able to compare across Fetch environments we kept the number of epochs the same across the environments.

In order to help assess our performance deliverables we also evaluate the agent numerically to see to what degree the agent is stable, competent and sample efficient. To assess stability we find the variance of mean task success rate once the agent is within 10% of its maximum task success. To access proficiency we use the maximum mean task success rate reached. Finally, to assess sample efficiency, we compute the average gradient of the graph (up to the point where 95% of the maximum score has been reached). The gradient is a good indicator of sample efficiency as it indicates how many interactions with the environment are needed to achieve a certain increase in performance. This data is encoded in the data used to plot the graphs however by explicitly presenting it, it helps make the agents' performance more explicit and allows us to draw stronger comparisons.

4.1 Performance

We first present results on the different off-policy and recurrent hidden state update strategies. We then present the results of our final agent in terms of performance and verify the value of each component added through an ablation study.

Our first set of results are using the different off-policy base algorithms (where every other component is enabled with the same hyper-parameters) as can be seen in Fig 9, Fig 10, Fig 11 on page 16. These graphs show that stochastic policies (TQC, SAC) are able to learn from the predominately intrinsic reward signal best and perform best on the Fetch environment. As TQC reaches the highest mean success rate and converges in the lowest number of epochs across the different Fetch environments, it is evidently the most suitable off-policy algorithm surveyed for the Fetch environments. We have chosen to keep this off-policy algorithm for all experiments going forward. From here all agents keep the same hyper-parameters. Before this, each off-policy algorithm uses different hyper-parameters and so it was not possible to have the same hyper-parameters. The hyper-parameters were found by taking the average across the best hyper-parameters found for each environment (so we use the same hyper-parameters across environments so we have one agent). The best hyper-parameters for each environment were found by the hyper-parameter tuning library Optuna as discussed in 3.6. The parameters were similar across the environments which indicates our agents robustness to hyper-parameters.

In order to determine whether the update strategy for the LSTM hidden state matters, we changed the strategy and kept everything else in the agent the same (with the additional components). Fig 12, Fig 13, Fig 14 indicate it does matter, with a stored-state strategy converging in the smallest number of epochs to the highest mean success rate across all tasks. Therefore, for the final agent we chose this strategy. This result is in contrast to the result found in [57] where [57] notes that the stored strategy may be victim to representation drift which leads to recurrent state staleness as the older and no longer relevant recurrent states are being used to update the model. We theorize our results are different because our sample buffer is smaller than the sample buffer they use (a result from the parameter sweep) therefore older experiences are more likely to be discarded and so the recurrent state is generally kept quite recent. We verified this in practice as increasing the buffer size did indeed erode performance.

These results feed into the results on the final agent. For comparison, we have compared our solution (as seen in Fig 6, Fig 7, Fig 8, Fig 15, Fig 16, Fig 17) to a basic TQC agent (with none of the additional components) trained on extrinsic rewards and a baseline TQC agent that is trained on the same reward signal as our agent however it has none of the additional performance features (we call this agent Intrinsic). The final agent (which we call Deep Ensembled Truncated Quantile Critic with Recurrent Networks which we shorted to DETRN+) is the most sample efficient, and competent agent of the approaches surveyed as can be seen in Fig 6, Fig 8, Fig 15, Fig 16, Fig 17. Whilst the extrinsic agent is the most stable (as seen in Fig 7) this is because it is unable to learn so its negligible increase in its performance is reflected in its low variance. We demonstrate a substantial increase in performance in maximum mean task success upon an agent trained on extrinsic rewards and a 51% increase upon the MUSIC baseline. The agent is proficient at Push however it achieves a lower mean task success rate for Slide and PickAndPlace which speaks to the difficulty of these environments. We discuss reasons for this in section 5.

	DETRN+	MUSIC Baseline	Extrinsic Agent
PickAndPlace	**0.46**	0.30	0.03
Push	**0.98**	0.67	0.06
Slide	**0.37**	0.24	0.02

Fig. 6. Maximum mean task success rate

	DETRN+	MUSIC Baseline	Extrinsic Agent
PickAndPlace	1.66×10^{-4}	6.79×10^{-4}	$\mathbf{1.61 \times 10^{-6}}$
Push	1.22×10^{-3}	2.21×10^{-3}	$\mathbf{4.89 \times 10^{-6}}$
Slide	5.73×10^{-5}	1.01×10^{-4}	$\mathbf{5.30 \times 10^{-7}}$

Fig. 7. Variance of mean task success rate once the agent is within 10% of its maximum mean task success rate.

	DETRN+	MUSIC Baseline	Extrinsic Agent
PickAndPlace	$\mathbf{1.14 \times 10^{-2}}$	6.90×10^{-3}	6.52×10^{-4}
Push	$\mathbf{2.71 \times 10^{-2}}$	1.80×10^{-2}	1.26×10^{-3}
Slide	$\mathbf{8.93 \times 10^{-3}}$	5.34×10^{-3}	5.98×10^{-4}

Fig. 8. Sample Efficiency. Average gradient of the graphs in Fig 15, Fig 16, Fig 17 (up to the point where 95% of the maximum score has been reached

