DISSERTATION

Improving central value functions for cooperative multi-agent reinforcement learning

Author:
Siddarth SINGH (1156261)

Supervisor:
Prof. Benjamin ROSMAN

UNIVERSITY OF THE
WITWATERSRAND,
JOHANNESBURG

submitted to
the Faculty of Science, in fulfilment of the requirements for the degree of
Msc Computer Science

in the

September 1, 2022

Declaration of Authorship

I, Siddarth SINGH (1156261), declare that this dissertation titled, "Improving central value functions for cooperative multi-agent reinforcement learning" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this dissertation has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this dissertation is entirely my own work.
- I have acknowledged all main sources of help.
- Where the dissertation is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: 

Date: 01/09/2022

UNIVERSITY OF THE WITWATERSRAND, JOHANNESBURG

Abstract

Faculty of Science
School of Computer Science and Applied Mathematics

Msc Computer Science

Improving central value functions for cooperative multi-agent reinforcement learning

by Siddarth SINGH (1156261)

Central value functions (CVFs) are methods which use a shared centralised critic to decompose the global shared reward in the cooperative settings into individual local rewards. CVFs are an effective method for value decomposition in multi-agent reinforcement learning problems. However many state-of-the-art methods are reliant on an easily defined ground truth state to perform credit assignment. These methods perform poorly in certain environments with high numbers of redundant agents. We propose a method called Relevance Decomposition Network (RDN) that makes use of layerwise-relevance propagation (LRP) as an alternative form of credit assignment that can better perform value decomposition with large numbers of redundant agents when compared to existing methods like Qmix and Value-Decomposition Network (VDN). Another limitation in the MARL space is that it has generally favoured Q-learning based algorithms. This can be attributed to the belief that due to the poor sample efficiency of on-policy learning they are ineffective in the large action and state spaces in the Multi-Agent setting. We make use of a small set of improvements that can be generalised to most on-policy actor-critic algorithms to accommodate a small amount of off-policy data to improve sample efficiency and increase training stability. We implemented our improved agent variants and test them in a variety of environments including the Starcraft multi-agent challenge (SMAC). Our proposed method was able able to greatly improve the performance of a basic naive multi-agent advantage actor-critic algorithm with faster convergence to high-performing policies and reduced variance in expected performance at all stages of training.

Contents

Chapter 1

Introduction

Consider the issue of optimising the overall travel times for multiple trains running simultaneously in a railway network. In this scenario, we would only be able to control the directions the trains can move using the switches on the track from a decentralised view based on the local observations of each train. In this problem we need to consider the heterogeneity of the different cooperative agents in the setting due to different models of trains in operation, scalability across thousands of active agents and how to accurately distribute a global reward across multiple agents. Currently, effective assessment to determine how optimal the schedule used in these systems is very difficult as they are controlled using handmade heuristics. Creating long-term planning schemes that must take into account the interactions of a large number of agents in complicated settings is difficult to perform optimally manually. Instead, MARL provides a potential avenue to automate these complex large-scale multi-agent tasks. In 2019 this prompted the *Aicrowd* Flatland challenge to encourage investigation into multi-agent reinforcement learning for the solving of complex railway simulations [1].

We consider a multi-agent setting where multiple potentially heterogeneous agents must work together to complete a cooperative task. A cooperative task is a task where all agents are awarded a shared reward upon completion and as such share a common goal. The agents in the environment are only able to view a limited space surrounding them (partial observability) and must rely on this local information to determine how to optimally contribute to the task. In the Flatland challenge this cooperative task is the minimization of the travel time for the last train to reach its destination.

Approaches prior to value-decomposition network (VDN) (Sunehag et al., 2018) attempt to learn a *joint actions* (Claus and Boutilier, 1998). One way to consider such a problem is by treating the actions of multiple agents together as *joint-actions*. While this is a successful approach in problems with small numbers of agents in a limited action space, this training method suffers from the curse of dimensionality due to the exponential scaling of the joint space in relation to the number of agents (Foerster, 2018).

A method to deal with the exponential growth of *joint-actions* is the use of value function decomposition using centralised value functions (CVFs) (Sunehag et al., 2018). A CVF is a critic network that learns to predict the joint reward in the environment at each timestep. After learning the joint reward the CVF is used to assign credit to the local agents to learn based on the local rewards decomposed from the joint reward. CVFs show a linear rather than exponential scaling of complexity when compared to learning over joint action spaces. CVFs also allow us to define a global reward and avoid using local shaped rewards. Local reward refer to the individual rewards assigned to each agent at each timestep from the environment.

[1] https://www.aicrowd.com/challenges/flatland-challenge

Often in single-agent RL to make learning the solution to the environment easier small rewards are assigned when smaller tasks are completed during each timestep. This is referred to as reward shaping (Sutton and Barto, 2018). Reward shaping can improve the speed at which agents learn but also require the use of human priors to assign and therefore can bias the policies learnt by agents during training (Grzes and Kudenko, 2009).

Methods using a CVF typically make use of a ground truth state representation as part of the critic input space ((Foerster et al., 2018; Rashid et al., 2018)). In complex environments constructing an accurate ground truth representation is a non-trivial task. An example of this is the Piano Movers problem (Schwartz and Sharir, 1983) when applied to the robotics setting. We can think of the problem in simple terms as having 2 robots being tasked to move a piano through a corridor where the corridor contains a bend requiring them to manoeuvre in coordination to pass. In such a case either we can determine come series of coordinates for the 2 robots to successfully carry the piano through the corridor or determine that there are no viable paths. An illustration of this is provided below in figure 1.1.

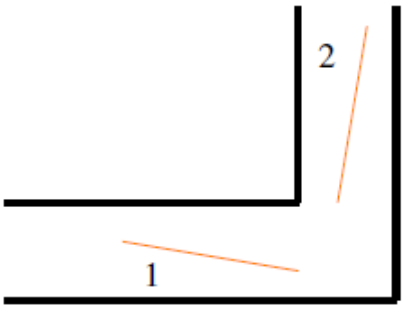
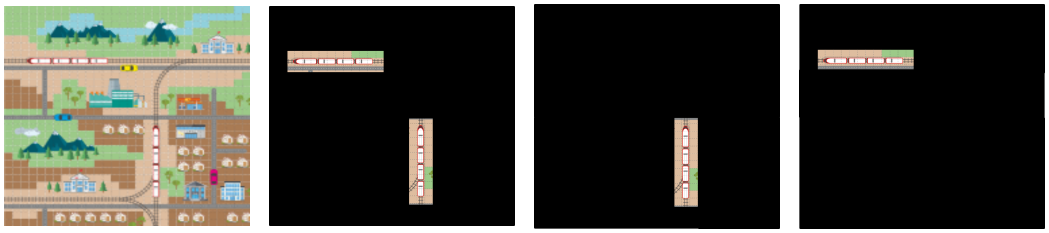


FIGURE 1.1: Diagram of the Piano Movers Problem. Showing viable to traverse the corridoe in orange.

If we were to carry out this task in real life we would be limited by the quality of the data obtained by the sensors and cameras on the robots. Typically these generate data that is noisy or of limited accuracy. This data is the local observations of the independent agents and is an egocentric local partial observation of the environment from the perspective of each agent. A concatenation of these local observations is still only a partial representation of the full state space as illustrated in figure 1.2 below.



