University of Cape Town

Advanced Topics in Reinforcement Learning

MAM4001W

Towards Intelligent Reinforcement Learning Agents

Author
Qiulin Li

Supervisor
Associate Professor Jonathan Shock

Abstract

Reinforcement learning is one of the many field in AI that has the potential to produce intelligent agents that can generalise and perform novel tasks in a range of environments. Agents and environments in the standard Reinforcement Learning Framework, often cannot generalise or perform novel tasks. This report provides a brief overview on Intelligence from a AI perspective as well as a review of the several different Reinforcement Learning frameworks and algorithms which have contributed to the development of intelligent agents.

Contents

Chapter 1

Introduction

The study of Artificial Intelligence is primarily focused on developing machines that have the intelligence of a human (Burns, 2022) . There have been many attempts to do this in several fields of AI such as Natural Language processing, Deep learning and Computer Vision with varying degrees of success. Reinforcement learning, however, has shown promising results in creating intelligent agents due to RL algorithm's ability to adapt to different scenarios (Demir, 2023) .

Intelligence, in the reinforcement learning framework, measures an agent's ability to succeed in a wide range of environments (Legg, 2021). This definition of intelligence is formalised in equation 1.1 where V is a value function π is the agent policy and μ is the environment (Legg, 2021). What is important to note is the presence of the parameter u, which represents a Reference Universal Turing Machine that defines the environments and goals the agent is most likely to face. The definition of intelligence is therefore considered in the context of the agents goals and environments.

$$\upsilon_u(\pi) = \sum_{\mu \in E} 2^{-K_u(\mu) V_\mu^\pi} \tag{1.1}$$

If we consider an agent that only encounters tasks and environments that humans experience, (u set accordingly), equation 1.1 will define the measure the human-intelligence of an agent. This definition of human-intelligence requires the definition of environments that humans experience, otherwise known as human-like environments.

Human-like RL should reflect reality by integrating features of reality. Reinforcement learning problems are often described in an episodic settings (Brooks,2021), where the state of the agent and the environment reset once the agent accomplishes the task. This however, is not the case in reality. Humans do not experience episodic conditions, hence agents in human-like environments must learn to adapt and learn in a non-episodic setting. Reinforcement learning is applied to many real-life systems, however there are two core differences between reality and simulated RL environments (Co-Reyes et al, 2020):

1. Reality is non-static
2. Agent must learn to adapt to a changing environment without encountering any terminal states

A non-static environment is difficult for an agent to learn in since the agent can no longer learn a deterministic sequence of decisions (Han, 2018). If we take the analogy of an Easter Egg Hunt. If the eggs are hidden in the same spot every year, the child taking part in the egg hunt will eventually learn how to collect all the eggs by memorising the spots where all the eggs are located. If the eggs are in a different location each year, the child cannot go to the same spots as always to find the eggs. Agents in Dynamic Environments require substantially more training and even then, can fail to accomplish their assigned task. Agents in the standard RL framework

also operate in an episodic setting (Sutton & Barto, 2018), which means that if it performs an action that results in a terminal state, the agent can simply reset and not perform the action in the next episode. This is not the case in reality, where agents must learn to avoid terminal states at all costs since there exists no resets.

Producing RL agents that can accomplish tasks in human-like environments is vital to both the development of intelligent agents as well as applications of RL to real-life systems, however, this is difficult due to the contrasts between the standard RL framework and reality. There have been attempts to bring the two closer by creating more realistic environments for agents to train in, as well as better algorithms to help agents learn in realistic environments. In this report, we review several topics in Reinforcement Learning and how they have contributed to the development of intelligent agents.

Chapter 2

The Standard Reinforcement Learning Framework

Many of the RL topics that have contributed to developing intelligent RL agents have introduced modifications to the original RL framework. In order to understand these advancements in RL, it's essential to first comprehend the basis of RL. Reinforcement learning involves training an agent to perform a specific task in some environment. The agent learns to perform a task by maximising its' cumulative reward over time. A basic description of RL is shown in figure 2.1.

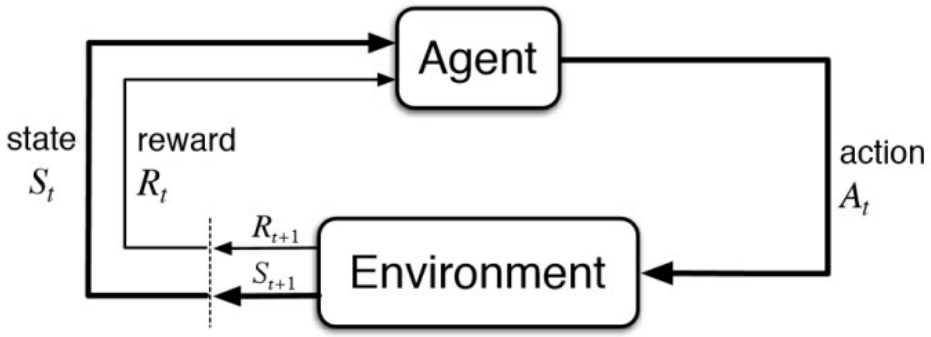

Figure 2.1: Basic RL diagram (Sutton & Barto, 2018)

The agent in the diagram above interacts with its' environments via an action A_t. The environment then returns the state of system as well as any reward r_t given for performing action A_t. The state of the system s_t represents aspects of the environment that influence the agent's actions. r_t and s_t are fed into the agent, where it can learn and update its internal knowledge which is formalised as a policy π and described as a strategy. The agent strives to improve its decision-making policy and maximize the cumulative reward (Sutton & Barto, 2018) it receives throughout the course of its interactions with the environment through iteratively interacting with the environment. This iterative loop is essential to the field of reinforcement learning since it drives the agent to making increasingly rewarding decisions in a given environment.

RL environments are systems with which the RL agent interact with to learn how to complete a task. These systems are commonly described using Markov Decision Processes (MDPs) models (Sutton & Barto, 2018), which provide a mathematical framework for decision making where the outcome is partially random and partially controlled by the decision maker (Agent in RL context). MDPs all also possess the Markovian property, which is that the future state of the system depends only on the current state and action taken by the agent (Brooks, 2021) , making all other previous states and actions irrelevant. MDPs are described as $MDP = (S, A, P_a, R_a)$ using the 4 variables shown in equation 2.1.

$$
\begin{aligned}
S &= \text{state space} \\
A &= \text{action space} \\
P_a &= P_a(s, s') = \text{ probability that action } a \text{ in state } s \text{ at time } t \text{ will lead to state } s \text{ at time } t+1 \\
R_a &= R_a(s, s') = \text{ is the reward expected for transitioning from state } s \text{ to state } s'
\end{aligned} \tag{2.1}
$$

The state and action space is the set of all possible states and actions that can occur (Sutton & Barto, 2021). The agent's objective in an MDP is to discover an optimal policy that maximizes the expected cumulative reward over time. MDPs are a fundamental to Reinforcement Learning since many RL environments are formalised using MDPs or variations of it. An RL agents must interact with its' environment in order get a potentially recieve a reward signal (Sutton & Barto, 2021). When state s_t is given as an input to the agent it is known as the agent's observation. The RL feedback loop shown in equation 2.1 shows that the agent will iteratively receive the resulting reward and state from performing some action. The agent utilises this information as well as some RL algorithm to improve the its policy. The algorithms that are used can be categorised into either value-based or policy-based methods.

Value-based methods train agents by estimating and optimizing the value function, which represents the expected cumulative reward an agent can obtain in various states (Miller, 2022). Q-learning is an example of a value-based method where the agent learns to approximate the Q-values for each state-action pair. The agent makes decisions by selecting actions with the highest Q-values (Sutton & Barto, 2021). The objective of value-based methods is to find an optimal value function will guide the agent to take actions that lead to the highest expected cumulative reward. These methods are effective for problems where the emphasis is on learning the value of actions.

Policy-based methods in reinforcement learning focuses on directly learning the optimal policy, which maps states to actions(Sutton & Barto, 2021). Policy-based approaches, such as REINFORCE parameterize the policy using some parameters often denoted θ and update them to maximize the expected cumulative reward. Policy-based methods are particularly useful when dealing with high-dimensional and continuous action spaces since they provide a flexible way to represent complex policies. Although policy-based methods are often computationally heavy compared to value-based methods (Miller, 2022), policy-based methods can handle a broader range of problems and can train agents directly from inputs without an explicit value function.

Now that we have described the standard Reinforcement learning framework, we can begin discussing modified RL frameworks that have been used to develop intelligent RL agents. Intelligent RL agents must be able to perform tasks in a range of environments. Since Intelligence in AI described in terms of human intelligence, we must develop human-like environments for the agent to interact with.

Chapter 3

Ecological Reinforcement Learning: Human-like environments

The term Ecological Reinforcement learning (Eco-RL) is a subfield in RL that focuses on how an RL agent is affected by its' environment. In particular, Co-Reyes et al (2022) focuses on how stochastic, non-episodic environments can impact an agent. Co-Reyes et al proposes two environmental properties, which when implemented, can result in dynamic non-episodic environments that can assist the agent in learning how to perform its' predefined task. These properties also reflect features of reality, therefore implementing them will produce human-like RL environments. Environments with these properties are described as Eco-RL environments in this report and are a type of Human-like environment since they represent environments that humans in reality experience.

Eco-RL Environments are dynamic and continuously change in order to reflect the stochastic nature of reality (Co-Reyes et al, 2022). It is more difficult to solve tasks in dynamic environments since the sequence of decisions taken by an agent must change each time due to the changing environments. This dynamism does however, also allow the agent to observe the system in a wider variation of states which indirectly increases agent exploration and state coverage. Dynamism in the MDP environment is represented using entropy as shown in equation **??** where$\mathcal{P}$ denotes dynamism and $\mathcal{H}$ is entropy (Co-Reyes et al, 2022). Dynamism is also particularly useful in the context of non-episodic human-like environments since non-episodic tasks mean the environment and agent cannot reset to its' initial state to explore different decision trajectories. This dynamism also provides a soft reset so the agent can explore different trajectories. This is something that commonly occurs in continuous learning where the agent can seamlessly transition from one state to another without a hard reset. Given an agent that must learn to complete a series of tasks, the agent will be able to transition from a goal state of one task to an initial state of the next task with minimal difficulty. This concept is known as **Environmental Dynamism** and was first described by Co-Reyes et al (2022) where they claimed that environmental dynamism can alleviate challenges associated with training agents under non-episodic settings.

$$\text{Environmental Dynamism} = \mathcal{H}_{\mathcal{P},\pi}\left(\mathcal{S}' \mid \mathcal{S} = s\right) \tag{3.1}$$

The Standard RL Framework described in Chapter 2 states that RL agents learn how to complete a task in their environment by maximising their cumulative reward. This method of rewarding an agent is simple, but does not perform well under many scenarios (Miller, 2022). The standard reward function often results in sparse rewards, where the agent rarely receives a reward during training. This makes training agents difficult since a sparse reward signal means there is no guidance for the agent to follow. Reward shaping was then implemented to provide more reward signals to the agent (Han,2018). It is a method of modifying the reward function such that it trains the agent by providing incremental smaller reward signals to guide the agent to learn the correct sequence of actions. There are several types of reward shaping such as Potential-based Reward shaping as

well as distance-to-goal reward shaping (Miller, 2022). Reward shaping, although useful, often requires prior knowledge of the optimal policy and can introduce a bias, resulting in a sequence of non-optimal actions being made. Another method to train an agent must thus be considered.