To verify and compare the contribution of each component of the agent and ensure none of them are orthogonal or limiting the agent, we have performed an ablation study whereby the agent is kept the same except one component is removed as seen in Fig 18, Fig 19 and Fig 20. These graphs indicate the value of skip connections as when these are removed the agent achieves a lower mean task success rate. This loss of performance is caused by the loss of mutual information between the input and final output as a result of the non-linear transformations [55] in the neural network. This leads to the true representation of the input (for example the state) becomes eroded in later layers making the later layers largely uninformed of the input. However, the addition of skip connections clearly resolves this issue by concatenating the input to the hidden layers in the network. The skip connections helps the neural networks exploit the benefit of deeper neural architecture such as the ability to learn more complex and expressive functions (seen through the higher task success rate) or a smoother loss landscape (seen through the more stable convergence at the end of the graph). Fig 18, Fig 19 and Fig 20 also show that the dropout layers do not appear to affect performance in any meaningful way, however as we will show later, they are very valuable in helping the generalization power of the agent. These graphs also indicate, the use of an LSTM appears to be quite critical, especially in the PickAndPlace task as without it the agent's maximum mean task success rate is lowered to 0.39. As the PickAndPlace has the most complex temporal dynamics (as it consists of completing two tasks sequentially), the LSTM is evidently important in learning these temporal dynamics. These graphs also show that DA (data augmentation) and HER appear to help the sample efficiency of the agent in all environments as when these features are removed the average gradient becomes shallower. They also show that ensemble learning also appears to help the agent as the agent achieves a lower maximum mean task rate when this component is removed.

4.2 Generalization Power

To test the generalization power of the agent we look at how well an agent trained on the PickAndPlace task can generalize to the Reach task. In order to successfully complete the PickAndPlace task the agent has to first pick up a box and then move it to a specified position. Conversely in the Reach task the agent has to move its gripper to a specific location which is similar to the second task in PickAndPlace. To test how well the agent is able to generalize, the agent is trained on the PickAndPlace environment for 50 epochs and then trained on the Reach task for 50 epochs. We have then graphed the mean success (across 100 trials in evaluation mode) of the agent across 50 epochs (as seen in Fig 21, Fig 22, Fig 23). This gives an indicate of how well the agent is able to use the skills it has learnt in an environment with a new goal space.

We compare an agent pre-trained on PickAndPlace and an untrained agent. We do this for both an agent trained on solely extrinsic rewards and extrinsic rewards supplemented by intrinsic rewards (our reward signal). In order to be able to draw comparisons between differences caused by the different reward signals, we use the same agent for both of these tasks - our agent DETRN+ without the recurrent network because it erodes the generalisation power of the agent as discussed later. Fig 21 demonstrates that the pre-trained agent that learnt from intrinsic reward achieves the highest maximum mean success rate on Reach and does so in the smallest number of epochs. Moreover, in Fig 21 the untrained agent that learnt from extrinsic rewards achieves higher mean success rates than the pre-trained agent that learnt from extrinsic rewards. This indicates that the pre-trained agent that learnt from extrinsic rewards was not able to transfer any of its skills and instead had to unlearn and then relearn these skills for the new task. As the pre-trained agent that learnt from a predominately intrinsic reward adapts quicker in Fig 21 (achieves a higher mean success in fewer epochs) than the pre-trained agent that learnt from extrinsic rewards, we can see that intrinsic agent has learnt a more meaningful state representation that is more transferable across different tasks. The value of each state, when learning from a task-dependent extrinsic reward, is clearly tied to the task. Conversely, as intrinsic rewards are not task-specific they are very valuable in increasing the generalization power of an agent as the state representation is independent of the task (and so can be transferred across tasks).

To verify and compare the value of each component and ensure that none of the components are harming the generalisation power of our agent we perform another ablation study seen in Fig 22. Our novel contribution of applying dropout to our agent, which is usually seen as destructive and unproductive, substantially increases the generalization power of the agent helping the agent to not over-fit to an environment. This can be seen through the significantly shallower gradient of the agent in Fig 22 when the dropout layer has been removed. Fig 22 also indicates that the use of a LSTM in the agent appears to erode the generalization ability as when it is removed the agent achieves a higher maximum mean task success rate. The use of LSTM has caused the agent to over-fit to the temporal dynamics of PickAndPlace (where the agent is expected to perform two tasks sequentially) and so struggles on the less complex temporal dynamics (where there are no sub-tasks or sub-task ordering) in the Reach task. Fig 22 also shows that data augmentation also appears to be quite significant for the generalization power of the agent which can be seen through the lower mean task success reached when this component was removed.

To verify the value of our novel data augmentation we perform a specific study seen in Fig 23 whereby we trial using only additive, only detractive, both or no data augmentation. The most effective data augmentation strategy is a maximally diverse pipeline (both detractive and additive noise) as evident through Fig 23 with additive noise being more beneficial (achieving a higher mean task success rate) than detractive noise. However, detractive augmentations increase the generalization power (over no augmentation). Hence the reconstructive process the agent learns through its detractive augmentation is clearly valuable with respect to the generalisation power of the agent. Moreover, even though these augmentation are detractive, the information contained within the transformed signal (e.g. sign of inputs) still allow a rich learning signal.

4.3 Intrinsic Rewards

To investigate how best to exploit an intrinsic reward signal we look two different practices. Firstly, we look at the impact that the parameter γ (the discount factor) has in this environment on the intrinsic reward signal. A lower γ encourages the agent to act more greedily by prioritising immediate reward whereas a higher gamma encourages the agent to pursue reward less greedily by not significantly discounting future reward. Fig 27, Fig 28, Fig 29 (which plots the intrinsic reward that has been re-scaled to be between

-1 and 0 as discussed in section 3.5.3) all show that the behaviour learnt from a higher gamma results in a higher intrinsic reward than the behaviour learnt from a lower gamma. This indicates that an agent acting less greedily will better maximise intrinsic reward.

As a second point of practice, we also look at whether it is best to get the agent to maximise the average expected rewards ($\frac{1}{T}\sum_{t=0}^{T}\gamma^t r_t$) or the total expected rewards ($\sum_{t=0}^{T}\gamma^t r_t$). By maximising the average expected rewards it focuses the agent on learning behaviour that results in a more consistent and stable rewards. Conversely, by maximising the total expected reward it focuses the agent on attaining big payoff and does not emphasis the consistency of the reward, rather that it just arrives at some point. Across Fig 24, Fig 25, Fig 26 the behaviour of the agent that aims to maximising the average expected rewards results in higher intrinsic reward in comparison to behaviour learnt from trying to maximise the total expected reward. This indicates that the intrinsic reward received by the agent is higher when the agent is acting in a way that it thinks will results in a more consistent reward signal.