(A) Fully observable ground truth space of environment

(B) Concatenation of local partially observable spaces

(C) Local partially observable space of agent 1 only

(D) Local partially observable space of agent 2 only

FIGURE 1.2: Example from left to right of a full observable space, a concatenation of local partially observable spaces and individual local partially observable agent spaces

and using most CVFs this results in sub-optimal performance for most algorithms (Samvelyan et al., 2019). Given the limitation of the observable space generated with information gathered from machines in real or realistic settings, it is not reliable to assume that an easily interpreted state approximation can be made from only these limited local observations. Often in these real-world applications, we are reliant on human intuition to determine the number of agents to deploy to solve the problem. Often this means more agents than required are assigned to the task which in certain cases can be detrimental to performance. In the case of the piano movers problem, these extra or redundant agents would reduce the navigable space in the corridor increasing the difficulty of the task. Optimally they would move out of the way of the essential agents and otherwise contribute nothing to the solving of the task. Even in this case, the redundant agents are still detrimental to solving the task. These redundant agents increase the size of the state space interpretation as the ground truth must not account for the non-essential agents. These larger state spaces are harder to interpret which makes learning the optimal policy more difficult during training. Therefore it is important to develop algorithms that can effectively separate agents that are essential to solving a task and agents that are redundant to allow MARL algorithms to be deployed into more realistic and eventually real-world challenges.

Within CVF algorithms there have been very few notable algorithms that make use of on-policy Policy Gradient (PG) methods, the most notable being COMA (Foerster et al., 2018) and IPPO/MAPPO (Yu et al., 2021). This is unfortunate as PG-based methods have been shown to have useful properties not utilised by off-policy Q-learning algorithms (Papoudakis et al., 2021). Most notably PG algorithms greatly outperform their Q-learning counterparts on sparse environments due to their stochastic policies allowing for a more effective exploration of the environment. When compared to Q-learning methods in more reward dense environments where there are multiple reward signals across a training episode PG algorithms were shown to have surprisingly poor performance (Samvelyan et al., 2019). This performance gap was attributed to the low sample efficiency of the PG methods which are not able to make use off-policy data so can only be trained on new data gathered using the current policy. Although more complex PG methods have been developed for MARL settings such as LICA (Zhou et al., 2020) they require up to 10x as many samples to achieve similar performance making them impractical to train until they reach comparable performance to Q-learning based methods.

In this dissertation, we explore the use of layerwise relevance propagation (LRP) (Montavon et al., 2019) as an alternative method to standard learned value decomposition and introduce 2 small modifications to the standard naive centralised actor-critic to have greatly improved sample efficiency. Standard learned value decomposition methods like VDN and Qmix have been shown to become unstable during training in environments with high numbers of redundant agents as they struggle to accurately decompose the global reward in cases where few agents have a proportionally high relevance to the success of the policy compared to others (Yang et al., 2020b). Network explainability methods have been used with success in RL to train agents with delayed rewards (Arjona-Medina et al., 2019). Rather than redistribute a delayed reward across time to produce a less sparse reward signal we instead propose using network explainability to redistribute the joint reward in a cooperative MARL setting across the independent agents in the environment. We

propose using LRP to decompose the value function learnt by a joint critic to assign credit to local agents based on the joint global reward in a method called Relevance Decomposition Network (RDN). This method can more effectively accommodate the redundant agents in an environment to learn more stable policies than both Qmix and VDN. For the centralised actor-critic rather than training from purely on-policy data, we maintain a small replay buffer containing data from the previous 4 to 8 training epochs under the assumption that between training epochs the overall policy changes are small enough that we can obtain greater sample efficiency without instability typically associated with using off-policy data in a standard advantage actor-critic (Wang et al., 2016). To enforce update limitation between epochs we also introduce a KL divergence mask which measures the difference between a weighted average of all previous policies and the current policy. Any timesteps in the mask exceeding the predefined KL divergence hyper-parameter are not used during training. This method called Extended-Masked Centralised Actor-Critic (EM-CAC) shows greatly improved sample efficiency and stability over the standard centralised actor-critic and shows that if properly adapted to the MARL setting the sample efficiency difficulties of PG algorithms can be minimised.

We evaluated our proposed methods on 2 domains the *Starcraft multi-agent competition* and a 2 step matrix game. Our results show that Qmix and VDN do experience reductions in performance as the number of redundant agents is increased and even in a simple matrix game VDN and Qmix are prone to converging on sub-optimal strategies due to inaccurate decomposition of the global reward. Our method RDN is largely unaffected by the redundant agents and can converge to a perfect policy in the matrix game. The testing environments are the *Starcraft multi-agent competition* and a 2 step matrix game. EM-CAC can consistently out perform the CAC model and not only converge faster and more stably to better policies but is also able to solve environments that naive CAC cannot.

1.1 Dissertation Outline

In chapter 2 we introduce some of the basic concepts of reinforcement learning (RL), multi-agent reinforcement learning (MARL) and network explainability methods.

In chapter 3 we explain the research problem and provide the significance and motivation behind our research, our research aims and objectives and our primary research questions.

In chapter 4 we introduce our proposed algorithms and explain the methodology to be used. This includes software used in the implementation, the data to be collected, our selection of baseline algorithms and, our algorithms.

In chapter 5 we evaluate the results of our experimentation. This section is split by our different algorithms with a breakdown of the results and an ablation study of each method.

In chapter 6 we conclude out findings and summarise the results discovered over the course of the experimentation process.

1.2 Contributions

The work presented in this dissertation is intended to generally improve CVF algorithms for MARL in the cooperative setting. We have identified two gaps in the current literature: The issues of current Q-learning based value factorisation methods in effectively assigning credit to train agents in scenarios with a high number

of redundant agents and a lack of research into developing more sample efficient PG algorithms, which are naturally able to solve these problems due to their more effective exploration methods.

We resolve the first issue with the development of our own algorithm, which we call Relevance Decomposition Network (RDN) (Singh and Rosman, 2021). RDN uses Layerwise-Relevance Propagation (LRP) as a more effective method to perform credit assignment in cases with high number of redundant agents.

The second issue is resolved by introducing two small modifications into the naive centralised actor-critic to augment sample efficiency and stability during training called Extended-Masked Centralised Actor-Critic (EM-CAC). EM-CAC can solve the challenging *MMM2* environment in SMAC which the naive centralised actor-critic cannot.

Chapter 2

Background

In this chapter we provide the background information and formalisms for the rest of the dissertation. We introduce Reinforcement Learning (RL) in section 2.1 which describes the basics of RL including the Markov Decision Process (MDP), Partially Observable Markov Decision Process (POMDP), tabular RL methods and introduces the multi-agent setting. Section 2.2 covers Deep Reinforcement Learning (DRL) methods which expand RL to higher dimensional spaces using artificial neural networks. Section 2.3 applies DRL to the multi-agent setting and covers independent agents, value function decomposition and counterfactual baselines.

2.1 Reinforcement Learning (RL)

Reinforcement learning (RL) is a form of trial and error learning in which an agent attempts to maximise a numerical reward based on actions performed in an environment (Sutton and Barto, 2018). The agent is given some knowledge of its current state and then using that information takes an action in response to the *state* s. A *state* is an observation given to the agent by the environment, for example, in a video game, this could take the form of the image of a map. A reward is then given based on the success of the action. This continues in a loop until a terminal state is reached and the episode ends. An *episode* refers to the completion of an action or reaching some additional termination condition. This learning cycle is shown below in 2.1.