Co-Reyes et al (2022) consider another feature of realistic Environments that can be exploited to assist agents in learning in dynamic environments. The idea draws inspiration from the environment under which young humans learn. Humans learn by first accomplishing simple tasks in cooperative environments an children learning basic arithmetic. It is only after basic arithmetic is understood, do humans begin learning more complex concepts such as calculus and algebra. We can implement the same idea in training a RL agent by first training it in a cooperative simple environment before it is expected to complete tasks in a dynamic non-cooperative environment (Co-Reyes et al, 2022). The environment is modified by either changing the dynamics $\mathcal{P}$ or initial state distribution $p(s)$ of the MDP environment. This is a concept known as **Environmental Shaping** since the environment is shaped and simplified in some way in order to assist the agent during training. Co-Reyes et al (2022) describes Environmental Shaping as altering the initial state or dynamics of the non-episodic training MDP to make learning more tractable.

An example of environmental shaping in terms of changing dynamics $\mathcal{P}$ is the presence of cooperative worker that assist the agent accomplish its' task. An example of changing the initial state distribution $p(s)$ is the existence of an easier version of the assigned task for the agent to complete. As with any RL algorithm, our initial policy will differ drastically from the optimal policy when the reward is sparse (Miller, 2022). The mismatch coefficient that measure of dissimilarity between two probability distributions of the initial and and optimal policy (Co-Reyes et al, 2022) will be extremely high due to the difference in the policies. According to Co-Reyes et al (2022) altering $p(s)$ reduces the mismatch coefficient which will allow our agent to more easily learn the optimal policy. This is due to the fact that the agent following a suboptimal policy in the shaped environment will encounter the same states as an agent following a good policy in an unshaped environment since environmental shaping increases our agent's likelihood of encountering a useful state (Co-Reyes et al, 2022).

Implementing these properties will produce dynamic non-episodic RL environments that better reflect Human-like Environments. These properties will also assist agents in learning how to complete tasks in these dynamic environments. It should be noted that Reward shaping is not utilised in these environments and as such, the reward is sparse. Co-Reyes et al (2022) demonstrates the usefulness of these properties using four experiments shown in figure 3.1 below. Both static and dynamic versions of these environments are compared as well as the effect of reward and environment shaping in the dynamic environments.

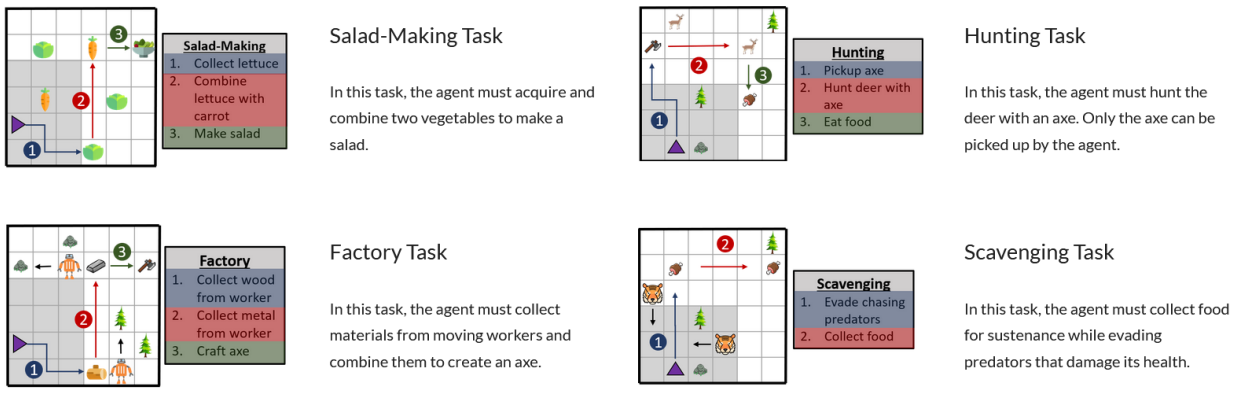

Figure 3.1: Experiments investigating learning in a non-episodic sparse reward setting

The experiments performed by Co-Reyes et al (2022) illustrate how agents training in dynamic environments

that have been shaped perform better than the agents that train in static unshaped environments. Each trained agent is the expected to complete the predefined task in an episodic version of their training environment in order to evaluate whether the Eco-RL properties were useful in facilating learning in the environment. Since similar success rates were found in all the experiments, we will only explain the results of one environment known as the Hunting Environment.

In the Hunting Task described above in Figure 3.1, if the agent is trained in a static environment, it will succeeds in catching the deer 0% of the time. If it trains in a dynamic environment, the trained agent will succeed in catching the deer 72% of the time illustrating how dynamism can facilitate learning in RL agents. Co-Reyes et al (2022) also compare the performance of environmental shaping and reward shaping on the same Hunting task. The agent trained in an shaped environment succeeded 40% of the time which outperforms agents trained with reward shaping that only succeeded at the task 22% of the time.

The experiments performed by Co-Reyes et al (2022) show that the Ecological RL framework produces human-like environments that assist agents in learning. Since Co-Reyes et al (2022) do not explicitly mention which algorithms were used in these experiments, we will reproduce the Hunting task environment and implement various algorithms to evaluate their performance in these Eco-RL Environments.

3.1 Implementations: Hunters & Taxis

The Hunting Task described in figure 3.1 is not easy to implement. Co-Reyes et al (2022) implemented the environment using MuJoCo, an open source physics engine, which is not compatible with many RL algorithm libraries. Our implementation of the Hunting task should conform to the OpenAI Gym API so it is compatible with RL Algorithm packages such as Stable Baselines. It should be noted that the Hunting task environment has many similarities to OpenAI's Taxi-v3 environment visualised below in figure 3.2. The agent's (taxi) task is to pick up the passenger and then drop of the passenger at the desired location, which introduces a sequential element to the task.

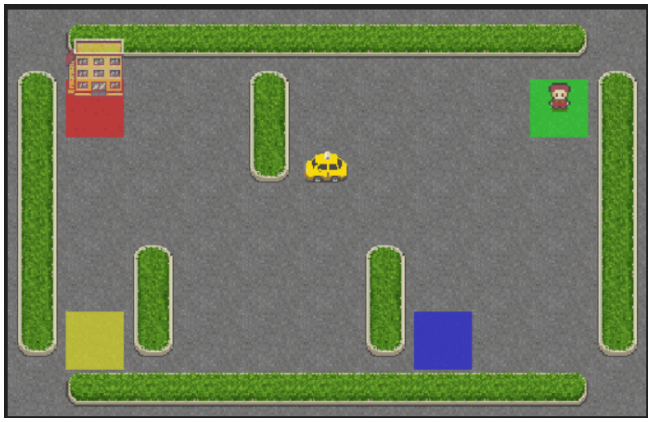

Figure 3.2: OpenAI's Taxi-v3 environment

The taxi and the passenger are randomly placed in either the red, blue, green or yellow blocks (Both will not be placed at the same block). The taxi has a discrete action space consisting of six actions, up, down, left, right, pick up and drop off. The taxi will be penalised if it attempts to drop off the agent when it is not in the taxi or at the wrong location. The taxi will similarly be penalised if it attempts to pick up the passenger at the wrong location. The taxi will however, receive a reward for picking up the passenger and dropping them off at the correct location.

The Taxi-V3 environment can be considered a static and episodic version of our Hunting task since both

require the agent to complete two tasks sequentially without a hard reset between the two tasks. The taxi task is substantially easier to accomplish compared to the Hunting task described by Co-Reyes et al (2022) due to the stationary drop-off and passenger location. We will run several algorithms on the simpler Taxi Environment and select the best performing ones to apply to the Hunting environments. The Taxi environment is created by OpenAI and forms part of their Toy text gym environments. We can thus create our own implementation of the Hunting Task described in figure 3.1 based off the implementation of the Taxi environment.The task of described in the Hunting Experiment of Co-Reyes et al(2022) is to train an agent (hunter) to first pick up and axe and then hunt a deer. Two random initialisations of our implementation is shown in Figure 3.3 which reflects a different positions for the prey (targets), axe as well as the hunter (agent).

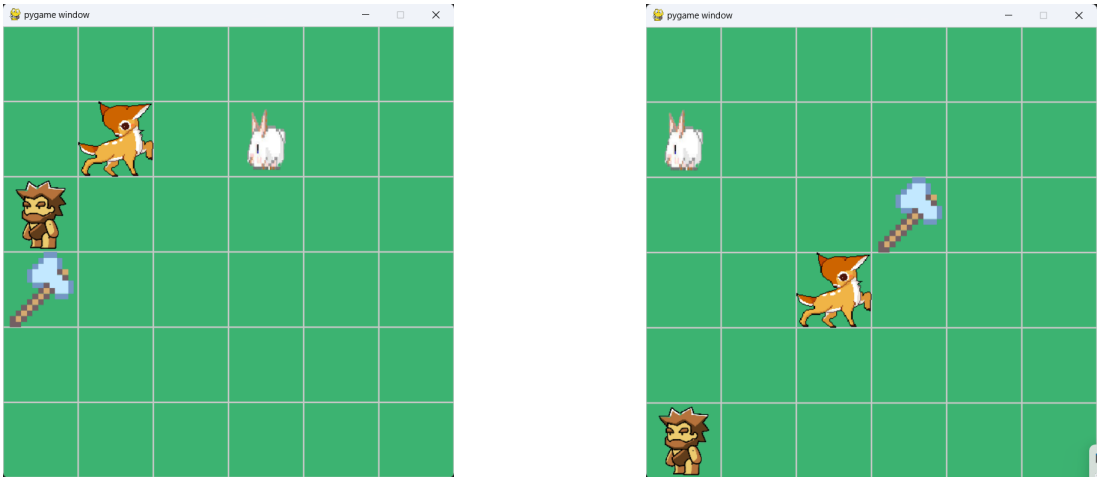

Figure 3.3: Gym implementation of Hunting Task

The task only mentions an axe and deer, however there is also a bunny present in figure 3.3 above. This bunny represents the implementation of environmental shaping and is thus an easier target than the deer. The bunny alters the initial state distribution $p(s)$ of the system and reduces the mismatch coefficient of the hunters current policy and the optimal one. The bunny is easier to catch since it has a 0.1 probability of moving towards the agent and a 0.9 probability of remaining in the same place. The deer is the unshaped target that will move away from the agent with a probability of 0.2, will take a random step with a probability of 0.7 and remain in the same place with a probability of 0.1. We train the agent to first catch the Bunny, which is a significantly easier task compared to catching the deer. Once that is accomplished, the agent will more easily be able to learn how to complete the original task of hunting the deer.

The action space of this environment, similar to the Taxi environment, is a discrete action space consisting of six possible actions, moving up, down, left, right, picking up the axe and hunting the animal. The reward function in this environment consists of both rewarding and penalising the agent for performing specifc actions at specific locations. The hunter is penalised if it attempts to pick up the axe or kill the prey in the wrong location and is rewarded for doing so in the right location. The reward mechanism can be summarised as follows:

Action	Reward
Pick up axe at axe location	+20
Pick up axe not at axe location	-10
Kill prey at prey location	+20
Kill pret not at prey location	-10

Table 3.1: Description of reward mechanism

We also implemented a -1 Reward for each step taken so the agent attempts to get the task done as efficiently as possible. We have now described two environments in which agents must accomplish a sequential task. The two environments differ in level of difficulty due to the dynamism in the Hunting environment. Completing tasks in a dynamic environment is extremely challenging, however Co-Reyes et al (2022) has shown that, if implemented correctly, dynamism can help an agent learn. Co-Reyes et al (2022) do not explicitly mention which algorithm is used when trained the agents in their experiments. Deep Reinforcement Learning algorithms are most commonly used for these experiments due to their ability to complete tasks in a variety of environments.