5 EVALUATION

This project aimed to meet and is assessed with respect to six deliverables (which will be tested across all the Fetch environments). The first three reflect performance, the fourth reflects the generalisation power of the agent and the final two reflects knowledge and understanding of the domain. The deliverables are:

1. Implement a stable agent. Specifically, produce an agent that always converges and does not get stuck. This can be assessed through graphing its performance. We can assess this more rigorously and numerically through calculating at its variance (see section 4 for how the variance is calculated). We expect it to have a low variance (less than 0.01). (Performance - Stability)

2. Implement a competent agent. Specifically, produce an agent that is capable of solving all four Fetch environments with a maximum mean task success rate above 91$\%$ (which is the average success rate of the current generation of robots for manipulator tasks). (Performance - Competence)

3. Implement a sample efficient agent. Specifically, produce an agent that interacts with the environment so few times so that it can be trained in under a day. It should also advance the sample efficiency of the other appropriate surveyed work which can be assessed by the sample efficiency metric presented in section 4. (Performance - Sample Efficiency)

4. Implement an agent capable of generalising well across environments - once it has adapted to the new goal space. This will be assessed through training an on PickAndPlace and testing its ability to generalise to Reach. (Generalisation Power)

5. Gain an insight into the practices that best exploit an intrinsic reward signal through understanding how best to optimise a policy trained on predominately intrinsic rewards. (Knowledge and Understanding)

6. To develop a deep understanding of RL state of the art that is applicable to this task and be able to discern the most appropriate and valuable practices and architecture. Moreover, to develop a deep understanding of the resultant agent and present the value or orthogonality of each component with respect to performance and generalisation power. This will be primarily assessed through the ablation studies (Knowledge and Understanding)

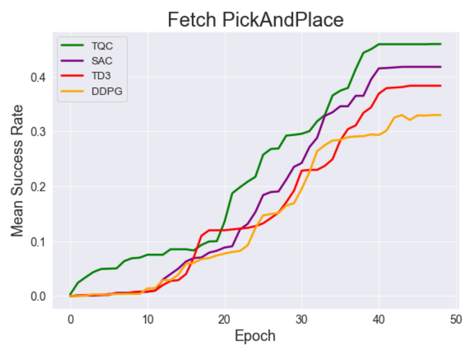
The project was successful with all but deliverable 2 being met fully met. Apart from this deliverable 1 being seen in Fig 7, Fig 15, Fig 16, Fig 17 deliverable 3 being met in Fig 8, Fig 15, Fig 16, Fig 17 deliverable 4 being met in Fig 21, deliverable 5 in Fig 24, Fig 25, Fig 26, Fig 27, Fig 28, Fig 29 and deliverable 6 met in Fig 18, Fig 19, Fig 20 and Fig 22.

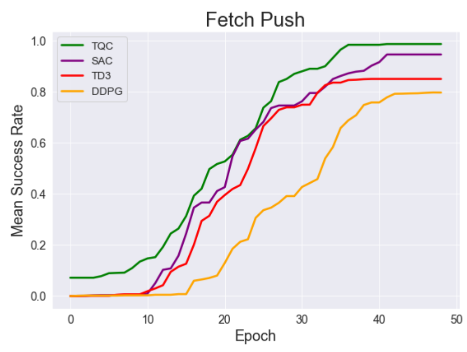
Deliverable 3 was successfully as the longest end to end training time was 14 hours and 32 minutes which is a feasible training time for deployment in industry. Moreover, it is the most sample efficient agent as seen in Fig 8 out of the other surveyed approaches. This can also be seen in Fig 15, Fig 16 and Fig 17 as our agent achieves the highest performance in 50 epochs than any of the other approaches. We attribute this to the data augmentation scheme and the HER buffer. This is because both of these re-purpose environment interactions which reduces the number of times the agent interacts with the environment.

Deliverable 4 was fully met as seen in Fig 21. This was a very important deliverable as the primary advantage of an RL approach is the flexibility of the solution. Therefore the delivery and success of the generalisation power of the agent is a significant and promising result. In Fig 21 we can see that the agent's ability in the PickAndPlace task is transferred (one it has learnt the new goal space) as it learns quicker than the untrained agent. Moreover, we demonstrated that intrinsic rewards help the agent to learn a more meaningful state representation that is better transferred between tasks. This is evident form Fig 21 where the agent trained only on extrinsic rewards had to unlearn and then learn a new state representation. Moreover, the ablation study (Fig 22) demonstrated the significance of certain components specifically the power of a dropout layer and the data augmentation scheme with respect to generalisation power.

Deliverable 5 was interesting as, to the best of our knowledge, this question has not been focused on in literature. We made the observation that the intrinsic rewards are best maximised when the agent behave less greedily (as seen in Fig 27, Fig 28, Fig 29) and it aims for a consistent intrinsic reward instead a high payoff (as seen in Fig 24, Fig 25, Fig 26). These can be done by maximising the average reward and setting the gamma coefficient to be high. We also demonstrated that in the Fetch environment intrinsic rewards are best exploited by a stochastic agent perhaps because of the variability of the intrinsic rewards (as seen in Fig 1, Fig 2, Fig 3).