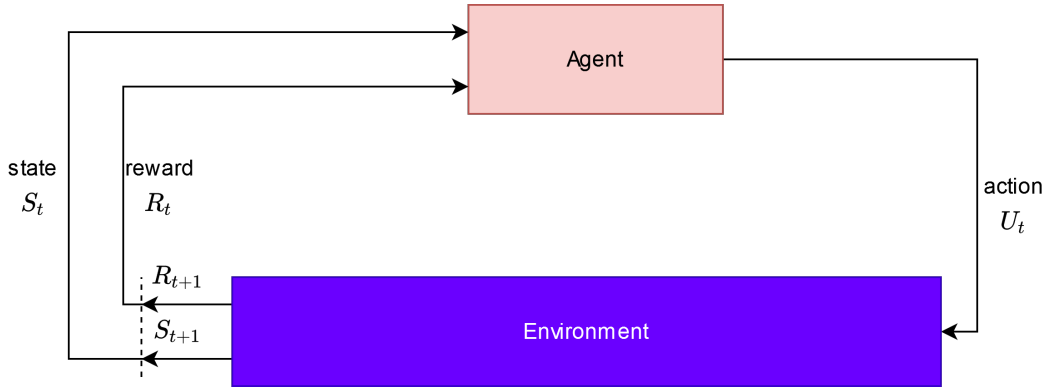


FIGURE 2.1: Reinforcement learning cycle

A *state* s_t is the input data representing the observations of the agent in the environment. For a strategy game where agents need to be aware of the data of multiple other agents, it could be represented as a vector containing the relative distance, health and types of enemies in observation range.

The *reward* is a signal received from the environment based on an action that is taken in the current state. We give an agent positive rewards to encourage behaviours and negative rewards to discourage behaviours. For example, an agent may receive +1 points for winning the game and -1 points for losing.

Actions are the how the agent interacts with the environment using some fixed set of interactions. In a game like chess, an action would be selecting the piece to move and the position on the board for it to be moved to.

When compared to supervised learning RL differs in that we typically do not learn a model by training the agent from a predefined dataset. Instead we aim to maximize the reward signal received by the agent. The reward r_t associated with each *state* s_t is learnt by iterative interaction with the environment. By learning this we can determine a policy $\pi(S)$ that maps states to actions to maximize the reward signal. An important consideration around this is the issue of delayed reward. Delayed reward refers to an action's ability to affect not only the immediate reward but rewards in future situations. Each action taken by the agent has a long-term effect on the future cumulative rewards of the policy aside from the immediate short-term gain.

2.1.1 Markov Decision Process (MDP)

RL problems are modelled as *Markov decision processes* (MDP). An MDP is a framework used to model decision-making in scenarios where outcomes are partly stochastic and partly under the control of a decision maker.

In an MDP we expect the state to have the Markov Property. This is the expectation that that state summarises and contains any information from prior states and only the current state is required to predict future states. Thus in an MDP it can be said that the future is independent of the past given the present. An example of this would be a checkers game where the current board state summarised all information required to determine the optimal action for the next turn as all prior actions are summarised in the current position of all pieces on the board (Sutton and Barto, 2018).

The learner which performs decision-making in the environment is called the agent. The environment compromises all things outside the agent which the learner interacts with. The agent interacts over a series of timesteps with the environment which responds to the agent's actions at each timestep and generates new situations based on the actions of the agent. Rewards are given by the environment to the agent over time to determine the agent's success towards accomplishing a task. A task is one instance of a reinforcement learning problem (Sutton and Barto, 2018). We focus on the discrete case of the MDP. In this case the agent and environment interact sequentially in a linear series of discrete timesteps. At each timestep t, the agent receives the representation of the environment's state, $s_t \in S$, where S is the set of possible states in the environment, and based on the information from the state the agent selects an action, $u_t \in U(s_t)$, where $U(s_t)$ is the set of discrete actions available in state s_t. After the action is performed the agent receives a numerical reward, $r_{t+1} \in R$ and transitions to a new state, s_{t+1}.

At each timestep, the agent performs a mapping from states to the probability of selecting each possible action at that state. This mapping is called the agent's policy and is denoted by π_t, where $\pi_t(u|s)$ is the probability of a specific action u being selected at state s. RL methods specify how the agent's policy is updated based on the results of its experience. The goal of an agent is typically to maximize the total

cumulative reward it receives from the environment over time (Sutton and Barto, 2018).

Formally the reward to be maximised for the agent is referred to as the expected return (Sutton and Barto, 2018), G_t where G_t is some function accumulating the rewards over each timestep. In the simplest case, the return is just the sum of the rewards over time $G_t = r_{t+1} + r_{t+2} + r_{t+3} + ... + r_T$ where T is the final timestep. This approach to return calculation is useful in applications with a final timestep where environment interaction has a finite end. This interaction from beginning to a finite end is an episode. Each episode ends in a terminal state which is a special end state that determines the end of interaction. After this terminal state the environment is reset with the agent returning to the starting state or to a starting state selected from some distribution of starting states.

Another important concept for the calculation of returns is discounting (Sutton and Barto, 2018). For this approach to returns, the agent tries to select actions to maximise the discounted rewards received in future are maximised. The discounted return is $G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$ where γ is the discount rate which is $0 \leq \gamma \leq 1$. The discount value of future rewards at the present timestep. If $\gamma = 0$ then the agent is only concerned with maximising immediate rewards and learns to choose u_t only to maximise r_{t+1}. An agent maximising only immediate rewards can maximise the discounted return by maximising each immediate reward however, only maximising immediate rewards reduces the estimated value of earlier rewards making it difficult to learn policies with long-term planning. As γ becomes closer to 1 the discounted return places more importance on future rewards and allows the agent to learn more farsighted policies.

RL algorithms usually require estimating value functions (Sutton and Barto, 2018). They are functions of state or state-actions pairs that estimate how good it is for the agent to be in the given state or how good it is to perform a specific action in a given state. "How good" a state or state-action pair is is defined by the expected future returns that can be obtained. The reward an agent would obtain is dependent on the action they will take. Following this, value functions are defined with respect to their associated policies. We can therefore define the value of a state s under the policy π as $v_\pi(s)$ where $v_\pi(s)$ is the expected return when starting in s and following π for all steps after. Formally $v_\pi(s)$ is defined as:

$$v^\pi(s) = E_\pi\{r_t | s_t = s\} = E_\pi\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s\} \tag{2.1}$$

Where E_π denotes the expected value given that the agent follows policy π, and t is at any timestep. The value of the terminal state if one is present is always zero. The function v_π is called the *state-value function for policy* π

Similarly, we define the value of taking action a in state s under a policy π as $q_\pi(s, a)$, as the expected return starting from s, taking the action a, and following policy π for all future steps:

$$q^\pi(s, u) = E_\pi\{R_t | s_t = s; u_t = u\} = E_\pi\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s; u_t = u\} \tag{2.2}$$

The function q_π is called the *action-value function* for policy π (Sutton and Barto, 2018).

In an MDP we consider an optimal policy to be one that maximizes the total amount of rewards it obtains from the environment and that is redundant from

maximising rewards, meaning that there is no state from which any other policy can achieve a better sum of discounted rewards (Sutton and Barto, 2018). Every MDP has at least one optimal policy and at least one that is stationary and deterministic (Sutton and Barto, 2018). Therefore for any MDP there is a policy π that maps : $S \rightarrow U$. The policy is referred to as being stationary as it does not change as a function of time and is deterministic since the same action is always chosen whenever the agent is in state s for all $s \in S$. There may be more than one optimal policy in an MDP however, we denote all optimal policies by π_*. All optimal policies share the same state-value function, called the *optimal state-value function* which is denoted by v_*. Optimal policies also share the same *optimal action-value function* denoted by q_*.