Chapter 4

Deep Reinforcement Learning Algorithms

Deep Learning is a family of machine learning methods that can be used to learn complex patterns from some given data (Goodfellow, Bengio & Courville, 2016). Neural networks, which form the core of Deep Learning, are inspired by the the structure of the human brain. Neural Networks consist of perceptron that interact similarly to how neurons do in our brains. The interactions between synapses and neurons in the brain produce our cognitive abilities which allow us to make decisions and perform actions, otherwise known as exhibiting intelligence. Machines utilise neural networks should thus be able to exhibit human intelligence (Goodfellow, Bengio & Courville, 2016), however this is not the case since the amount of computational power required to build a neural network that is the same size as a human brain does not exist yet. Instead we can combine Deep learning with Reinforcement learning to develop Intelligent RL agents.

Deep Reinforcement Learning (Deep RL) is a subfield of RL that combines reinforcement learning with deep learning . Agent can utilize Neural networks to learn intricate patterns and representations from high-dimensional input spaces to solve a wide range of complex decision-making tasks (François-Lavet et al, 2018). The integration of deep neural networks allows agents to handle complex tasks in various fields, such as gaming, robotics and natural language processing.

Reinforcement learning trains an agent to take the optimal action to take given the state of the system. Agents will follow something known as a policy which tells our agent which action to take given the state of the system. RL algorithms aim to find the optimal policy using methods that can be categorised as either policy-based or value-based method. Deep RL has been used to train agents to accomplish a variation of tasks, thus we will outline one value-based and one policy-based method.

4.1 Deep Q-Networks

Deep Q-Networks are an extension of Q-learning, a standard value-based RL algorithm. The core concept in Q-learning are the Q-values which are the expected cumulative future rewards the agent can obtain by taking a certain action in a particular state or following a certain policy. These Q-values are placed in a Q-table which acts as a lookup table that the agent refers to when selecting which action to perform to maximise reward. The values of the Q-table are found iteratively through the Bellman Equation shown and described below (Sutton & Barto, 2018).

$$\underbrace{\text{New}Q(s,a)}_{\text{New Q-Value}} = Q(s,a) + \underbrace{\alpha}_{\text{Learning Rate}} [\underbrace{R(s,a)}_{\text{Reward}} + \underbrace{\gamma}_{\text{Discount rate}} \overbrace{\max Q'(s',a')}^{\text{Maximum predicted reward, given new state and all possible actions}} - Q(s,a)] \tag{4.1}$$

The Q-learning algorithm works by allowing agents to explore different states to find the different state-action value pairs. The problem arises when the agent is given action and state spaces that are too large to explore and iteratively update. Deep Q-networks can, however, mitigate this by learning a Q-function instead of Q-values (Mnih et al, 2013) . The Deep Q-Networks algorithm consists of two neural networks, the Q-network and the Target-network as well as an additional component known as Experience Replay. The Q-network takes the current state and action as input predicts the Q values for any selected action. The Target-Network predicts the best Q value out of all actions that can be taken from that state, also known as the target Q-value. We use a loss function to compute the difference between the two Q-values from the two networks and update the Q-Network by minimising the loss function (François-Lavet et al, 2018). DQN also utilses the Experience Replay technique, which is a memory technique that stores data about past experiences (current state, action, reward, next state) from an agent's interaction with an environment into a table known as Replay memory. The two Neural Networks can then randomly sample data from the Replay Memory to update their weights and learn. Experience replay was introduced to prevent agents from 'Catastrophic forgetting', where the agent forgets the things it previously learnt. A summary of how DQN works is shown in figure 4.1 below.

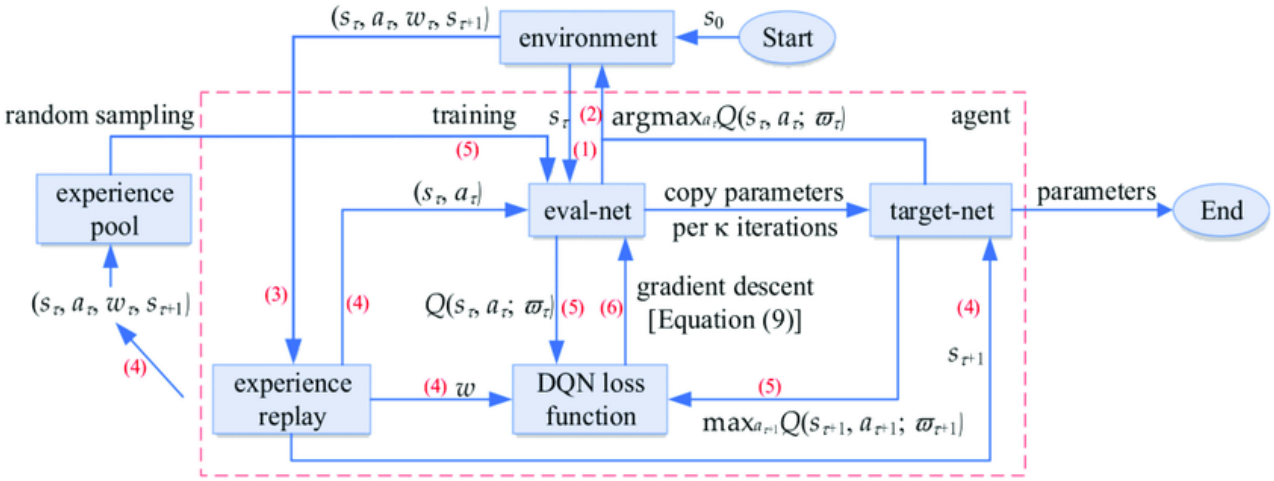

Figure 4.1: DQN flow(Wang et al, 2021)

The figure above begins at the Start with the agent interacting with the environment via the Q network (denoted Eval network in the figure). The interactions are defined in terms of the current state, action, reward and next state denoted $(s_t, a_t, \omega_t, s_{t+1})$ that are then fed into the Experience Replay mechanism. The entire interaction is then stored in the Experience Pool, also known as Replay Memory. The current state and action of the agent (s_t, a_π) is sent to the Q network, where it predicts the Q value $Q(s_t.a_t, \bar{w}_t)$ where $\bar{w}_t$ denotes the weights of the associated Neural Network. The future state (s_{t+1}) is fed into the Target network where it returns the best possible action and Q-value given the future state $\text{Max}Q(s_{t+1}, a_{t+1}, \bar{w_{t+1}})$. Both Q-values from the two networks as well as the current reward w is fed into the loss function where the difference between the Target Q-value, Q-network Q value as well as current reward is computed. The weights of Q-network is updated using this loss function using a variation of gradient descent shown in equation 4.2 below where α is the learning rate (Mnih et al, 2013).

$$w_{t+1} = w_t + \alpha[Q_{Max} - Q_{eval}(s_t, a_t.w_t)]\nabla_w Q_{eval}(s_t, a_t, w_t) \tag{4.2}$$

The weights of the Q-network are iteratively updated with equation 4.2 shown above, however, the weights of the Target-network are not similarly updated. Instead, the target network is replaced by a copy of the Q-Network after k iterations where k is some integer that is chosen arbitrarily. This is how both the Q-Network as well as the Target network is updated and the Q network learns a better approximation of the Q function which allows the agent to select the best action to achieve its' predefined task (Mnih et al, 2013). This process is known as the DQN algorithm which is an off-policy, value-based algorithm. Value-based algorithms have two key characteristics, they estimate the value function of various state-action pairs and find an optimal policy based on these values. Although value-based algorithms have been shown to be extremely powerful in solving several tasks in a diverse set of environments, such as playing Atari (Mnih et al, 2013), there are cases where it fails. Policy-based methods are therefore preferred under certain conditions such as continuous action spaces or stochastic policies (Miller, 2021).

4.2 Proximal Policy Optimization

Proximal Policy Optimization, also known as PPO, is a policy-based Deep RL algorithm that is used to find an optimal policy for some given task. PPO optimises a policy (rule) which the agent follows to complete its' task. The policy is parameterised into a Neural Network with parameters θ which incorporates the "deep" part of the Deep RL algorithm. The policy parameters are updated using incremental steps to ensure stable convergence (Simonini, 2022) . Policy Gradient methods optimise an objective function in order to converge to the optimal policy. The objective function J for any policy gradient methods is to maximise cumulative reward by finding optimal θ as described in equation 4.3 (Simonini, 2022).

$$\begin{aligned} J(\theta) &= \mathbb{E}[\sum_{t=0}^{T-1} r_{t+1}] \\ J(\theta) &= \sum_{t=i}^{T-1} P(s_t, a_t|\tau) r_{t+1} \end{aligned} \tag{4.3}$$

The $\mathbb{E}$ denotes the expected value which is calculated via $\mathbb{E}[f(x)] = \sum_x P(x)f(x)$ where $P(x)$ is the probability of the occurrence x. $P(s_t, a_t|\pi)$ is the probability of encountering s_t, a_t given trajectory τ. If equation 4.3 is used to update Policies, the results are often variable and may not converge to an optimal policy. PPO uses a new objective function known as the clipped surrogate objective function which constrains the policy update at each step (Schulman et al, 2017). The new objective function consists of a ratio function $p_t(\theta)$ that compares the probability of taking action a_t at time t if we are in state s_t according to both the current and old policy. The ratio function is shown in equation 4.4. If $p_t(\theta) > 1$, then taking action a_t at state s_t is more likely in the current policy compared to the old. If $p_t(\theta)$ is between 0 and 1 then it is less likely for action a_t to be taken in state s_t according to the current policy (Simonini, 2022).

$$p_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{OLD}}(a_t|s_t} \tag{4.4}$$

The ration function also provides an easy way to estimate divergence between the current and old policy. The clipped objective function also utilising Clipping, which is a mechanism that penalises changes if they lead to an $r_t(0)$ away from 1. This ensures that the old and current policy do not diverge greatly so the policy is updated incrementally. Although Clipping can be done using TRPO and Kubleck-Lieber Divergence, we will primarily focus on Clipping using the Ratio function $p_t(\theta)$ since it is easier to implement and provides better performance (Simonini, 2022). This clipping is denoted $clip(r_t, \theta), 1-\epsilon, 1+\epsilon)$ where ϵ is chosen to define the clip range $[1-\epsilon, 1+\epsilon]$. The Clipped Surrogate Objective Function utilising Clipping and the ratio function $p_t(\theta)$ is shown below in equation 4.5 with $\hat{A}_t$ denoted an advantage factor.

$$J^{CLIP}(\theta) = \hat{\mathbb{E}}[min(p_t(\theta)\hat{A}_t, clip(p_t, \theta)1-\epsilon, 1+\epsilon)\hat{A}_t)] \tag{4.5}$$

If action a_t is better than all other possible actions in state s_t, then $\hat{A}_t > 1$. If action a_t is worse than all the other possible actions in state s_t then $\hat{A}_t < 1$. The objective function selects the minimum value between the clipped and non-clipped objective function. Figure 4.2 shows that different combinations of $\hat{A}_t$ and ratio function $p_t(\theta)$ will result in different values returned from the objective function.