For this project we gained an in depth understand of the relevant SOTA by reviewing a significant amount of literature. We also gained a deep understanding of the resultant agent as we found that all included components were largely complementary (as seen in Fig 18, Fig 19, Fig 20) with exception of R2D2 for generalisation power (Fig 22) indicating that in general RL does not need to focus on ensuring SOTA advances are compatible to other SOTA. The ablation studies (Fig 23) allowed us to establish the value of our novel data augmentation scheme as well the





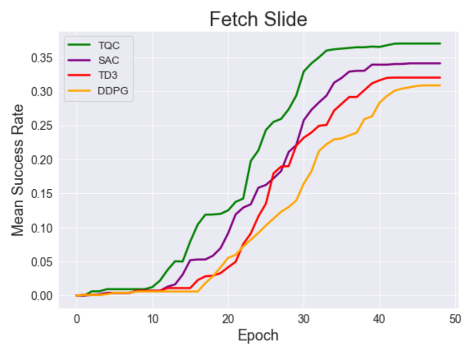
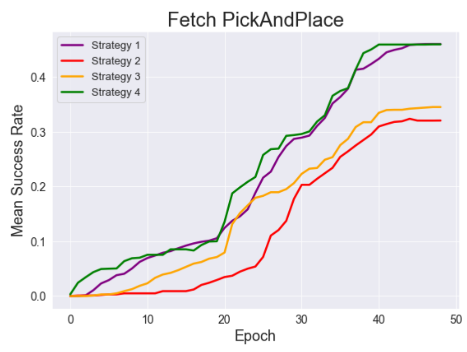
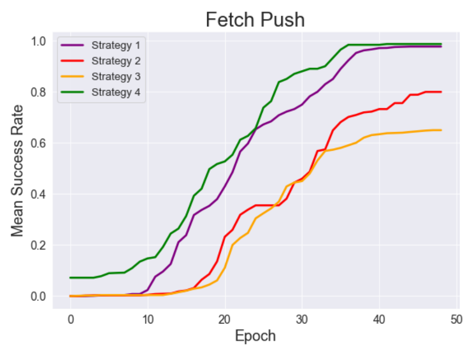


Fig 9, Fig 10, Fig 11 showing mean success rate against epochs for different off-policy agents on PickAndPlace, Push and Slide





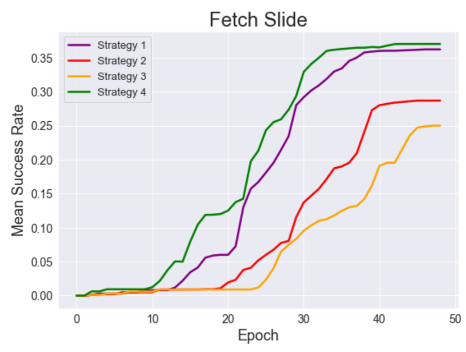
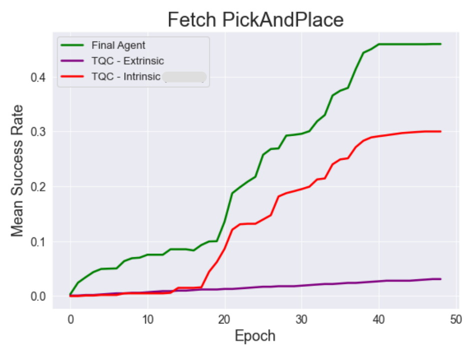
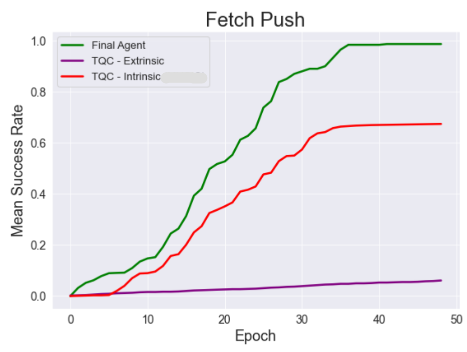


Fig 12, Fig 13, Fig 14 showing mean success rate against epochs for different recurrent strategies (where Strategy 1 is stored-state, Strategy 2 is burn-in, Strategy 3 is bootstrapped and Strategy 4 is random updates) on PickAndPlace, Push and Slide





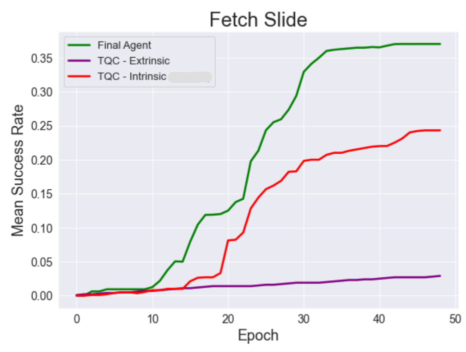
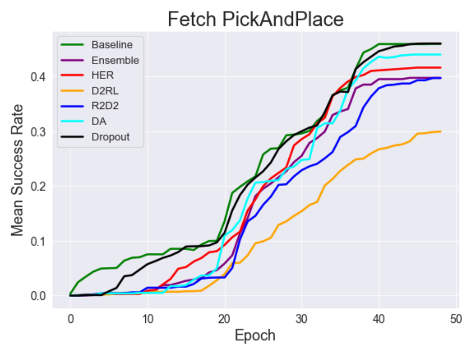
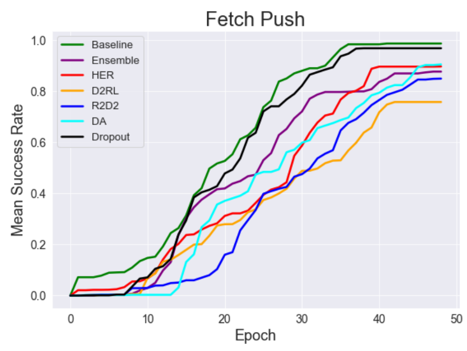


Fig 15, Fig 16, Fig 17 showing the mean success rate against epochs for different agents on PickAndPlace, Push and Slide