Partially Observable Markov Decision Process (POMDP)

In many real-world applications, the agent only has access to a partially observable state space where not all state information is visible (Hausknecht and Stone, 2015). These partially observable domains can be modelled as Partially Observable Markov Decision Processes (POMDPs). The tuple for the POMDP is shown below (Kaelbling et al., 1998):

$$POMDP = \langle S, U, r, p, \Omega, O, \gamma \rangle \tag{2.3}$$

Where S is a discrete set of states, U is a discrete set of actions, and $p : S \times U \rightarrow \Delta$ is the transition function, where Δ is the set of probability distributions over the state space S, $r : S \times U \rightarrow R$ is the reward function of the agent, Ω is a set of observations from the environment, O is a probability distribution over observations for each state and actions and γ is the discount factor used to weight future rewards where $\gamma \in [0, 1)$. In the POMDP setting due to partial observability states can look similar based on the incomplete view of the state (Cassandra, 1998). Therefore in the POMDP setting, we must learn O to perform inference to determine which state the agent is in based on interactions with the environment.

Temporal Difference (TD) Learning

Temporal-Difference (TD) (Sutton and Barto, 2018) learning is a method of reinforcement learning that is capable of learning directly from raw experience without model dynamics (model-free)and is able to update estimates without using full episode rollouts (Singh, 1995). Its simplest form, TD(0) is an update that is calculated over only one timestep and is defined below:

$$V(S_t) \leftarrow V(s_t) + \alpha[R_{t+1} + \gamma V(s_{t+1}) - V(s_t)] \tag{2.4}$$

Where V is the expected value of the state s at timestep t $s_t \in S$ is the state at time, $r_t \in R$ the rewards in time t and $\alpha \in$ [0,1] and $\gamma \in$ [0,1] are constant parameters. The parameter α is the learning rate and determines the extent to which the new experience is able to override the estimates made by the older policy. The second parameter γ is the discount factor discussed above.

Q-Learning

Q-Learning (Sutton and Barto, 2018) is a form of model-free reinforcement learning meaning it does not require a model of the environment's transition function in order to learn. It iteratively learns the optimal Q-value function for the MDP. The update

rule takes the form of an iterative Bellman update and is shown below:

$$Q(s_t, u_t) \leftarrow (1 - \alpha_t)Q(s_t, u_t) + \alpha_t[R(s_{t+1}) + \gamma \max_u Q(s_{t+1}, u)] \tag{2.5}$$

Where $Q(s_t, u_t)$ is the state-action value at timestep t, α_t is the learning rate, $r(s)$ is the reward signal from the current state s, γ is a term used to weight the value of future rewards and $\max_u Q(s_{t+1}, u)$ is the expected value of the state-action pair with the highest value in the next timestep $t + 1$.
The Q-learning update is done under the assumption of the agent choosing the action with the highest expected Q-value at the evaluated state. This update does not rely on the agent's actual policy to determine the action chosen. Due to this, we can perform this update independently of the actual policy of the agent. Algorithms capable of being updated in this manner are off-policy algorithms (Sutton and Barto, 2018).

2.1.2 Multi-Agent Settings

In this dissertation, we consider Reinforcement Learning (RL) in multi-agent (MA) settings. Learning an MA policy for an RL agent introduces new complexities to the problem of training an agent using RL to learn an optimal policy. Agents in this MA environment must now interact not only with the environment but simultaneously with each other (Busoniu et al., 2008). This inter-agent interaction during the learning process renders the *Markov property* (the future dynamics, transitions and rewards depend only on the current state) invalid as the environment is no longer stationary ((Matignon et al., 2012; Nowé et al., 2012; Tuyls & Weiss, 2012)).

Cooperative, Zero-sum and General-Sum Games

There are three types of multi-agent settings called games (Fudenberg and Tirole, 1991). Depending on the type of multi-agent problem that needs to be solved the how the reward signal is produced and the difficulties of agent interaction change.

Cooperative refers to a setting in which multiple agents collaborate to receive a joint cooperative reward, which is called a team-reward. The difficulty in this setting is from multiple agents having to learn to effectively collaborate with respect to a reward signal that is uniformly assigned across all agents.

Zero-sum is the opposite, where the rewards summed across all agents are 0 across all states. Essentially the reward won by one agent is lost by another. The difficulty in this setting is due to stability issues arising from agents maximising with respect to a constantly changing opponent policy which destabilises training.

Most complex settings require a mixture of cooperative and adversarial behaviours. These are called *general-sum* settings. In this setting, the primary difficulty is due to agents needing to decide when collaborative or adversarial behaviour is advantageous.

Each setting provides a different challenge. An environment with partial observability under a cooperative setting allows us to learn and test communication protocols. For a zero-sum setting, learning may be destabilised due to agents optimising against each other's policies. In a general-sum setting agents will need to learn when collaboration is required over adversarial behaviours (Foerster, 2018).

Dec-POMDPs

Dec-POMDPs generalise POMDPs to the multi-agent decentralised setting (Oliehoek, 2012). In this setting multiple agents act under uncertainty based on partial observations of the world. Agents execute actions under limited local state observations. At each timestep agents perform actions in parallel with each agent obtaining a new local observation along with a global reward in the case of cooperative settings. For general-sum or zeros-sum settings, agents would receive a local reward based on individual performance. Regardless of the type of game, execution is decentralised as agent actions are based on local observations not on the global true state.

Formally, a Dec-POMDP can be defined by the tuple $\langle A, S, U_i, P, \Omega_i, O, \gamma, b_0 >$, where A is a finite set of agents; S is a finite set of states with a designated initial distribution b_0; U_i is a finite set of actions for each agent i with $U = \times_i U_i$ the set of join actions; P is the state transition probability function, $P : S \times U \times S \rightarrow [0, 1]$, that specifies the probability of transitioning from state $s \in S$ to $s' \in S$ when the joint actions $\vec{u} \in U$ are taken by the agents; R is a reward function: $R : S \times U \rightarrow \mathbb{R}$, the immediate reward for being in state $s \in S$ and taking the joint actions $\vec{u} \in A$; Ω_i is a finite set of observations for each agent, i, with $\Omega = \times_i \Omega_i$ the set of joint observations; O is an observation probability function: $O : \Omega \times U \times S \rightarrow [0, 1]$, the probability of seeing joint observations $\vec{o} \in \Omega$ given joint actions $\vec{u}$ were taken which results in the state $s' \in S$; and γ is the discount function which weighs the value of future and past rewards. To solve Dec-POMDPs we must find a *joint policy*. This is a set of policies mapped to individual agents. Additionally, as agents are not able to directly observe the global state it is a requirement to learn a history of observations. Using this history an agent can map a local policy to its local POMDP. Unlike single-agent systems, we cannot usually estimate the hidden state from individual agent observation histories as it is no longer static and becomes dependent on the behaviours of all agents in the joint system (Foerster, 2018).

Centralised Training, Decentralised execution (CTDE)

In fully observable settings it is possible to learn a centralised controller, $\pi^C(\vec{u}|s_t)$, that maps from states into a probability distribution over *join actions*, $\vec{u}$ (Foerster, 2018):

$$\pi^C(\vec{u}|s_t) : \mathbf{U} \times S \rightarrow [0, 1] \tag{2.6}$$

This approach has two primary drawbacks. The joint action space, **U**, is exponential with the number of agents represented by the joint policy as the joint policy must be able to represent all combinations of the individual actions spaces of all agents it represents and, in many real-world settings, agents have to operate using only their local observations o^a which makes a centralised controller impractical.