	$p_t(\theta) > 0$	A_t	**Return Value of** *min*	**Objective is Clipped**	**Sign of Objective**	**Gradient**
1	$p_t(\theta) \in [1-\epsilon, 1+\epsilon]$	+	$p_t(\theta)A_t$	no	+	✓
2	$p_t(\theta) \in [1-\epsilon, 1+\epsilon]$	−	$p_t(\theta)A_t$	no	−	✓
3	$p_t(\theta) < 1-\epsilon$	+	$p_t(\theta)A_t$	no	+	✓
4	$p_t(\theta) < 1-\epsilon$	−	$(1-\epsilon)A_t$	yes	−	**0**
5	$p_t(\theta) > 1+\epsilon$	+	$(1+\epsilon)A_t$	yes	+	**0**
6	$p_t(\theta) > 1+\epsilon$	−	$p_t(\theta)A_t$	no	−	✓

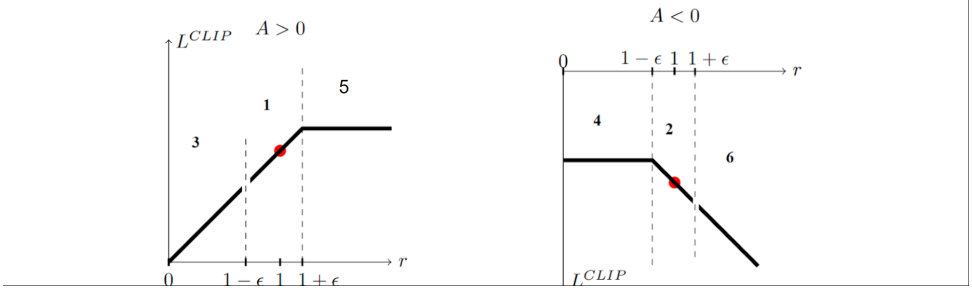

Figure 4.2: Ratio functions along with their corresponding resulting objectives and gradients (Simonini, 2022)

Case 1 and 2 are simple to understand. If the ratio function $p_t(\theta)$ is with the clipping range, then, regardless of the Advantage factor, the unclipped objective will be returned and parameters θ will be updated using some calculated gradient. Case 3 and 6 represent cases where the ratio function is outside the clipping range but the clipped objective is not used. Case 3 has $p_t(\theta) < (1-\epsilon)$ indicates the probability of taking action a_t in state s_t is low. However, it also has a positive $\hat{A}_t$ which means that taking action a_t is good, thus we will thus update the probability of taking action a_t in order to find the optimal policy. Case 6 has $p_t(\theta) > (1+\epsilon)$ and a negative $\hat{A}_t$ which means the action a_t is bad and there is a high probability of taking it, thus the policy must be updated using the calculated gradient to reduce the probability of taking this action. Case 4 and 5 represent cases where the Probability of taking a good action is already high and the probability of taking a bad action is already low and there is therefore no need to update the policy.

Now that we understand what the objective function is, we can utilise it to update the policy π_θ. This objective function is used to update the policy according to equation 4.6 below. This means that θ is updated every k steps to the θ value that maximises reward (Schulman et al, 2017a).

$$\theta_{k+1} = argmax_\theta \mathbb{E}[J(\theta_k)] \tag{4.6}$$

Equation 4.6 can be used iteratively to update the parameters θ of a policy. This is done until the parameters are no longer updated and the algorithm has converged to an optimal policy. The Psuedocode of the PPO algorithm from its' original paper is shown below in figure 4.3

Algorithm 4 PPO with Adaptive KL Penalty

Input: initial policy parameters θ_0, initial KL penalty β_0, target KL-divergence δ
for $k = 0, 1, 2, ...$ **do**
  Collect set of partial trajectories $\mathcal{D}_k$ on policy $\pi_k = \pi(\theta_k)$
  Estimate advantages $\hat{A}_t^{\pi_k}$ using any advantage estimation algorithm
  Compute policy update

$$\theta_{k+1} = \arg\max_{\theta} \mathcal{L}_{\theta_k}(\theta) - \beta_k \bar{D}_{KL}(\theta||\theta_k)$$

  by taking K steps of minibatch SGD (via Adam)
  if $\bar{D}_{KL}(\theta_{k+1}||\theta_k) \geq 1.5\delta$ **then**
    $\beta_{k+1} = 2\beta_k$
  else if $\bar{D}_{KL}(\theta_{k+1}||\theta_k) \leq \delta/1.5$ **then**
    $\beta_{k+1} = \beta_k/2$
  end if
end for

Figure 4.3: PPO pseudocode (Schulman et al, 2017a)

In this section we have described two Deep RL algorithms that incorporate neural networks into their algorithms in order to better learn the optimal sequence of actions that will complete a task. Deep Reinforcement learning has allowed RL to be used to perform a variety of tasks indicating the power of these algorithms (François-Lavet et al, 2018). These algorithms do, however, have shortcomings. Since it utilises neural networks, Deep RL algorithms are considered data hungry and will require a large amount of data to learn efficiently. These algorithms have also been shown to produce agents with unexpected behaviour which can lead to dangerous actions. Lastly, deep RL algorithms, like many others, focus on maximising cumulative reward.

The reward given at each time step is determined by a reward function which guides the agent to complete its' task. However, reward functions are difficult to design since it must captures the objective of a task, which in many cases, is not easily (Miller, 2021). Reward functions, if designed incorrectly can incentivize agents to perform unexpected and potentially harmful behaviour. When using algorithms that require a reward signal, such as Deep RL algorithms, designing the reward function is often the most challenging part of implementation. Imitation learning is a new subfield in RL that can be used to help agents to learn equally as well, if not better, than deep RL algorithms without using an explicitly designed reward function.

Chapter 5

Imitation Learning

Imitation learning is a subfield of RL that draws inspiration from imitative learning , a type of social learning where animals acquires a novel action after watching another individual perform it (Over & Carpenter, 2012). Imitative learning in humans in particular has been well studied due to a demonstrator's (eg. celebrity) ability to influence other humans to copy their actions, regardless of whether the action is good or bad. Children in particular tend to over-imitate (Over & Carpenter, 2012). RL agents are often compared to children since both must learn to perform tasks from scratch with no prior experience. Imitation learning represents an important field of RL where RL agents are trained in a way similar to children to complete tasks and potentially develop human intelligence in the process.

Recall that agents in a Standard RL framework learn by maximising cumulative reward. This makes the reward function important since it will influence an agent's behaviour as well as how well it will complete its' task. Since poorly designed Reward functions can lead to unexpected or unsafe behaviour, designing a good Reward function is thus incredibly important for agents to learn how to complete tasks successfully and safely (Miller, 2021). Imitation Learning is a new subfield of RL that reduces the need and thus challenges of designing a reward function allowing models to learn to complete a task without any unexpected behaviour (Smartlab AI, 2019).
Imitation learning occurs when an agent is given samples of expert demonstrations and attempts to mimic the expert's actions through various means (Hussein et al, 2017) . The expert's behaviour is considered optimal and ethical thus mimicking it will train the agent to optimally complete tasks. The simplest imitation learning method is known as **Behaviour Cloning** where the agent learning the expert's policy using supervised learning. The agent is given expert demonstrations in terms of trajectories $\pi*$, these expert trajectories can be broken down into a set of state-action pairs

$$(s_0*, a_0*), (s_1*, a_1*), (s_2*, a_2*)... \tag{5.1}$$

Behaviour Cloning attempts to create policy π_θ that mimics $\pi*$. This is done using some sort of supervised learning method such as a Neural Network (Stanford, 2023). Policies can be interpreted as a mapping from the input state to the output action. The neural network produces an optimal policy by minimising the loss between the current and optimal policy. Although Behaviour cloning is simple to implement, it has many issues. Behavioural cloning results in agents that do not explore nor adapt well and also fails when implemented in certain tasks and environments. In particular, Behaviour cloning fails in continuous action spaces or when there are multiple optimal strategies to accomplish a task (Stanford, 2023). One of the critical issues in Behavioural Cloning is the compounding errors that can result in agents behaving erratically. A new form of imitation learning, known as Inverse Reinforcement learning was developed in order to reduce the prevalence of compounding fatal errors.

5.1 Inverse Reinforcement Learning

Inverse Reinforcement learning attempts to optimise the inverse objective function of the Standard RL framework. The inverse of the reward function is the cost function which determines the cost of taking action a in state s. If Standard RL maximised the reward function, then Inverse RL minimizes the cost function. It is through using this cost function that IRL algorithms reduce the prevalence of fatal errors in RL agents. Through mimising the cost function, the agent attempts to learn the reward function that would generate the experts behaviour instead of the expert's policy . The reward function is often an accurate representation of the task, thus learning the correct reward function can allow the agent to learn to complete the same task in novel environments.

One of the benefits of IRL is that it can allow agents to generalise and adapt which are key characteristics of Intelligent RL agents. Inverse Reinforcement learning focuses primarily on learning reward functions that can be combined with other standard RL algorithms to train agents. This can be seen as both a benefit and drawback of IRL since the IRL algorithm will exhibit the advantages and issues of the selected RL algorithm used to train the agent. IRL also only requires demonstrators behaviour to be **near** optimal compared to behaviour cloning which expects expert demonstrations with optimal trajectories.

Inverse Reinforcement Methods can fall into one of three categories. The methods can be classed as Maximal Margin Methods, Maximum Entropy methods or Bayesian Methods. We will describe each class of IRL methods briefly.

5.1.1 Maximal Margin Methods

Maximal Margin methods were the first IRL algorithms described in 2000 (Stanford, 2023). The first IRL algorithm assumed three things:

1. Optimal policy is known
2. State space is finite
3. State transition dynamics are known

The second and consequently third condition were relaxed so agents could learn non-trivial optimal solutions to a task. Lastly the first condition is relaxed and the optimal policy is assumed unknown so the agent learns from the trajectories of the optimal policy rather than the policy itself. The Maximal Margin methods attempts to estimate reward functions that will maximize the Euclidian distance between the optimal policy and all other policies (Adams, Cody & Beling, 2022). These methods assume that the reward is a linear combination of features of the environment or task as shown below where ϕ_i represents some features of the problem.

$$R(s) = \alpha_1\phi_1(s) + \alpha_2\phi_2(s) + \cdots + \alpha_n\phi_n(s) \tag{5.2}$$

α_i represents weights that are updated to learn the optimal Reward function. These Margin based methods rely heavily on expert trajectories which is a critical limitation of early IRL algorithms since expert trajectories are often not readily available (Adams, Cody & Beling, 2022). IRL methods based on Bayesian statistics were then developed. This second class of IRL algorithms, known as **Bayesian Methods**, have two defining characteristics.

1. Maximizes the posterior distribution of the reward function
2. Use prior distributions on the reward to define the posterior distribution.

5.1.2 Bayesian Methods

The first Bayesian algorithm was developed in 2007 modified Markov Chain Monte Carlo (MCMC) algorithm to determine the reward, which is a probabilistic model consisting of several reward functions (Adams, Cody & Beling, 2022). The Bayesian method is an improvement on the Maximal Margin methods since the optimal policy is not required, nor is it assumed that the expert behaiour is optimal (Stanford, 2023). External information about the problem is also integrated into the reward function through the prior distribution in Bayesian statistics which allows the reward to be more complex and thus train agents better. Bayesian algorithms do however assume that the environment is fully observable. .. Bayesian methods will also fail if the agent encounters states that the expert never interacted with. There have been several variations of Bayesian algorithms that have addressed these limitations, however these algorithms are complex and still struggle to learn from noisy trajectories. **Maximum Entropy methods** is the last class of Inverse Reinforcement Learning Algorithms that can learn from noisy trajectories from near-optimal demonstrations.