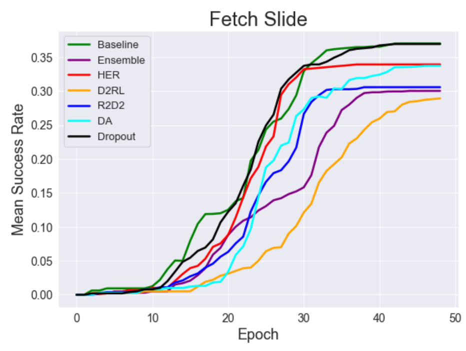
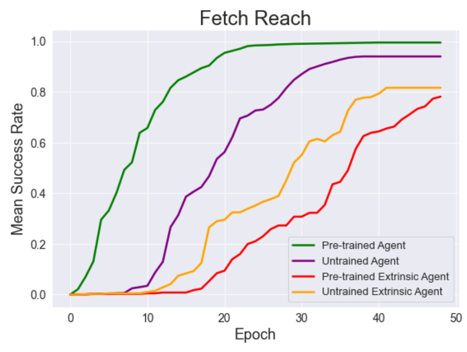
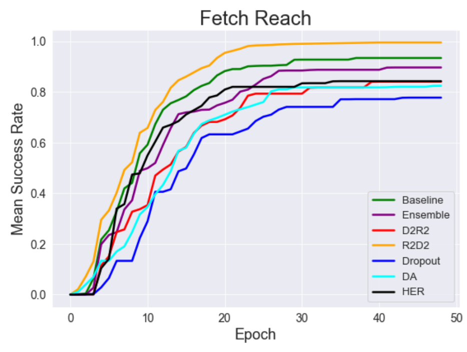


Fig 18, Fig 19, Fig 20 showing the mean success rate against epoch for ablation studies on PickAndPlace, Push and Slide





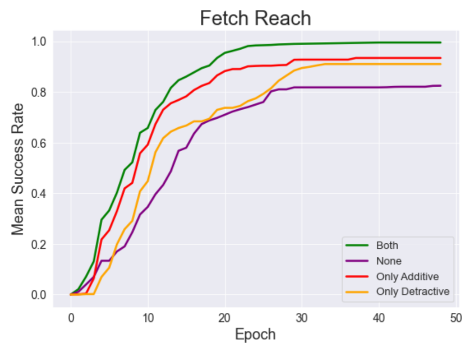
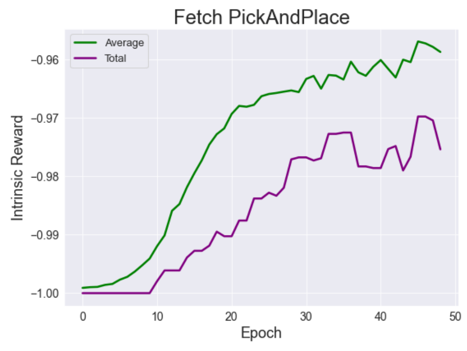
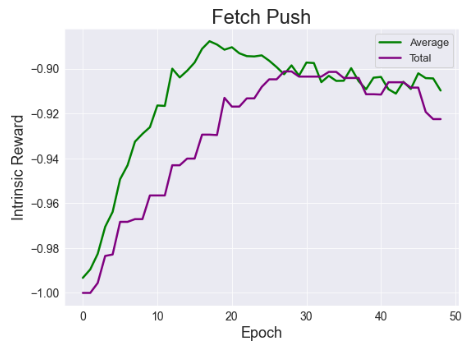


Fig 21 showing the mean task success against epochs for different agents on Reach, Fig 22 showing mean success rate against epoch for an ablation study with all components on Reach, Fig 23 showing mean success rate against epoch for an ablation study with just state augmentation techniques on Reach





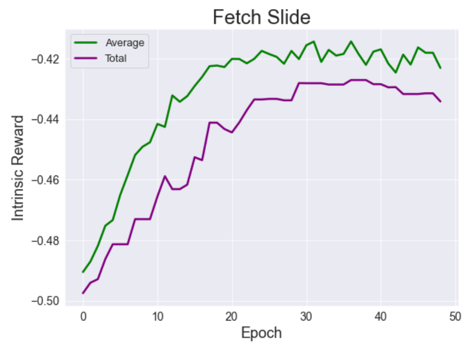
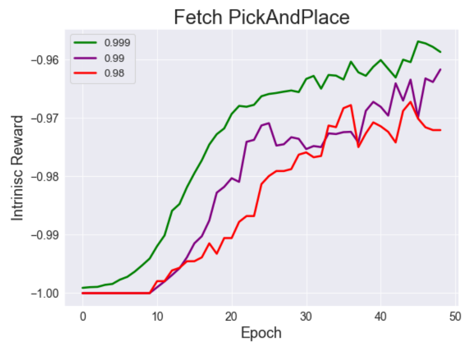
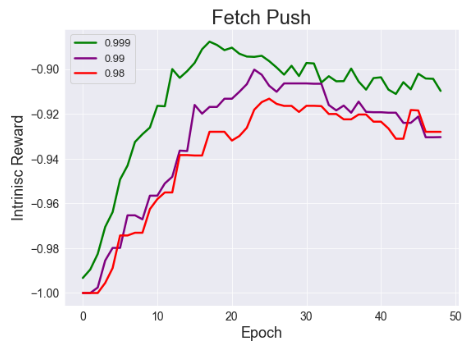


Fig 24, Fig 25, Fig 26 showing the re-scaled (to be between -1 and 0) intrinsic reward on PickAndPlace, Push and Slide for maximising either the total rewards or the average rewards





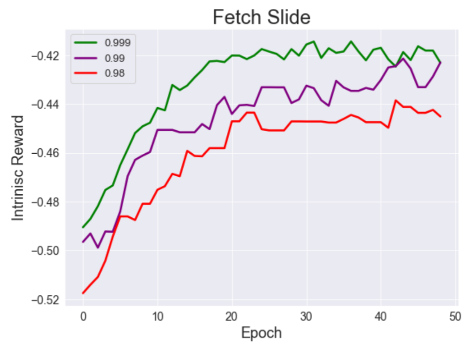


Fig 27, Fig 28, Fig 29 showing the re-scaled (to be between -1 and 0) intrinsic reward for different gamma values on PickAndPlace, Push and Slide

value of dropout (Fig 22). And in general proved to be very valuable in our understanding of the resultant agent. Therefore deliverable 6 was successfully met.