Due to these drawbacks, this dissertation will only focus on *decentralised control*. In this setting each agent has a local policy, $\pi^a(u^a|s_t)$ which only maps from states or local observations to the probability distribution over individual actions for each agent. Rather than taking into consideration an exponential action space the agent only needs to consider its own action space and in the case of partially observable settings the agent policies can be conditioned on the local observations of each agent.

Although we learn a decentralised policy, we are still able to make use of information that would not normally be available to the agent for centralised training as long as it if not provided during execution time. This is the Centralised Training, Decentralised execution (CTDE) paradigm (Lowe et al., 2017). Usually, when training RL agents we make use of a simulator as the training environment. In these simulated environment,s it is possible to access data that an agent is not able to utilise

during execution purely for training (Samvelyan et al., 2019).

This can take the form of open communication channels or access to the ground truth state space. This allows us to make use of additional information that may aid in finding better policies during training time while reducing computational overhead during execution time.

CTDE is an effective training paradigm for MARL environments. It has been found to have good computational scaling with increasing numbers of agents (Sunehag et al., 2018). However, it faces issues in more complex cases. In certain cases where complex agent interactions arise or with greater numbers of heterogeneous agents it becomes more difficult to accurately assign credit for task completion. As task complexity increases and agent roles become less obvious, more agent types must interact with each other and, all the different interaction outcomes between them need to be accounted for in order to solve the environment. Heterogeneity refers to the number of different types of agents that must interact with each other in the environment.

2.2 Deep Reinforcement Learning (DRL)

Standard RL techniques for Q-learning use tables to store the value of all state-action pairs. This is not suitable for large state spaces as they must be either projected into a simpler state space creating inaccurate predictions or require unreasonable amounts of memory for practical applications. We instead make use of artificial neural networks (ANNs) as function approximators which replace the look-up table. The use of ANNs allows RL to be expanded to more complex domains that require processing high-dimensional data for state observations. RL has been expanded into spaces which use the pixel information data displayed on monitors. Convolutional Neural Networks (CNNs) can be used to perform feature extraction in image data to learn a more compressed representation of the data (Albawi et al., 2017). This compressed representation of the state space can be used to train agents for DRL to learn games from only pixel images more akin to how humans interpret data from video games (Mnih et al., 2013a). In environments where the agent does not have full visibility of the state space, it must use the history of its observations to infer the missing information. In practice, to maintain a representation of the agent's observation history through all timesteps recurrent neural networks are used to maintain an approximation of the hidden information at each timestep (Hausknecht and Stone, 2015).

Reinforcement learning presents its own challenges in the context of deep learning. In the standard supervised training methods used in tasks like computer vision a large amount of hand labelled training data is typically used. RL algorithms must instead learn from a reward signal that is often noisy or delayed. The delay between actions and resulting rewards can be thousands of timesteps making the association between the input and target values difficult to calculate. Additionally in RL the data being used to train the agent is both drawn from the environment and evaluated based on the agent policy. As the agent learns, the policy changes, which also changes the distribution the training data is drawn from. This is problematic for standard supervised learning which assumes the distribution of the training data is static (Mnih et al., 2013b).

Deep Q-Learning

A deep Q network (DQN) (Mnih et al., 2015) expands Q-learning into higher dimensional spaces using ANNs. It is notable as one of the first methods to use Deep Reinforcement Learning (DRL) effectively in the *Atari Learning Environment* (ALE) suite (Bellemare et al., 2013). The ALE suite is a testing suite that contains an assortment of different *Atari* games with an interface making them able to be played by RL agents and has been commonly used to benchmark DRL agents in different types of tasks.

For Deep Q-learning we use the same idea of an iterative Bellman update described in section 2.2.1 however, we instead make use of a non-linear function approximator in the form of an ANN to learn an approximate Q-value function $Q(s, u; \theta) \approx Q^*(s, u)$ where θ are the parameters of the NN (Bellemare et al., 2013). A diagram of the data flow of a standard DQN is shown in figure 2.2.

The Q-learning network can be trained by optimising a sequence of parameters θ_i using loss function L_i that updates the neural network(s) at each iteration i:

$$L_i(\theta_i) = E[(y_i - Q(s, u; \theta_i))^2] \tag{2.7}$$

Where $y_i = E_{s' \sim \mathcal{E}}[r + \gamma max_{u'} Q(s', u'; \theta_{i-1}) | s, u]$ is the target for iteration i and uses as the expected value for the given state and $p(s, u)$ is a probability distribution over states s and action a. The parameters from the previous iteration θ_{i-1} are held fixed when optimising the loss function $L_i(\theta_i)$. The targets depend on the weights of the NN and are not static in contrast to the fixed targets used in supervised learning. By differentiating the loss function with respect to the weights of the NN the gradient is shown as :

$$\nabla_{\theta i} L_i(\theta_i) = E_{s,u \sim p(.); s' \sim \mathcal{E}}[(r + \gamma max_{a'} Q(s', u'; \theta_{i-1}) - Q(s, u; \theta_i)) \nabla_{\theta i} Q(s, u; \theta_i)] \tag{2.8}$$

We then optimise this loss function using stochastic gradient descent. If the weights are updated after each timestep rather than updates in a batch, and the expectations are replaced by single samples from the behaviour distribution and the simulation environment $\mathcal{E}$ respectively then we have the standard Q-learning algorithm.

The standard DQN algorithm is *model-free*. The RL task is solved directly using samples from environment $\mathcal{E}$ without constructing an internal model of $\mathcal{E}$.

The DQN outputs a vector of Q-values for each action possible and the agent then performs the one with the highest estimated reward. Typically one-hot encoding is used to map each action possible for the agent to an encoding.

Experience Replay (ER)

An important component of the original DQN is the use of an experience replay buffer which is used to store data from the agent's interactions with the environment (Mnih et al., 2015). Typically this data is a tuple containing the state s_t observed by the agent, the action a_t the agent chose when visiting that state, the reward r_t received as a result of the action chosen and the previous state the agent transitioned from s_{t-1}. These tuples are stored in a replay buffer when the agent is interacting with the environment. After each episode of interaction with the environment the agent enters a training phase. During training, these replay tuples are sampled using uniform random sampling to be used as training data to update the DQN.

If the DQN was trained on consecutive data from the environment then training would be inefficient as the batches of data would be strongly correlated. By

randomly sampling we can remove the correlation between samples and reduce the variance of updates. Additionally, by using data from many different training episodes we can prevent training from being dominated by samples generated from newly learnt behaviours. For example, if the maximising action is to go left then training samples would become dominated by trajectories supporting this action. This can create feedback loops where an agent is unable to escape local minima. In practice, we implement ER by storing the last n episodes and uniformly sampling d data points per update step. The basic version of ER is limited as it gives uniform importance to all data points and more complex sampling schemes can be used such as Prioritised Experience Replay (PER) (Schaul et al., 2015) which weights the sampling scheme to prioritize selecting samples that the agent has had high TD error during its predictions.

Fixed-Q Targets

During training, both the target and evaluation data are generated by the same NN. This means that the parameters being used to determine the optimally of the policy are the same parameters that generated the data that is being evaluated. This creates instability in training as the target values used are non-stationary which created high variance. Essentially when compared to tabular methods it could be like trying to learn Q-values if all values on the Q table changed each time a small update was done. Instead, we can use a target network that is a clone of the main policy network.