5.1.3 Maximum Entropy methods

Maximum Entropy (MaxEnt) methods use maximum entropy to estimate the reward function r which is parameterised according to equation 5.3 with parameters θ and $f_{\mathcal{T}}$ representing some features of the states in path $\mathcal{T}$ (Hussain, 2023).

$$r(f_{\mathcal{T}}) = \theta^T f_{\mathcal{T}} \tag{5.3}$$

In MaxEnt methods, it is assumed that an agent is observing a demonstrator that maximizes a linear mapping from the state features to rewards which means the demonstrator has found the optimal policy to maximise cumulative reward. The the probability of observing any single trajectory from the expert is shown in equation 5.4 where θ is a vector of feature weights, f_{T_M} is the path feature count, and $Z(\theta)$ is the partition function given the feature weights. (Hussain, 2023)

$$\mathbb{P}\left(\mathcal{T}_m \mid \theta\right) = \frac{1}{Z(\theta)} e^{\theta \mathbf{f}_{\mathcal{T}_{I_m}}} \tag{5.4}$$

Equation 5.4 represents the probability of observing trajectories in a deterministic MDP, otherwise known as a static environment. However, for non-deterministic trajectories, Equation 5.5 represents the probabilities of observing the trajectories of the demonstrators where T represents the Transition distribution.

$$\mathbb{P}(\mathcal{T}|\theta, P) \approx \frac{e^{\theta^\top \mathbf{f}_{\mathcal{T}}}}{Z(\theta, T)} \prod_{s_{t+1}, a_t, s_t} P_T(s_{t+1}|a_t, s_t) \tag{5.5}$$

The reward weights are learned from the demonstrated behavior by maximizing the log likelihood $L(\theta)$ under the entropy distribution shown in equation 5.5 (Hussain, 2023). This means we will find the parameters θ that will increase the likelihood of the agent observing states found in the demonstrators trajectories. The reward weights $\theta*$ along with $L(\theta)$is shown below where $\tilde{\mathcal{T}}_m$ is the m_{th} observed trajectory

$$\theta^* = \arg\max_{\theta} \sum_{m=1}^{M} \log \mathbb{P}\left(\tilde{\mathcal{T}}_m \mid \theta, P\right)$$
$$L(\theta) = \log P(\mathcal{T}|\theta, T)$$

Maximum entropy methods are easily extended to the continuous space, however the method has a few strong assumptions. It first assumes that the system dynamics are known as well as that the number of states is finite so it will not be difficult to compute the state visitation frequency (Stanford, 2023). Unfortunately

these assumptions do not hold for many RL tasks. Maximum Entropy methods have been widely implemented in cases where these assumptions hold due to MaxEnt's ability to learn from noisy trajectories and sub-optimal demonstrator behaviours. There have been many extensions of MaxEnt methods that have been modified to accomplish different tasks in different environments, however, as with any IRL algorithm, the inferred reward function must be used alongside some RL algorithm to generate optimal policies.

5.2 Apprenticeship Learning

Apprenticeship Learning and IRL are both utilised in Imitation learning, however they differ in approach and goal. Inverse Reinforcement Learning focuses primarily on finding a good reward function that can later be used to find an optimal policy, whereas Apprenticeship learning focuses on finding the optimal policy directly (Adams, Cody & Beling, 2022)). Apprenticeship learning does not require additional subroutines to find costand reward functions making it more computationally efficient. This would imply that Apprenticeship learning is preferred, however, Piot et al (2013, 2017) compared the performance of policies found using IRL and Apprenticeship Learning and found that IRL outperformed Apprenticeship Learning, especially in the case of noisy demonstrations or POMDPs (instead of MDPs).

Generative Adversarial Imitation Learning (GAIL) is a new apprenticeship learning algorithm that builds on previous IRL methods to combine the benefits of both algorithm classes. GAIL also utilises a well-known deep learning architecture known as Generative Adverserial Networks (GANS) (Adams, Cody & Beling, 2022).

5.2.1 What is a GAN?

Generative Adversarial Networks (GANs) are a type of generative model that consist of two neural networks known as the generator and the discriminator (Bau et al, 2013). The generator network takes random noise as input and generates data samples, such as images or text, that is fed to the discriminator which evaluates and classifies these generated samples as real or fake. This adversarial interaction between the generator and the discriminator allows the two networks to compete and thus iteratively improve one another. The basic structure of a GAN is shown below in Figure 5.1

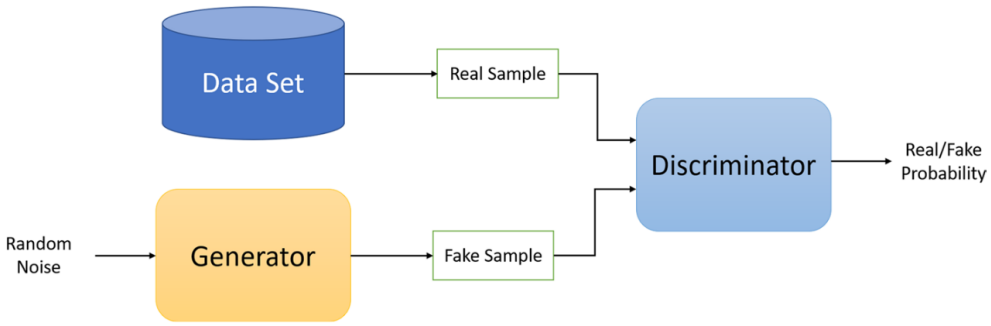

Figure 5.1: Basic Architecture of a GAN(Adams, Cody & Beling, 2022)

The training process in GANs is a minimax game (Bau et al, 2013), where the generator aims to produce data that is indistinguishable from real data, while the discriminator aims to correctly classify samples as real or fake. Over time, this leads to the generator creating increasingly realistic data, and the discriminator becoming better at distinguishing real from generated data. GANs have been highly successful in a wide range of applications, including image generation, style transfer, super-resolution, and even the generation of text and music. They have significantly advanced the field of generative modeling and are used in creative applications, such as art generation and content synthesis, as well as in practical applications like data augmentation and

image editing.

One of the benefits of GANs is their ability to generate diverse and realistic data samples (Bau et al, 2013). Variations of GANs have also been developed such as Conditional GANs (cGANs), Wasserstein GANs (WGANs), and BigGANs. GANS are particularly popular in Generative AI with models such as MidJourney frequently used as a baseline for Image generation, however the architecture can be modified and applied to many different spheres of AI, for example, it can be applied to Reinforcement Learning in the context of GAIL (Ho & Ermon, 2016).

5.3 Generative Adverserial Imitation Learning (GAIL)

GAIL is a variation of the optimisation problem in Causal MaxEnt IRL methods that was first described by Ho & Ermon (2016). The optimisation problem is shown in equation 5.6 . This equation has variables π and π_E representing the learnt and expert policy respectively. $\mathbb{E}$ represents the expected cumulative cost that an agent would incur when following a policy π. $H(\pi) \triangleq \mathbb{E}_\pi[-\log \pi(a \mid s)]$ is the γ-discounted causal entropy of the policy where γ is some variable set arbitrarily. Maxent IRL attempts to find a cost function $c \in \mathbb{C}$ that assigns high costs to the learning policy and low costs to the expert policy as shown by the Maximize operator on $\mathbb{E}_\pi$ (Ho & Ermon, 2016).

$$\underset{c \in \mathcal{C}}{\text{maximize}} \left(\min_{\pi \in \Pi} -H(\pi) + \mathbb{E}_\pi[c(s,a)] \right) - \mathbb{E}_{\pi_E}[c(s,a)] \tag{5.6}$$

The found cost function can be used in any RL algorithm to find the optimal policy that will minimise cumulative costs. Larger environments will produce larger sets of cost functions since there are more actions and state combinations that will not lead the agent closer to accomplishing its' task. GAIL assumes that the set of cost functions $\mathbb{C}$ is large, however this will result in the policy overfitting to its' training dataset . A cost function regularizer ϕ is thus implemented to allow GAIL to be utilised in large environments without overfitting.

The modified Optimisation problem computed in GAIL is shown below in equation 5.7. where the Cost function Regularizer is shown in equation 5.8 and 5.9.

$$\underset{\pi}{\text{minimize}} \psi^*_{\text{GA}}\left(\rho_\pi - \rho_{\pi_E}\right) - \lambda H(\pi) = D_{\text{JS}}\left(\rho_\pi, \rho_{\pi_E}\right) - \lambda H(\pi) \tag{5.7}$$

The RHS of equation 5.7 illustrates the optimisation perspective of GAIL. The LHS of the equation reflects GAIL in terms of Jenson-Shannon Divergence[1] ($= D_{\text{JS}}$) and uses casual entropy H as a policy regulariser that encourages exploration in the policy. Equation 5.7 attempts to find a policy with an occupancy measure[2] that minimizes Jensen-Shannon divergence to the expert's.

The regularizer shown below only penalises cost functions c minimally if it outputs a negative cost amount to expert state-action pairs. If function c assigns large costs to the expert policy , then the regularizer ϕ_{GA} will heavily penalize cost function c.

$$\psi_{\text{GA}}(c) \triangleq \begin{cases} \mathbb{E}_{\pi_E}[g(c(s,a))] & \text{if } c < 0 \\ +\infty & \text{otherwise} \end{cases} \quad \text{where } g(x) = \begin{cases} -x - \log\left(1 - e^x\right) & \text{if } x < 0 \\ +\infty & \text{otherwise} \end{cases} \tag{5.8}$$

[1] Jensen-Shannon Divergence (DJS): The Jensen-Shannon divergence is a measure of the similarity between two probability distributions.

[2] An occupancy measure, often denoted as $p_\pi(s)$, is a probability distribution that characterizes the likelihood of being in a particular state s when following a specific policy π in a Markov decision process (MDP). It quantifies how frequently an agent visits or occupies each state during the execution of the policy.

$$\psi^*_{\mathrm{GA}}\left(\rho_\pi - \rho_{\pi_E}\right) = \max_{D \in (0,1)^{\mathcal{S}\times\mathcal{A}}} \mathbb{E}_\pi[\log(D(s,a))] + \mathbb{E}_{\pi_E}[\log(1 - D(s,a))] \tag{5.9}$$

GAIL trains both a generative model G and a discriminative classifier D as it is a GAN. The discriminative classifier D attempts to distinguish between the generated distribution of data and the true data distribution. The Generator G will succeed if the Discriminator can no longer differentiate between real data and data generated by G. This generator-descriminator architecture is applied to imitation learning in GAIL. The generated data distribution in a GAN is synonymoous with the agents' occupancy measure p_π and the demonstrators occupancy measure p_E is equivalent to the true data distribution in GAIL.

$$\mathbb{E}_\pi[\log(D(s,a))] + \mathbb{E}_{\pi_E}[\log(1 - D(s,a))] - \lambda H(\pi) \tag{5.10}$$

The policy of the agent π_θ is parameterized using weights θ and Discriminator are parameterised using a Discriminator Network D_w with weights w. Recall that GAIL intends to solve the optimisation problem in 5.7. This can be translated finding a saddle point (π, D) in equation 5.10, where the equation is now in the context of discriminator networks and generators. GAIL attempts to find this saddle point and consequently solve equation 5.7 and 5.10. The GAIL algorithm is shown below in figure 5.2.