Deliverable 1 can be seen through Fig 15, Fig 16, Fig 17 and was successful as the agent always converges and does not get stuck. Moreover as evident in Fig 7 it has a very low variance. It is not the lowest because of the extrinsic agents lack of ability to learn which is reflected in its low variance. This was a particularly challenging deliverable because the reward function was not purposely engineered to help guide the policy optimisation therefore the agent demonstrates an impressive robustness to the variability of the reward signal. This indicates that resultant agent is robust which is significant as a lot of SOTA RL has been shown to be brittle.

Unfortunately we were not able to fully meet deliverable 2 as we only exceeded a maximum mean task success rate of 91% in Reach and Push. This demonstrates the difficulty of the Slide and PickAndPlace as none of the surveyed approaches were able to achieve the desired level of success. We partly attribute this lack of success to a limitation in the way in which we combined the intrinsic and extrinsic rewards. This is because for the Slide and PickAndPlace the agent is capable of achieving the task sometimes however it is not capable of completing the task consistently. We assert that this is because the agent is not encouraged to pursue developing consistency but is instead focused on still exploring and learning to control the environment (as motivated by the intrinsic reward). This happens because the intrinsic reward is more frequently received and so makes up the majority of rewards the agent is encouraged to maximise. In the beginning this is helpful as it encourages the agent to explore and gain an understanding of its environment. However, it does not place any emphasise on prioritising any specific part of the environment, rather placing emphasis on acquiring an understanding of the dynamics of the whole environment. Hence the intrinsic reward is not good at prioritising the specific parts of the environments needed for these two environments.

To mitigate this issue we propose to change the way the intrinsic and extrinsic rewards are combined. In this paper, we simply add the scaled rewards with the parameter θ. However, we could try and introduce a scheme for the hyper-parameter θ similar to how the alpha hyper-parameter is adjusted in SAC [43]. Here we would take a gradient step with respect the parameter θ so that θ was adapted based on whether the extrinsic task-dependent reward or the intrinsic task-independent needed to be prioritised. We would expect that initially the intrinsic reward is needed more to produce a knowledgeable agent capable of controlling the whole environment, with the extrinsic being needed later to prioritise the relevant parts of the environment. However coming up with a suitable loss function for this might be challenging. Alternatively we could have tried an annealing scheme with the intrinsic reward annealing overtime. To verify that the lack of success with respect to deliverable 2 was indeed caused by the intrinsic reward being too strong we would need to implement the schemes above.

The primary limitations of this project are related to time and resource constraints. Unfortunately, we were unable to run the experiments across as many random seeds as an academic study would aim for. However this was partially remedied by our sequential sampling scheme. Moreover, our results and claims were found using one intrinsic reward signal and one robotic simulation. For statistical significance, to confidently assert the claims and results of this paper we would need to experiment with other intrinsic reward signals and other robotic simulations. Furthermore, whilst we distributed our agent across 32 CPUs, it is likely that the performance could have been augmented further by distributing across several GPUs instead. We were limited in the CPU setup because the Fetch environments are computationally expensive (time consuming) to render. Therefore we chose to render the results rarely which limited our ability to gain an intuitive insight into the agents behaviour. It is likely that a GPU would have sped up this rendering process so it would be more feasible to render the results more frequently to gain insights into the agents behaviour. We also could have changed the setup such that learning and acting happened in parallel instead of sequentially (as in our current set up) by having one learning and several actors. This would allow the learning process to converge quicker and has been shown to produce a more proficient agent [62]. Finally, whilst we put a lot of effort into making sure the solution would also work on a physical robot (so it had practical value), the only way to verify this would be to test the agent on a physical robot which is unfortunately not possible.

6 Conclusion

This project has successfully created a skillful agent, able to teach itself to perform various manipulation tasks to varying degrees of success with no prior knowledge of the environment or any existing skills. It achieves this by supplementing the sparse task reward with an intrinsic task independent reward that it computes itself. The agent is a combination of novel contributions and recent advances in RL that have never been combined or applied to the Fetch environments. The resultant agent surpasses the performance of all surveyed approaches. It also is able to generalise across environments (once it has adapted to the new goal space). We were also able to gain a deep understanding of the resultant agent and the value of each contribution in relation to performance and generalisation power showing that all included components, with the exception of recurrent networks on generalisation power, were complimentary. Furthermore, in this paper we also gain some insight in how learning with a task-independent intrinsic reward signal differs from a task-specific extrinsic reward signal finding that it is best to encourage less greedy behaviour and rather promote consistent reward instead of large payoffs (as seen with task dependent reward) and use stochastic agents. Moreover, we demonstrate that intrinsic rewards learn a more meaningful state representation than can be transferred across tasks when compared to the state representation learnt by a purely extrinsic agent.