The Q target y_i is calculated by an ANN which is parameterised by weights w' which are a set of frozen weights copied from the weights of the DQN every i training iterations. The DQN using the frozen weights is called the target network and is illustrated in figure 2.2. Taking into account this fixed-Q target the update functions becomes:

$$\nabla\theta = \alpha[(R(s) + \gamma max_u \widehat{Q}(s', u, \theta^-)) - Q(s, u, \theta)]\nabla_\theta Q(s, u, \theta) \tag{2.9}$$

Where ∇w is the change in weights of the neural network, (R + γ max$_a$ $\widehat{Q}(s', u, \theta^-)$) is the maximum possible Q-value for the next state, $\widehat{Q}(s, u, \theta)$ is the current predicted Q-value and $\nabla_\theta \widehat{Q}(s, u, \theta)$ is the gradient of the current predicted Q-value. Per specified number of steps the target network is updated. This prevents the Q-targets from fluctuating as the DQN trains and improves stability. There are multiple other enhancements that have been made to the basic DQN architecture to improve exploration and better estimate Q-values (Hessel et al., 2017) however, these improvements bring additional sets of hyperparameters to tune and are not universally used as they do not improve performance on all environments (Hessel et al., 2017).

Deep Recurrent Q-Networks (DRQN)

Standard DQNs assume full observability. Essentially it is assumed that the agent receives a state s_t as input where there is no hidden information (Watkins, 1989). In partially observable environments we consider the full ground truth state s_t to be hidden and the agent instead receives only observation z_t according to the observation function $O(s, u)$ where the observation is correlated with the global state but is not a complete representation of it (Kaelbling et al., 1998). To accommodate this instead of calculating $Q(s, u)$ where the Q function is based on the individual state s_t we instead approximate $Q(\tau, u)$ where τ is the history of local observations learnt

using a recurrent neural network which is able to maintain an internal state to represent an aggregation of observations over time (Hausknecht and Stone, 2015). The hidden states of the RNN are represented as an additional input to the DQN, h_{t-1} which is used t predict the hidden state h_t for the observations at each timestep.

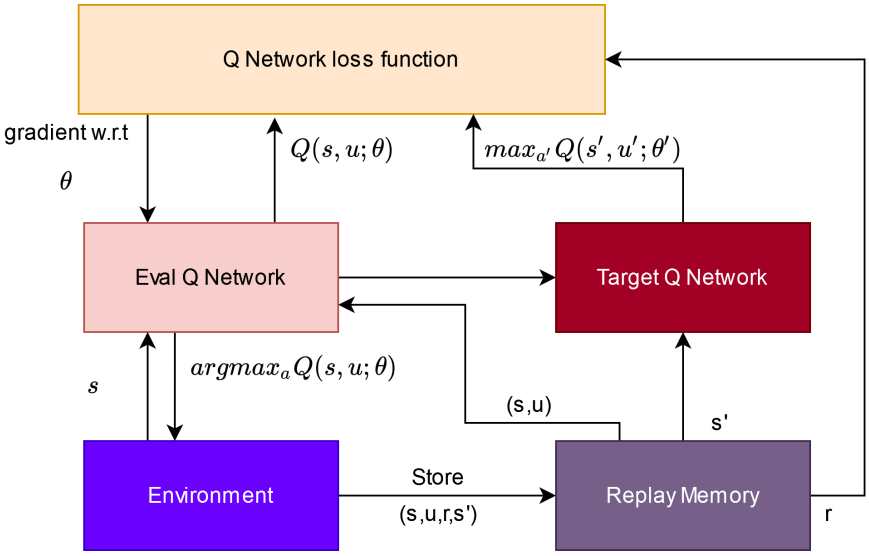


FIGURE 2.2: Basic DQN Diagram

Policy Gradients (PG)

With Q-learning we are required to learn a Q-value function that records the values of each action at each state the agent interacts with. Essentially rather than directly learning a policy we instead learn an optimal policy by learning to select the highest value action at each state. Instead, we can use Policy Gradient (PG) methods which directly learn to optimize for an agent's policy π^a (Williams, 1992). The PG method is to instead model a policy parameterised by θ, $\pi_\theta(u|s)$. The value of the reward function of the environment is then dependent on this policy and by optimising θ we can determine the optimal policy for the given MDP. The objective function for PG algorithms is defined as:

$$J(\theta) = \sum_{s \in S} d^\pi(s) V^\pi(s) = \sum_{s \in S} d^\pi(s) \sum_{u \in U} \pi_\theta(u|s) Q^\pi(s,a) \tag{2.10}$$

Where $d^\pi(s)$ is the stationary distribution of the Markov chain for π_θ (on-policy state distribution under π). PG methods are more useful in continuous spaces as they do not need to calculate the value of each possible action at each state which can become too computationally expensive in large state-action spaces. Gradient ascent is used to move the parameter θ in the direction of gradient $\nabla_\theta J(\theta)$ to find the value for θ to produce the highest possible return.

Computing the gradient $\nabla_\theta J(\theta)$ is difficult as it depends on both the action selection policy π_θ and the stationary distribution of states following the action selection behaviour. In most cases for RL problems the transition properties of the environment are unknown and estimating the effect on the state distribution by a policy update is difficult. We can use the policy gradient theorem (Sutton and Barto, 2018) to reformulate the derivative of the objective function to not include the derivative

of the state distribution $d^{\pi}(s)$ which allows us to simplify the gradient computation of $\nabla_{\theta} J(\theta)$ to:

$$\nabla_{\theta} J(\theta) = \sum_{s \in S} d^{\pi}(s) \sum_{u \in U} Q^{\pi}(s,u) \nabla_{\theta} \pi_{\theta}(u|s) \tag{2.11}$$

The vanilla policy gradient update has no bias but high variance as the magnitude of each update or the extent to which the parameter θ changes during each update is based on the expected reward of only one state-action pair:

$$\nabla_{\theta} J(\theta) = E_{\pi}[Q^{\pi}(s,u) \nabla_{\theta} ln \pi_{\theta}(u|s)] \tag{2.12}$$

PG algorithms can take many forms aside from the vanilla PG algorithm and the gradient calculation can be written in the general form $g = E[\sum_{t=0}^{\infty}] \Psi_t \nabla_{\theta} log \pi_{\theta}(u_t|s_t)$ where Ψ_t is the function representing the calculation of the size of the gradient updates and g is the gradient of the PG algorithm (Schulman et al., 2018).

The simplest formation of Ψ_t takes the form of $\sum_{t=0}^{\infty} r_t$. We do this by performing gradient ascent on an estimate of the total discounted reward against chosen actions. The simplest form of this is the REINFORCE (Monte-Carlo policy gradient) (Williams, 1992) algorithm which has the update function below :

$$\nabla_{\theta} J(\theta) = E_{\pi}[G_t \nabla_{\theta} ln \pi_{\theta}(a|s)] \tag{2.13}$$

where R_t is the total discounted rewards over an episode and π_{θ} is the policy of the agent. REINFORCE uses the full trajectory over the episode to calculate the gradient and requires completed episodes to learn a policy. A commonly used variant of REINFORCE subtracts a baseline value from R_t to reduce the variance of the gradient estimation without affecting bias.

Another method to reduce the variance of the gradient is the use of a learnt baseline. This method is called an *Actor-Critic* (AC) (Sutton et al., 1999) which is illustrated in full in figure 2.3. An Actor-Critic has two models which can share parameters but can also be separate: A **critic** illustrated in green in figure 2.3 which learns a state-value function $V(s)$ or action-value function $Q(a|s)$ and an **actor** illustrated in red in figure 2.3 which updates the policy parameters θ for the policy $\pi_{\theta}(a|s)$ in the direction of the gradient suggested by the **critic**.