Algorithm 1 Generative adversarial imitation learning

1: **Input:** Expert trajectories $\tau_E \sim \pi_E$, initial policy and discriminator parameters θ_0, w_0
2: **for** $i = 0, 1, 2, \ldots$ **do**
3: Sample trajectories $\tau_i \sim \pi_{\theta_i}$
4: Update the discriminator parameters from w_i to w_{i+1} with the gradient

$$\hat{\mathbb{E}}_{\tau_i}[\nabla_w \log(D_w(s,a))] + \hat{\mathbb{E}}_{\tau_E}[\nabla_w \log(1 - D_w(s,a))] \tag{17}$$

5: Take a policy step from θ_i to θ_{i+1}, using the TRPO rule with cost function $\log(D_{w_{i+1}}(s,a))$. Specifically, take a KL-constrained natural gradient step with

$$\hat{\mathbb{E}}_{\tau_i}\left[\nabla_\theta \log \pi_\theta(a|s) Q(s,a)\right] - \lambda \nabla_\theta H(\pi_\theta), \quad \text{where } Q(\bar{s},\bar{a}) = \hat{\mathbb{E}}_{\tau_i}[\log(D_{w_{i+1}}(s,a)) \mid s_0 = \bar{s}, a_0 = \bar{a}] \tag{18}$$

6: **end for**

Figure 5.2: Basic Architecture of a GAN (Ho & Ermon, 2016)

The GAIL algorithm above begins with sampling trajectories from the agent's policy. The sampled trajectories are fed into the discriminator network D_w where the weights w are updated using Adams optimization(Ho & Ermon, 2016). Once w is updated, then we use TRPO to update the parameters θ of the agents policy π_θ. We use TRPO since it prevents the agent's policy from exhibiting large variations due to noise in the policy gradient.

GAIL is a model-free algorithm which means agents will requires more environmental interactions to learn (Ermon, 2023), however GAIL has been shown to be able to learn in large, high-dimensional environment. It has been shown to perform better than other imitation learning algorithms such as Behaviour cloning or DAGGER. GAIL has widely been acknowledged as a State-of-the-art algorithm in imitation learning that has been used to train agents in robotics, games and several other fields. Since GAIL does not learn a reward function that is unique to a specific task or environment, it allows agents to easily adapt to novel environments (Ermon, 2023), which is a key factor when describing reinforcement learning in a human context.

Chapter 6

Autonomous Reinforcement Learning

In this report, we focus on creating agents that can learn in a non-episodic dynamic environment that mimic reality. Autonomous reinforcement Learning is a topic in RL that focuses primarily on applying RL to autonomous systems [1] Autonomous RL is a framework defined by two key characteristics (Sharma et al, 2021):

1. Agents must learn through own experiences
2. There must be no human intervention

One of the central themes in ARL is applications to reality, thus the ARL framework has multiple overlaps with the Eco-RL Framework defined in Chapter 3. ARL will, similar to standard and Eco-RL, define an Environment as an MDP with an initial state s_0. Where ARL differs from Standard RL is that there are no resets whatsoever, even after the agent completes its' assigned task(s) (Sharma et al, 2021) . ARL uses a learning algorithm $\mathbb{A}$ that maps state transitions to actions. $\mathbb{A}$ can be considered a mapping, from which we can approximate an optimal policy π. We can evaluate the policy using Deployed and Continuing Policy Evaluation equations shown in figure 6.1 below. If the Deployed policy equation return higher values, this will indicate that the quality of the policy is better. The same is true for the Continuing Polict Evaluation Equation where higher values reflect more reward accumulated by an agent over its' lifetime

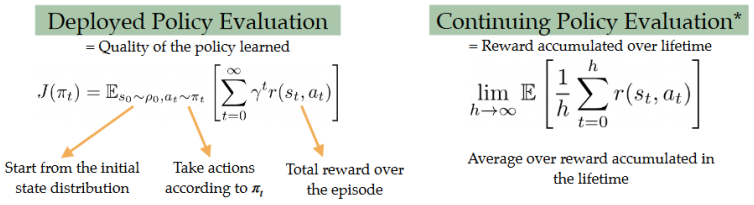

Figure 6.1: Policy Evaluation Equations (Sharma et al, 2021)

We utilise the Deployed Policy Evaluation when the agent is in a deployed setting. The agent is expected to complete its task(s) without resetting itself under these conditions. The Continuing Policy Evaluation Equation is used when the agent's interactions with the environment cannot be separated into training and deployment. When the agent is trained in an ARL framework, the agent must learn to both complete the task and reset itself to continue learning how to improve how it completes the task.

Although standard RL algorithms such as DQN can be implemented in an ARL environment, however, these algorithms will struggle due to the non-episodic nature of the tasks. Algorithms were therefre designed specifically for tasks in an ARL framework. Due to similarities between the ARL frameworks and our Eco-RL

[1] systems that can complete a series of tasks in a changing environment (eg. Self-driving Cars).

Environments, ARL algorithms should similarly perform well in our Human-like Hunting Environment. In particular, we will describe a new ARL algorithm known as MEDAL (Sharma. Ahmad & Finn, 2022).

6.1 Matching Expert Distributions for Autonomous Learning (MEDAL)

MEDAL is a new imitation learning algorithm that has been developed by Sharma, Ahmad & Finn (2022) that has been designed to help agents learn under non-episodic ARL conditions. MEDAL, similar to other non-episodic ARL algorithms, learn two policies. The first is a forward policy π_f shown in equtation 6.1 that solves the assigned task. This policy starts exploring from p_0 which is the state that it will begin at during deployment settings. The second is a backward policy π_b which explores the possible states that can serve as better starting states for the agent to be in, in order to solve task. pi_b is shown in equation 6.2 where D represents the Jensen-Shannon divergence.

$$\pi_f \approx \mathbb{E}\left(\sum_{t=0}^{\infty} \gamma^t r(s_t, a_s)\right] \tag{6.1}$$

$$\pi_B \approx D_{JS}(p^{\pi_b}(s)||p^*(s)) \tag{6.2}$$

MEDAL similarly draws inspiration from imitation learning and utilises expert demonstrations during training Ahmad & Finn. The pink line denoted p^* in figure 6.2 is the Trajectory taken by the expert to get from the initial state p_0 to the final state p_g when the goal is accomplished. The forward policy π_f often explores various states during training, which brings it away from the optimal state. The backwards policy π_b is trained take the agent from the out-of-distribution state back to the states encountered by the demonstrator. Once the agent is back in an ideal (demonstrator) state, the forward policy is used to explore different states again until it reaches the final state p_g.

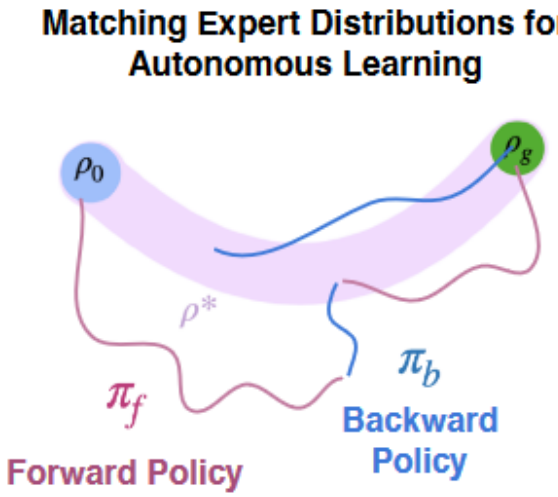

Figure 6.2: Diagram illustrating MEDAL (Sharma. Ahmad & Finn, 2022)

Since ARL is non-episodic, the backwards policy π_b is used to take the agent back to some demonstrator state, from where it is easier to return to p_g. The sequence of using alternating π_f and π_b is thus repeated several times in order to train the agent. Since π_b must return the agent to a state similar to that of the demonstrator p^*, this means that the D_{JS} between the state distribution of the demonstrator and π_b must be minimised. This is done using a Classifier shown in Equation 6.3 below (Sharma. Ahmad & Finn, 2022). The Classifier

rewards the algorithm for selecting being in a state in the demonstrator's state distribution and penalises the agent for not being in a demonstrator state.

$$C(s) = \begin{cases} +1 & s \in demos \\ -1, & s \approx p^{\pi_b}(s) \end{cases} \tag{6.3}$$

Using this Classifier, we can construct an alternative version of π_b that is similar to the forward policy. This is shown in equation 6.3 which is maximised during training.

$$\pi_b \approx -\mathbb{E}\left[\sum_{t=0}^{\infty} log(1 - C(s_{t+1}))\right] \tag{6.4}$$

MEDAL is another imitation learning algorithm that was designed for the Autonomous RL framework by Sharma. Ahmad & Finn. (2022). It has been shown to perform well when applied to ARL benchmark tasks, in particular the Table-top and Sawyer Door environments shown in figure 6.3 below. In the Table-top environment the arm is expected to move the mug to the goal position, return it to its original postion and repeat. The arm in the Sawyer Door environment must learn to close the door from various starting positions and then reopen it to learn how to perform the task again.

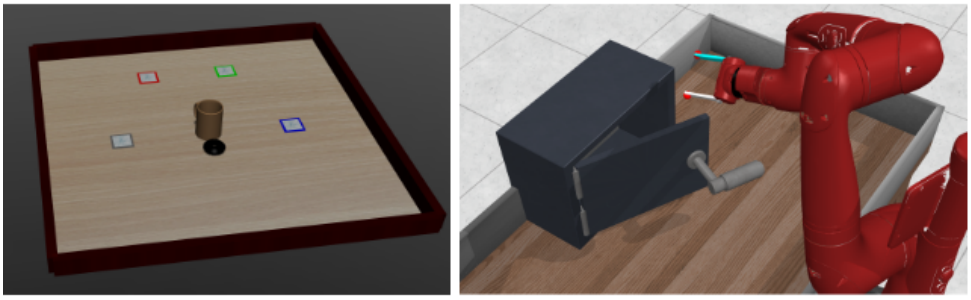

Figure 6.3: ARL Benchmark environments: Table-top (left) and Door closing (right)

The performance of MEDAL in both these environments is evaluated by comparing it to other ARL algorithms such as forward-backward RL (FBRL), value-accelerated persistent RL (VaPRL), R3L which has a backward policy that optimizes a state-novelty reward and standard naive RL whete the arm is trained episodically where only a forward policy optimizes the task using a reward function. All of these algorithms learn a policy which is used to maximise cumulative reward. Each algorithm is run 5 times. The returns of the runs for each algorithm are then averaged and displayed in Figure 6.4 with the standard error displayed in ().

Method	Tabletop Organization	Sawyer Door
naïve RL	0.32 (0.17)	0.00 (0.00)
FBRL	0.94 (0.04)	**1.00 (0.00)**
R3L	0.96 (0.04)	0.54 (0.18)
VaPRL	**0.98 (0.02)**	0.94 (0.05)
MEDAL	**0.98 (0.02)**	**1.00 (0.00)**

Figure 6.4: Average return of the final learned policy for various algorithms

Although MEDAL does not outperform any other algorithms, it matches the performance of the best performing algorithm for both the TableTop and Sawyer-Door environment indicating that it is a SOTA ARL algorithm. Sharma Ahmad & Finn (2022) also compare MEDAL to imitation learning algorithms since both have access to prior demonstration data. We use the Deployed Policy evaluation equation shown in Figure 6.1 to evaluate MEDAL compared to various imitation learning algorithms.