In this paper we make a series of novel contributions which we summarise in order of importance. Firstly, we have shown the value that dropout layers, widely considered to be harmful in RL, have in increasing the generalisation power of an agent without eroding the agents performance. We also demonstrate the promise of the practice of state augmentation showing that both additive and our novel detractive scheme are valuable however a maximally diverse scheme is the best. Our novel annealing probability scheme was also a factor in the impressive performance of the state augmentation. We found that in order to best leverage an intrinsic reward signal, it is best to get the agent focus on maintaining a consistent reward function instead of focusing on large payoffs. Finally, we have also found that it is better to act less greedily (than is standard for extrinsic task-dependent rewards) to maximise intrinsic rewards. We have shown that stochastic (instead of deterministic) agents are most appropriate for exploiting an intrinsic reward signal. We have also shown that the value recurrent networks have in partially observable environments in discrete domain is also transferable to partially observable continuous control environment. However, we found that erode the generalisation power of the agent as they over-fit to the temporal dynamics. Apart from this, we have shown the components of the agent are additive and that no orthogonality exists between the components which is reassuring as it does not suggest that different state of the art practices undermine each other.

As a point of further work we could look at different methods of prioritising different samples in the HER buffer [63]. This would help the agent learn from the most informative samples thereby further augmenting the sample efficiency of the agent. Furthermore, in our paper the agent learns from a vector input that describes the state. We could look at extending our solution to learn from images instead of a vector input. We initially chose to focus on vector inputs as images can easily be distorted (for example due to different light levels) and it is easier to keep inputs consistent. However, in some situations it may be easier to feed the agent an image than describing its state as a vector (for example it is difficult to compute its angular velocity as a component of the state). Furthermore, as discussed above we could look at distributing our agent further to augment performance. Finally, we could look as the value of replacing the LSTM with a transformer as this has been shown to increase the stability and performance in a POMDP environment [64].

REFERENCES

[1] Matthias Plappert et al. “Multi-goal reinforcement learning: Challenging robotics environments and request for research”. In: *arXiv preprint arXiv:1802.09464* (2018).

[2] Greg Brockman et al. “Openai gym”. In: *arXiv preprint arXiv:1606.01540* (2016).

[3] Athanasios S Polydoros and Lazaros Nalpantidis. “Survey of model-based reinforcement learning: Applications on robotics”. In: *Journal of Intelligent & Robotic Systems* 86.2 (2017), pp. 153–173.

[4] Marc Deisenroth and Carl E Rasmussen. “PILCO: A model-based and data-efficient approach to policy search”. In: *Proceedings of the 28th International Conference on machine learning (ICML-11)*. Citeseer. 2011, pp. 465–472.

[5] Vikash Kumar, Emanuel Todorov, and Sergey Levine. “Optimal control with learned local models: Application to dexterous manipulation”. In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2016, pp. 378–383.

[6] Chelsea Finn and Sergey Levine. “Deep visual foresight for planning robot motion”. In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2017, pp. 2786–2793.

[7] Marc Peter Deisenroth, Carl Edward Rasmussen, and Dieter Fox. “Learning to control a low-cost manipulator using data-efficient reinforcement learning”. In: *Robotics: Science and Systems VII* 7 (2011), pp. 57–64.

[8] Jan Peters, Katharina Mulling, and Yasemin Altun. “Relative entropy policy search”. In: *Twenty-Fourth AAAI Conference on Artificial Intelligence*. 2010.

[9] Lucian Busoniu et al. *Reinforcement learning and dynamic programming using function approximators*. CRC press, 2017.

[10] Sergey Levine et al. “End-to-end training of deep visuomotor policies”. In: *The Journal of Machine Learning Research* 17.1 (2016), pp. 1334–1373.

[11] Lasse Espeholt et al. “Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures”. In: *International Conference on Machine Learning*. PMLR. 2018, pp. 1407–1416.

[12] Matteo Hessel et al. “Multi-task deep reinforcement learning with popart”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 01. 2019, pp. 3796–3803.

[13] Hado P van Hasselt et al. “Learning values across many orders of magnitude”. In: *Advances in Neural Information Processing Systems* 29 (2016).

[14] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. “A reduction of imitation learning and structured prediction to no-regret online learning”. In: *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings. 2011, pp. 627–635.

[15] Umar Syed and Robert E Schapire. “A game-theoretic approach to apprenticeship learning”. In: *Advances in neural information processing systems* 20 (2007).

[16] Umar Syed, Michael Bowling, and Robert E Schapire. “Apprenticeship learning using linear programming”. In: *Proceedings of the 25th international conference on Machine learning*. 2008, pp. 1032–1039.

[17] Chelsea Finn, Sergey Levine, and Pieter Abbeel. “Guided cost learning: Deep inverse optimal control via policy optimization”. In: *International conference on machine learning*. PMLR. 2016, pp. 49–58.

[18] Jonathan Ho and Stefano Ermon. “Generative adversarial imitation learning”. In: *Advances in neural information processing systems* 29 (2016).

[19] John Schulman et al. “Trust region policy optimization”. In: *International conference on machine learning*. PMLR. 2015, pp. 1889–1897.

[20] Brian D Ziebart et al. “Maximum entropy inverse reinforcement learning.” In: *Aaai*. Vol. 8. Chicago, IL, USA. 2008, pp. 1433–1438.

[21] Arthur Aubret, Laetitia Matignon, and Salima Hassas. “A survey on intrinsic motivation in reinforcement learning”. In: *arXiv preprint arXiv:1908.06976* (2019).

[22] David Warde-Farley et al. “Unsupervised control through non-parametric discriminative rewards”. In: *arXiv preprint arXiv:1811.11359* (2018).

[23] Karol Gregor, Danilo Jimenez Rezende, and Daan Wierstra. “Variational intrinsic control”. In: *arXiv preprint arXiv:1611.07507* (2016).

[24] Yuri Burda et al. “Large-scale study of curiosity-driven learning”. In: *arXiv preprint arXiv:1808.04355* (2018).

[25] Bradly C Stadie, Sergey Levine, and Pieter Abbeel. “Incentivizing exploration in reinforcement learning with deep predictive models”. In: *arXiv preprint arXiv:1507.00814* (2015).