The most common on-policy actor-critic method is the advantage actor-critic (A2C) which makes use of a learnt advantage function $A(s,u) = Q^{\pi}(s_t, u_t) - V^{\pi}(s_t)$. Commonly in place of $Q^{\pi}(s_t, u_t)$ we instead use $r_t + V^{\pi}(s_{t+1}) - V^{\pi}(s_t)$ where $V^{\pi}(s_{t+1})$ is the expected value of the next state and is calculated using a target network which has its weights frozen and is updated to match the evaluation network after a fixed number of training iterations. As this form of value function approximation allows us to bootstrap future values AC methods are able to train using partial back-ups where an episode has not progressed until the terminal state and do not need full Monte-Carlo updates like REINFORCE (Schulman et al., 2015). This gives the update function:

$$\nabla_{\theta} J(\theta) = \sum_{t=0}^{T-1} \nabla_{\theta} log_{\pi_{\theta}} A(s_t, u_t) \tag{2.14}$$

where $A = G - V$ with G being the total discounted rewards from a rollout or a TD error, V being the expected value of the current state and T is the number of timesteps of the rollout.

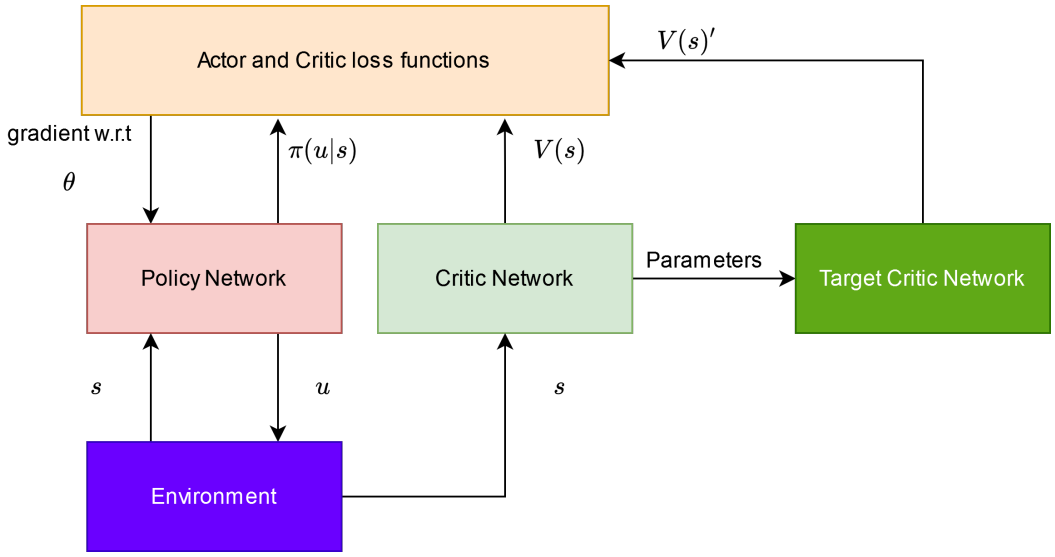


FIGURE 2.3: Basic Actor-Critic Diagram

2.3 Deep learning for MARL

In a MA environment, the large parameter space and representational captivity of ANNs is very useful. Primarily in purely cooperative settings agents can used a single shared parameterised policy to represent all cooperative agents. The single policy is trained to represent the policy of all individual agents provided that have the same observation and action space.This greatly improves training times when compared with multiple separate models (Singh et al., 2018). ANNs also learn information through back-propagation which allows easy computation of gradients. This has been even shown to be able to learn complex communication protocols (Singh et al., 2018).

Independent Actor-Critic and Q-Learning

Independent Q-learning (IQL) is an extension of Q-learning to a multi-agent setting (Tampuu et al., 2015). In these setting each agent a learns an independent Q-function, Q^a. Q^a is only conditioned on the action u^a of agent a rather than on the joint action of the group of coordinating agents $\mathbf{u}$. This can be addressed practically by treating each agent as a separate DQN. Each agent independently and simultaneously learns its own DQN $Q^a(s, u^a; \theta_i^a)$. This can lead to convergence issues as the learning of each agent makes the environment appear non-stationary from the egocentric perspectives of each other agent. An additional issue is the non-stationary environment violates one of the assumptions used for the Experience Replay (ER) buffer used in standard deep Q-learning. Due to the changing environment dynamics as the agents learn the data in the ER buffer does not accurately reflect the properties of the agents over time and its utility for training rapidly deteriorates (Foerster et al., 2017).

In independent actor-critic (IAC) each agent estimates the advantage using their own local observation history which is also used as the state space input for the policy. It is commonplace for the policy and critic networks to share parameters which has been found to improve training speed. During training, agents learn to treat the other agents in the environment and their policies as static parts of the environment

effectively learning them as noise. It has been noted that when IAC is applied in MA domains it exhibits high variance during training and performs poorly (Lowe et al., 2017). This occurs as all the rewards of all the agents in the environment depend on every other agent they must interact with, and as the number of agents increases, the likelihood of taking the correct gradient direction for each update decreases exponentially (Lowe et al., 2017).

In both IAC and IQL agents learn to only maximise their own returns and do not model other agents as active actors within the environment to develop policy models. These methods are referred to as *naive learning* (NL) methods and are illustrated in figure 2.4. *Naive learning* methods have poor convergence guarantees and are generally unstable as they can converge to any of the existing Nash Equilibria in the environment and often struggle to converge to stable high-quality policies. This is due to each agent's value function being dependent on both the joint policy and joint action of all agents in the environment and as such, each agent's optimal policy is also dependent on the policy of all other agents. Therefore as agents learn new individual policies the optimal policies for each other agent also change ((Littman, 2001; Littman, 1994)). They still are commonly used as benchmarks for more novel methods as they are still able to perform reasonably in many environments (Foerster, 2018).

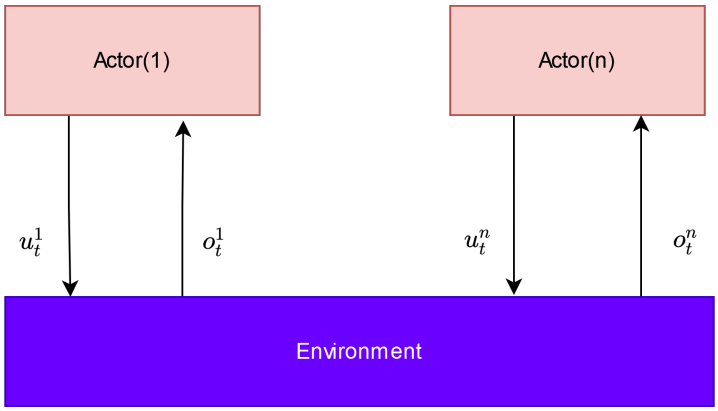


FIGURE 2.4: Relationship between independent agents

Value Function Decomposition

Another approach to purely independent training of MARL agents is the use of value factorisation. In the purely cooperative domain, we use a single global reward to represent the success of the entire group of cooperative agents. This makes it difficult for agents to determine their individual contributions to the success of the joint policy (Rashid et al., 2018). We can use value factorisation methods to decompose the global reward into local rewards associated with each individual agent which allows them to assess their contributions more effectively. We illustrate the relationship for this paradigm in figure 2.5. The structure is similar to the independent agent layout, however, a decomposition layer is given the Q value of each agent at each timestep and then this decomposition layer is used to determine the relationships between the local and global rewards. In the context of the piano movers' problem in the introduction, a global reward would be the reward given to the agents collectively upon reaching the end of the corridor however, it is reasonable to assume both agents would not have used the exact same trajectories to reach the end. We would decompose the global reward at the end of the task into a local reward so that each agent can evaluate its own actions concerning its own contributions at each step.