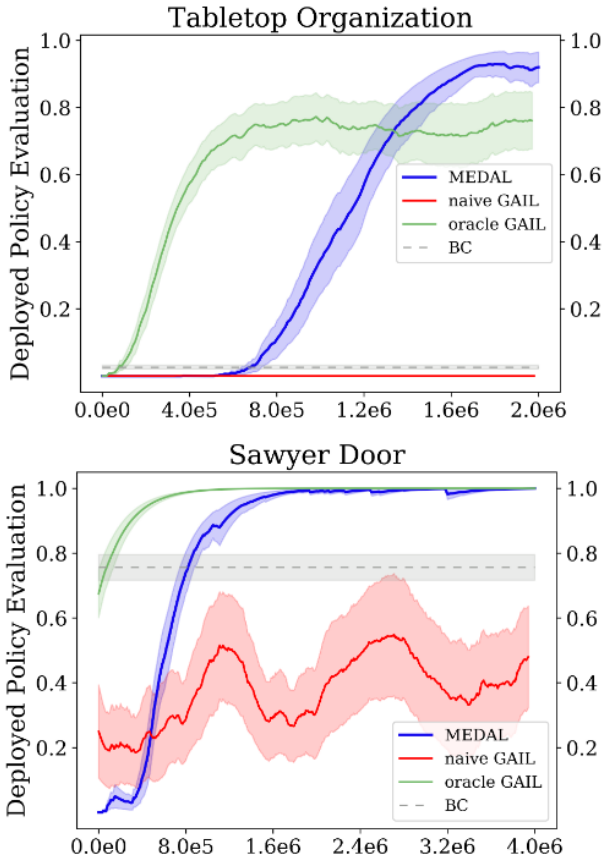

Figure 6.5: A Comparison of MEDAL vs Imitation learning algorithms

Imitation learning algorithms such as Behaviour Cloning as well as GAIL are commonly used applied to agents learning in an episodic setting. Oracle GAIL shows the Policy evaluated in an episodic setting. Naive GAIL and Behaviour Cloning shows the policy in a non-episodic setting. Figure 6.5 shows that both standard imitation learning algorithms do not perform well in non-episodic settings since the only Imitation learning algorithm that performed well was Oracle GAIL which is evaluated under episodic settings. Figure 6.5 visualises how MEDAL outperforms all Imitation learning algorithms in a non-episodic setting. MEDAL is therefore a powerful ARL algorithm that should be considered, not only in the context of ARL, but in the development of Intelligent RL agents since it can perform well in non-episodic environments which better reflect reality.

Chapter 7

Single-Life Reinforcement Learning

Humans in reality are often required to complete many tasks on a single try without knowing what the best course of action to complete that task is. This formalises the Single-life reinforcement learning Problem (SLRL) first described by Chen et al (2022). In order to create an agent with human-like intelligence it must be able to complete tasks under human conditions.

Chen et al (2022) describes an algorithm that can solve the SLRL problem known as QWALE. The core challenges of creating intelligent RL agents is that they must learn how to adapt and overcome novel states without entering any terminal states. This algorithm also allows agents to accomplish a sequence of tasks without a soft reset to its initial state after each task is accomplished. This is extremely similar to reset-free continual RL, however, the key difference is that the agent in SRL does not need to return to its' initial state before accomplishing its' next task. The difference between Reset-free continuous RL, Episodic RL and Single-Life RL is shown below in Figure 7.1.

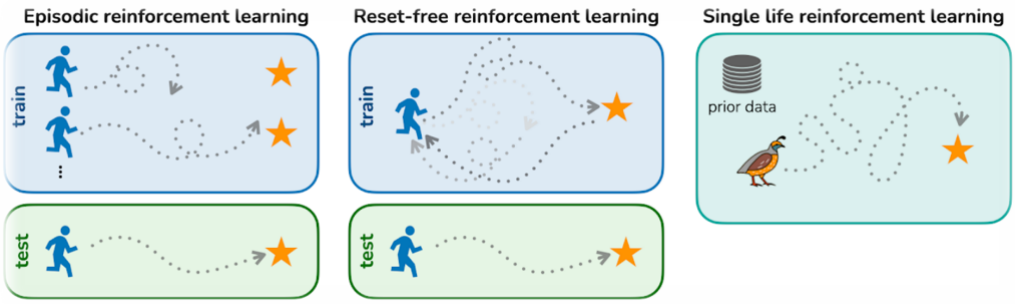

Figure 7.1: Difference between Single Life RL and other RL algorithms (Chen et al, 2022)

Figure 7.1 illustrates how the behaviour of Single-Life RL agents more accurately reflects the behaviour of humans by continuously adapting to new tasks. Single-Life Reinforcement Learning (SLRL) utilises an algorithm known as Q-weighted Adversarial Learning (QWALE) that is an extension of GAIL.

7.0.1 Q-Weighted Adversarial Learning

Q-Weighted Adversarial learning is a variation of GAIL specifically to help agents recover from out-of-distribution states (Chen et al, 2022). It similarly utilises state-action distributions from a demonstrator that knows how to accomplish the task. However, if we assume that our expert is not indeed an expert and its' trajectory is only near-optimal, as is often the case, there is no need to match our trained agents state-action distribution ($p^{\pi}(s,a)$) with the entire state-action distribution of the demonstrator ($p^{*}(s,a)$). QWALE will instead only match the state-action pairs that contributed to the completion of the task. Our target state-action distribution p^{*}_{target} can thus be written as equation 7.1 where p_{β} represents some behaviour policy.

$$p^*_{target}(s,a) = p_\beta(s,a)exp(Q^{\pi_{target}}(s,a) - V^{\pi_{target}}(s)) \tag{7.1}$$

Recall that GAIL focuses on minimising the Jensen-Shannon Divergence (D_{JS}) between the state action distribution of the expert ($p^*(s,a)$) and our agent ($p^\pi(s,a)$). If we substitute ($p^*(s,a)$) for $p^*_{target}(s,a)$, then minimising D_{JS} between the desired state-action distribution and the agents can be written as equation 7.2 where $\mathbb{E}_{s,a\approx p^\beta}[exp(Q^{\pi_{target}}(s,a) - V^{\pi_{target}}(s))]$ is a constant and can thus be ignored

$$min_\pi D_{JS}((p^\pi(s,a))||(p^*(s,a)) = min_\pi max_C \mathbb{E}_{s,a\approx p^\beta}[exp(Q^{\pi_{target}}(s,a) - V^{\pi_{target}}(s))logD(s,a)] + \mathbb{E}_{s,a\approx p^\pi}[log(1-D(s,a))] \tag{7.2}$$

QWALE utilises a Q-function $Q(s,a)$ trained in some training environment that can distinguish useful and noisy transitions in the prior data (Chen et al, 2022). The Q-function can be trained using any sort of method such as Monte-Carlo Estimations or other offline RL algorithms that are beyond the scope of this report. Since QWALE is a variation of GAIL, it still utilises GAN architecture consisting of a generator and a discriminator. The QWALE- discriminator is known as a Q-weighted Discriminator

The Q-Weighted Discriminator is described as such since it weights all positive states with a factor $exp(Q(s,a) - b)$ where b is the bias term. Since introducing a random bias will lead to odd behaviour, we assume the bias is the value of the most recent state $Q(s_t, a_t)$. QWALE, similar to GAIL, trains both the Generator and Discriminator in an alternating order in order to finetune both the policy and its' critic. it should be noted thay the Q-values are normalised between 0 and 1 during training.

Algorithm 1 Q-Weighted Adversarial Learning (QWALE)

1: // Single Trial Deployment
2: **Require:** $\mathcal{D}_{\text{prior}}$, test MDP $\mathcal{M}_{\text{test}}$, pretrained critic $Q(s,a)$, and (optionally) policy π;
3: **Initialize:** replay buffer for online transitions $\mathcal{D}_{\text{online}}$; parameters ϕ for discriminator $q_\phi(\text{prior} \mid s_t)$, timestep $t = 0$
4: **while** task not complete **do**
5: Take $a \sim \pi(\cdot \mid s_t)$, and observe r_t and s_{t+1}
6: $\mathcal{D}_{\text{online}} \leftarrow \mathcal{D}_{\text{online}} \cup \{(s_t, a_t, r_t, s_{t+1})\}$
7: $\phi \leftarrow \phi - \eta\nabla_\phi L(\phi)$ // Update discriminator according to Eq. 2
8: $r'(s_t) = r(s_t) - \log(1 - q_\phi(\text{prior} \mid s_t))$
9: $Q(s,a), \pi \leftarrow \text{SAC}(Q(s,a), \pi, \mathcal{D}_{\text{prior}} \cup \mathcal{D}_{\text{online}}, r')$
10: Increment t

Figure 7.2: QWALE Pseudocode (Chen et al, 2022)

QWALE utilises the Q-weighted discriminator as a sort of reward function that will guide the agent towards states that will lead to better outcomes compared to previous states according to its Q-values. The QWALE-Discriminator will prefer states with higher Q values which will be closer to the goal due to the integration of the Q-function. Since the Discriminator trains the policy in GAIL, this will result in a policy that tells agents to take actions to get to states with better Q values. Figure 7.2 shows how QWALE is implemented in practice. Before QWALE can be implemented, we must have some prior data (suboptimal demonstrator trajectories), an MDP to test the agent ($\mathbb{M}_{\approx\sim\approx}$) and a pretrained critic which is the Q-function $Q(s,a)$. Step 5 and 6 represent the agent interacting with the environment and adding those interactions to the replay buffer. Step 7 is the optimization step where we finetune the parameters ϕ of discriminator q_ϕ that discriminates between prior data and current states.The parameters ϕ will be updated using a cross-entropy loss function shown in equation 7.3 **??**

$$L(\phi) = -\mathbb{E}_{D_{\text{prior}}}[\exp(Q(s,a) - b)\log q_\phi(prior|s)] - \mathbb{E}_{D_{online}}[\log q_\phi(online|s)] \tag{7.3}$$

The discriminator will modify the reward from $r(s_t)$ to $r'(s_t)$ which is then used to update the critic $Q(s,a)$ using the Soft Actor Critic Algorithm. This loop of updating q_ϕ and $Q(s,a)$ is done iteratively until the the

task is complete. This concludes our description of QWALE in theory. Chen et al (2022) implements QWALE in order to complete tasks in different environments. We focus on the results of QWALE in the Tabletop and Halfcheetah Environments shown in Figure 7.0.1.

Figure 7.3: Tabletop Environment: Learning to move a mug

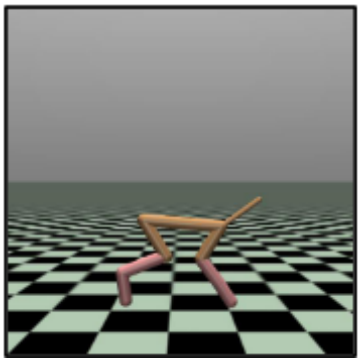

Figure 7.4: Cheetah Environment: Learning how to walk

The task in a Tabletop environment is to move the mug to the goal position which is a common task in RL. The task in the cheetah environment is to train the cheetah to run a certain distance. The performance of QWALE compared to other SOTA methods such as GAIL and SAC is shown belown in Figure 7.5. It should be noted that QWALE takes 20%-40% less steps than the next best algorithm to complete a task. For both environments, QWALE outperforms GAIL and SAC by the number of successes and average cumulative reward.