[26] Deepak Pathak et al. “Curiosity-driven exploration by self-supervised prediction”. In: *International conference on machine learning*. PMLR. 2017, pp. 2778–2787.

[27] Hyoungseok Kim et al. “Emi: Exploration with mutual information”. In: *arXiv preprint arXiv:1810.01176* (2018).

[28] Ronen I Brafman and Moshe Tennenholtz. “R-max-a general polynomial time algorithm for near-optimal

reinforcement learning". In: *Journal of Machine Learning Research* 3.Oct (2002), pp. 213–231.

[29] Marc Bellemare et al. "Unifying count-based exploration and intrinsic motivation". In: *Advances in neural information processing systems* 29 (2016).

[30] Georg Ostrovski et al. "Count-based exploration with neural density models". In: *International conference on machine learning*. PMLR. 2017, pp. 2721–2730.

[31] Nal Kalchbrenner et al. "Conditional image generation with PixelCNN decoders". In: *Advances in Neural Information Processing Systems* (2016).

[32] Jarryd Martin et al. "Count-based exploration in feature space for reinforcement learning". In: *arXiv preprint arXiv:1706.08090* (2017).

[33] Shakir Mohamed and Danilo Jimenez Rezende. "Variational information maximisation for intrinsically motivated reinforcement learning". In: *Advances in neural information processing systems* 28 (2015).

[34] Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[35] Richard S Sutton. "Learning to predict by the methods of temporal differences". In: *Machine learning* 3.1 (1988), pp. 9–44.

[36] Tom Schaul et al. "Universal value function approximators". In: *International conference on machine learning*. PMLR. 2015, pp. 1312–1320.

[37] Marcin Andrychowicz et al. "Hindsight experience replay". In: *Advances in neural information processing systems* 30 (2017).

[38] Pierre-Yves Oudeyer and Frederic Kaplan. "How can we define intrinsic motivation?" In: *the 8th International Conference on Epigenetic Robotics: Modeling Cognitive Development in Robotic Systems*. Lund University Cognitive Studies, Lund: LUCS, Brighton. 2008.

[39] Rui Zhao et al. "Mutual information state intrinsic control". In: *arXiv preprint arXiv:2103.08107* (2021).

[40] Mohamed Ishmael Belghazi et al. "Mine: mutual information neural estimation". In: *arXiv preprint arXiv:1801.04062* (2018).

[41] Peter Henderson et al. "Deep reinforcement learning that matters". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 32. 1. 2018.

[42] Logan Engstrom et al. "Implementation matters in deep policy gradients: A case study on PPO and TRPO". In: *arXiv preprint arXiv:2005.12729* (2020).

[43] Tuomas Haarnoja et al. "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor". In: *International conference on machine learning*. PMLR. 2018, pp. 1861–1870.

[44] Scott Fujimoto, Herke Hoof, and David Meger. "Addressing function approximation error in actor-critic methods". In: *International Conference on Machine Learning*. PMLR. 2018, pp. 1587–1596.

[45] Timothy P Lillicrap et al. "Continuous control with deep reinforcement learning". In: *arXiv preprint arXiv:1509.02971* (2015).

[46] Arsenii Kuznetsov et al. "Controlling overestimation bias with truncated mixture of continuous distributional quantile critics". In: *International Conference on Machine Learning*. PMLR. 2020, pp. 5556–5566.

[47] Aviral Kumar et al. "Conservative q-learning for offline reinforcement learning". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 1179–1191.

[48] Kimin Lee et al. "Sunrise: A simple unified framework for ensemble learning in deep reinforcement learning". In: *International Conference on Machine Learning*. PMLR. 2021, pp. 6131–6141.

[49] Piotr Januszewski et al. "Continuous Control With Ensemble Deep Deterministic Policy Gradients". In: *arXiv preprint arXiv:2111.15382* (2021).

[50] Denis Yarats et al. "Improving sample efficiency in model-free reinforcement learning from images". In: *arXiv preprint arXiv:1910.01741* (2019).

[51] Xingyou Song et al. "Observational overfitting in reinforcement learning". In: *arXiv preprint arXiv:1912.02975* (2019).

[52] Antonin Raffin et al. *Stable baselines3*. 2019.

[53] Kaiming He et al. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[54] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[55] Samarth Sinha et al. "D2rl: Deep dense architectures in reinforcement learning". In: *arXiv preprint arXiv:2010.09163* (2020).

[56] Matthew Hausknecht and Peter Stone. "Deep recurrent q-learning for partially observable mdps". In: *2015 aaai fall symposium series*. 2015.

[57] Steven Kapturowski et al. "Recurrent experience replay in distributed reinforcement learning". In: *International conference on learning representations*. 2018.

[58] Marcin Andrychowicz et al. "What matters in on-policy reinforcement learning? a large-scale empirical study". In: *arXiv preprint arXiv:2006.05990* (2020).

[59] Misha Laskin et al. "Reinforcement learning with augmented data". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 19884–19895.

[60] Tero Karras et al. "Training generative adversarial networks with limited data". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 12104–12114.

[61] Takuya Akiba et al. "Optuna: A next-generation hyperparameter optimization framework". In: *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*. 2019, pp. 2623–2631.

[62] Dan Horgan et al. "Distributed prioritized experience replay". In: *arXiv preprint arXiv:1803.00933* (2018).

[63] Rui Zhao and Volker Tresp. "Energy-based hindsight experience prioritization". In: *Conference on Robot Learning*. PMLR. 2018, pp. 113–122.

[64] Emilio Parisotto et al. "Stabilizing transformers for reinforcement learning". In: *International Conference on Machine Learning*. PMLR. 2020, pp. 7487–7498.