	Method	Avg ± Std error	Success / 10	Median		Method	Avg ± Std error	Success / 10	Median
Tabletop	GAIL-s	83.2k ± 23.8k	8	75.6k	Cheetah	GAIL-s	99.2k ± 23.0k	7	77.4k
	GAIL-sa	61.5k ± 28.7k	7	**2.4k**		GAIL-sa	102.0k ± 19.3k	8	85.6k
	SAC	123.7k ± 25.5k	7	157.2k		SAC	128.9k ± 23.9k	5	138.2k
	SAC-RND	94.8k ± 26.9k	7	51.4k		SAC-RND	158.4k ± 20.5k	3	200.0k
	QWALE (ours)	**44.4k ± 24.6k**	**9**	8.9k		QWALE (ours)	**74.3k ± 9.5k**	**10**	**68.9k**

Figure 7.5: A Comparison of QWALE to other algorithms(Chen et al, 2022)

One of the benefits of QWALE is its ability to adapt to novel situations, which were implemented in the form of moving the original position of the mug and putting hurdles in the cheetah's path. The performance of QWALE in novel situations is not explicitly evaluated by Chen et al (2022), however the performance of a cheetah trained using QWALE in a novel environment can be found https://sites.google.com/stanford.edu/single-life-rl

QWALE is a novel algorithm that can be used to train agents to adapt to new environments without any sort of hard resets, thus solving the Single-life RL problem. This algorithm is not however, without flaws since it, like many other algorithms, is not able to complete the assigned task with a 100% success rate. QWALE also requires a pretrained Q-function as well as a pretrained policy in some cases which can be difficult to find for novel environments. Overall, QWALE is a powerful algorithm that furthers development of RL agents that possess human-like intelligence

Chapter 8

Closing Remarks

8.1 Discussions

Developing intelligent agents that can perform a variety of tasks in novel environments is important since intelligent agents can be used to improve current RL applications such as self-driving cars as well as robotic manipulations. As mentioned by Legg (2021), the development of Intelligent agents must begin by formalising realistic environments. The Ecological RL Framework focuses on implementing features of reality into RL environments whilst alleviating the challenges agents encounter when learning in more realistic human-like environments. Co-Reyes et al (2022) illustrated that features of reality such dynamism and continuity can be integrated into RL environments to produce more realistic human-like RL environments that agents can interact with and learn from.

Another one of the core problem in developing intelligent agents is finding algorithms that can help agents learn in complex dynamic environments. We have outlined Deep RL algorithms that utilises neural networks to train agents as a solution. However these agents fail to generalise and perform novel tasks in new environments indicating that although an agent can utilise a brain-like structure, it will not exhibit human intelligence. We then move onto Imitation learning, which focuses on developing agents that learn via imitative learning, a process exhibited by animals and humans. Imitation learning utilises concepts from Deep RL as well as other fields of Machine learning to produce algorithms that can learn by copying an expert similar to how humans learn. Imitative agents can perform tasks similar to those demonstrated by the expert, however, standard imitation learning method tend to fail when implemented in non-episodic environments and are thus unsuitable to use in developing intelligent agents. We do, however, discuss a variation of an imitative algorithms known as QWALE as well as a similar but non-imitative algorithm known as MEDAL since both exhibit potential in training and producing intelligent RL agents.

MEDAL is an algorithm developed in the Autonomous RL framework that focuses on teaching an agent how to learn in a continual setting where there are no resets. MEDAL can successfully train agents to complete tasks in a non-episodic setting which is one of the key features of human-like environents. QWALE, also known as Q-weighted adversarial learning is a variation of GAIL that is shown to be able to adapt to novel environments in a continual setting. QWALE satisfies one key criteria of intelligent agents, namely that it can adapt to new environments. Agents trained by QWALE, cannot however, generalise and learn to complete novel tasks in these new environments indicating that it does not (yet) produce intelligent agents.

8.2 Future Work

This report has provided a brief overview of topics and frameworks in RL that have been attempting to solve the challenges associated with producing intelligent agents. This paper does omit any implementations of the described algorithms and their subsequent result. In section ... we create an implementation of a Human-like RL environement that is compatible with many RL algorithm packages. We also link a github repository where the Deep RL algorithms are applied to both environments described in section ... Future works should formalise the results obtained from implementing the Deep RL algorithms in the Eco-RL environment. MEDAL and QWALE are two promising algorithms that can develop intelligence in agents known as MEDAL and QWALE. Since Figure 6.5 illustrates that MEDAL performs better than GAIL, MEDAL should be compared to QWALE to see which one is better since QWALE is a variation of GAIL designed for non-episodic tasks. In addition, MEDAL and QWALE should be implemented in the Eco-RL environment to see how it performs in more realistic human-like environments.

8.3 Conclusions

The developments to creating intelligent agents in RL has produced several segmented solutions. There are algorithms that can solve one of the problems of intelligent agents but not all. However, given the rate of advancement in RL research as well as the performance of the algorithms described in this report, it is likely that intelligent agents will be produced sooner rather than later.

Glossary

- **Entropy** - A measure that quatifies the uncertainty in a probability distribution, reflecting the randomness or unpredictability of the system.

- **Jensen-Shannon Divergence** -Symmetric measure of the similarity between two probability distributions. It is derived from the Kullback-Leibler Divergence

- **Kubleck-Lieber Divergence**- Measure of how one probability distribution diverges from a second, expected probability distribution. It is used to quantify the information lost when one distribution is used to approximate another.

- **Objective Function**- Mathematical function that measures the performance of a model or algorithm. Also known as a loss function or cost function.

- **Partition Function**- Normalization constant that ensures that the sum or integral of probabilities over all possible states equals 1. It is crucial for defining a valid probability distribution

- **Turing Machine**- Abstract mathematical model of computation proposed by Alan Turing. It consists of an infinite tape, a read/write head, and a set of rules that govern the machine's transitions between states. Turing Machines serve as a foundation for understanding computability and complexity in theoretical computer science.

- **State Coverage** -The extent to which a learning agent explores and encounters different states in its environment during training.

- **Value Function**-Estimates the expected cumulative reward that an agent can obtain from a given state or state-action pair.

- **Q-value**- Q-value, or action-value function, represents the expected cumulative reward an agent can achieve by taking a specific action in a given state and following a certain policy. I

- **Q-function**- A function that determines the Q-value

Chapter 9

References

In order of appearance

- Burns, E. (2022). *What is artificial intelligence (AI)?* [online] TechTarget. Available at: https://www.techtarget.com/searchenterpriseai/definition/AI-Artificial-Intelligence.
- Demir, N. (2023). *Can we develop AGI with reinforcement learning? — A summary of 'Reward is enough' paper.* [online] blog.datagran.io. Available at: https://blog.datagran.io/posts/can-we-develop-agi-with-reinforcement-l [Accessed 16 Nov. 2023].
- Brooks, R. (2021). *What is reinforcement learning?* [online] University of York. Available at: https://online.york.ac.uk/what-is-reinforcement-learning/.
- Legg, S. (2021). *Ecological Theory of Reinforcement Learning: How Does Task Design Influence Agent Learning?* [online] neurips.cc. Available at: https://neurips.cc/virtual/2021/workshop/21865 [Accessed
- Co-Reyes, J.D., Sanjeev, S., Berseth, G., Gupta, A. and Levine, S., 2020. Ecological reinforcement learning. arXiv preprint arXiv:2006.12478.
- Miyoung Han. (2018) Reinforcement Learning Approaches in Dynamic Environments. Databases [cs.DB]. Télécom ParisTech, 2018. English. NNT : tel-01891805
- Sutton, R.S. and Barto, A. (2018). *Reinforcement learning : an introduction.* Cambridge, Ma ; Lodon: The Mit Press.
- Miller, T. (2022). *COMP90054: Reinforcement Learning — Introduction to Reinforcement Learning.* [online] gibberblot.github.io. Available at: https://gibberblot.github.io/rl-notes/intro.html.
- Goodfellow, I., Bengio, Y. & Courville, A. 2016. Deep learning. 1st ed. Cambridge (EE. UU.): MIT Press.
- François-Lavet, V., Henderson, P., Islam, R., Bellemare, M.G. and Pineau, J. (2018). An Introduction to Deep Reinforcement Learning. *Foundations and Trends® in Machine Learning*, 11(3-4), pp.219–354. doi:https://doi.org/10.1561/2200000071.
- Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D. and Riedmiller, M., 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602.*
- Wang, Yuhong Chen, Lei Zhou, Hong Zhou, Xu Zheng, Zongsheng Zeng, Qi Jiang, Li Lu, Liang. (2021). Flexible Transmission Network Expansion Planning Based on DQN Algorithm. Energies. 14. 1944. 10.3390/en14071944.
- Li, Y., 2017. Deep reinforcement learning: An overview. *arXiv preprint arXiv:1701.07274.*

- Simonini, T. (2022). *Proximal Policy Optimization (PPO).* [online] huggingface.co. Available at: https://huggingface.co/blog/deep-rl-ppo.
- Schulman, J., Wolski, F., Dhariwal, P., Radford, A. and Klimov, O., 2017a. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347.*
- Schulman, J., Klimov, O., Wolski, F., Dhariwal, P. and Radford, A. (2017b). *Proximal Policy Optimization.* [online] openai.com. Available at: https://openai.com/research/openai-baselines-ppo.
- Over, H., Carpenter, M. (2012). Imitative Learning in Humans and Animals. In: Seel, N.M. (eds) Encyclopedia of the Sciences of Learning. Springer, Boston, MA. https://doi.org/10.1007/978-1-4419-1428-6_270
- Smartlab AI, (2019). *A brief overview of Imitation Learning.* [online] Medium. Available at: https://smartlabai.medium.com/a-brief-overview-of-imitation-learning-8a8a75c44a9c.
- Ahmed Hussein, Mohamed Medhat Gaber, Eyad Elyan, and Chrisina Jayne. 2017. Imitation learning: A survey of learning methods. ACM Comput. Surv. 50, 2, Article 21 (April 2017), 35 pages
- Stanford. (2023). Available at: https://web.stanford.edu/class/cs237b/pdfs/lecture/cs237b_lecture_12.pdf.
- Adams, S., Cody, T. & Beling, P.A. A survey of inverse reinforcement learning. *Artif Intell Rev* **55**, 4307–4346 (2022). https://doi.org/10.1007/s10462-021-10108-x
- Hussain, N. (2023). *Maximum Entropy Inverse Reinforcement Learning.* [online] Available at: https://www.pair.toronto.edu/csc2621-w20/assets/slides/lec8_maxentirl.pdf [Accessed 16 Nov. 2023].
- Piot, B., Geist, M. and Pietquin, O., 2013, May. Boosted and reward-regularized classification for apprenticeship learning. In *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems* (pp. 1249-1256).
- Bau, D., Zhu, J.Y., Strobelt, H., Zhou, B., Tenenbaum, J.B., Freeman, W.T. and Torralba, A., 2018. Gan dissection: Visualizing and understanding generative adversarial networks. *arXiv preprint arXiv:1811.10597.*
- Ho, J. and Ermon, S., 2016. Generative adversarial imitation learning. *Advances in neural information processing systems*, *29.*
- Ermon, S. (2023) 'Generative Adversarial Imitation Learning', *CVPR18. By Stanford University*, Berkeley: University of California, 16 November.
- Sharma, A., Xu, K., Sardana, N., Gupta, A., Hausman, K., Levine, S. and Finn, C., 2021. Autonomous reinforcement learning: Formalism and benchmarking. *arXiv preprint arXiv:2112.09605.*
- Sharma, A., Ahmad, R. and Finn, C., 2022. A state-distribution matching approach to non-episodic reinforcement learning. *arXiv preprint arXiv:2205.05212.*
- Chen, A., Sharma, A., Levine, S. and Finn, C., 2022. You only live once: Single-life reinforcement learning. *Advances in Neural Information Processing Systems*, *35*, pp.14784-14797.