Value decomposition network (VDN) (Sunehag et al., 2018) introduces one of the first central value decomposition framework for multi-agent cooperative settings where a centralised decomposition function is used to relate the global to local rewards. In this framework, we only receive the global reward and then learn how to distribute the reward across all agents that participated in the task based on their individual contributions.It makes use of an *additive* assumption to relate individual local rewards to a singular global reward. When this is used it assumes that the relationship between local and global rewards and linearly additive :

$$Q_{tot} \approx \sum_{i=1}^{n} Q_i(o^i, u^i) \tag{2.15}$$

where Q_{tot} is the global reward at the current step and Q_i is the individual rewards for each of n agents. Essentially the sum of the rewards of each individual agent is approximately the same as the global reward at any timestep. In most cases, the relationship between global and local rewards is not exactly linearly additive but the *additive assumption* is still accurate enough to ensure effective training. Additionally this assumes that each agent is always acting greedily with respect to its local Q-value and is therefore not applicable to algorithms that make use of stochastic policies.

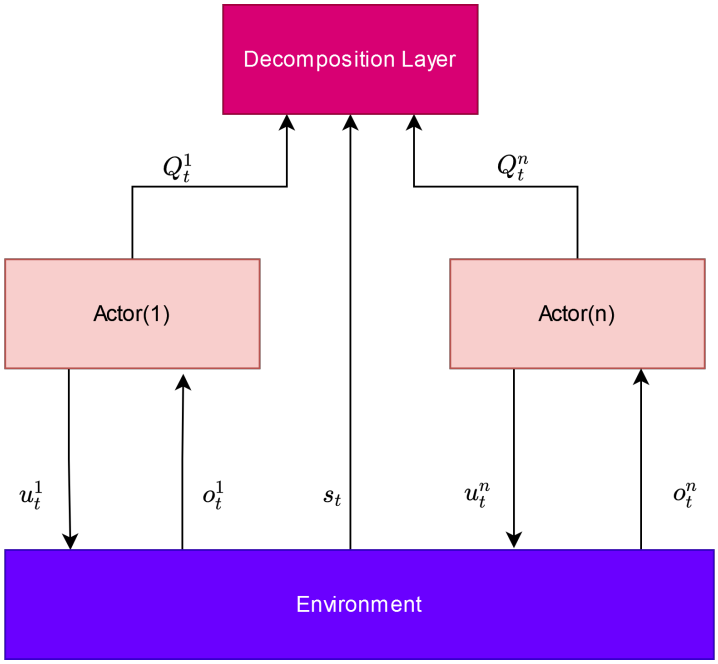


FIGURE 2.5: Relationship between value factorisation agents

This method is improved upon with Qmix (Rashid et al., 2018) which relaxes the additive assumption by forcing the evaluation of a monotonic function through the use of hypernetworks which are functionally neural networks that are used to learn the weights of another neural network instead of directly training them (Ha et al., 2016). This monotonic constraint allows a more relaxed formation of the relationship between Q_{tot} and each individual Q_a. To ensure monotonicity we use the constraint below:

$$\frac{\partial Q_{tot}}{\partial Q_a} \geq 0, \forall a \in A \tag{2.16}$$

Where ∂Q_{tot} the gradient of the global reward and ∂Q_a the gradient of each individual agent's rewards are both monotonic (always increasing or always decreasing). The weights of the decomposition network are produced using hyper-networks which produce a vector which is then reshaped to the appropriate size. Hyper-networks are ANNs that are used to learn the weights of another neural network. Each hyper-network uses absolute activation functions so that the weights of the decomposition network are always non-negative. The full diagram for Qmix is illustrated in figure 2.6.

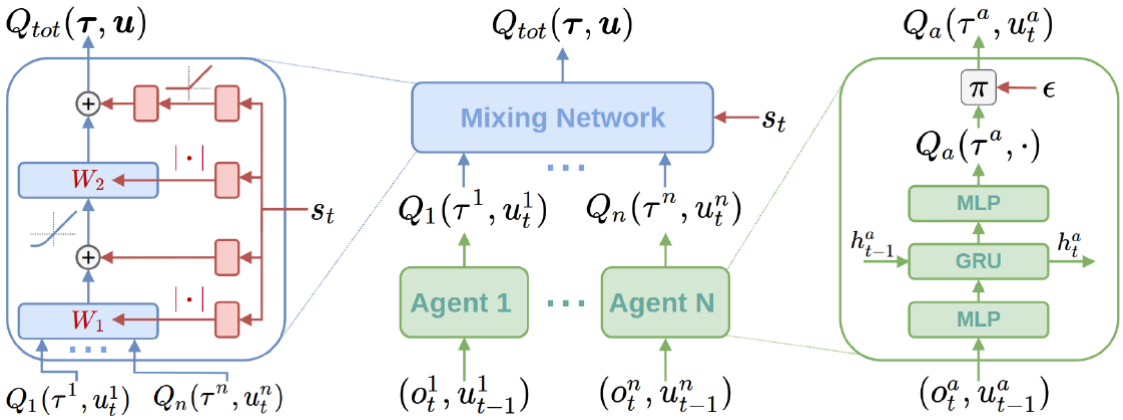


FIGURE 2.6: Full diagram of Qmix with mixing network diagram in the left and agent networks on the right. Hypernetworkers are in red and produce the weights and biases for the mixing network layers shown in blue. (Rashid et al., 2018)

This architecture displays greater performance than the purely additive method used in VDN for most cases, especially in cases with heterogeneous agents. With Qmix we can describe the relationship between Q_{tot} and all individual Q_a by saying that the joint-action set with the maximum value is the same as the group of all local actions with the maximum value:

$$\underset{\mathbf{u}}{\operatorname{argmax}}\, Q_{tot}(\tau, \mathbf{u}) = \begin{pmatrix} \operatorname{argmax}_{u^1} Q_1(\tau^1, u^1) \\ . \\ . \\ . \\ \operatorname{argmax}_{u^n} Q_n(\tau^n, u^n) \end{pmatrix} \tag{2.17}$$

Where τ is the joint action observation history and $\mathbf{u}$ is a joint action. With this formulation we only need that the global argmax on Q_{tot} yields the same results as a set of individual argmax operations performed on the Q-values of each individual agent. This means that rather than requiring learning to fully decompose the direct contribution of each agent to the global reward we only need to learn the local best action to satisfy the optimal joint action space.

Q-value path decomposition

Both VDN and Qmix use an end-to-end training method that relies on back-propagation from the decomposition later towards the networks of the independent agents to perform multi-agent credit assignment. In other fields most notably computer vision (CV) there are already methods used to relate the value of individual inputs towards their relevance to the output of the ANN (Gilpin et al., 2019). They

have been used with success in the single-agent RL case to solve "delayed MDPs" where the reward signals are rewarded very sparsely through the environment by learning to predict the final reward in the episode and redistribute the reward across each timestep to improve learning (Arjona-Medina et al., 2019). Recently this has been applied to MARL in a method called Q-value path decomposition (QPD) which makes use of an explainability method call *Integrated Gradients* (IGs) (Yang et al., 2020a). A more thorough explanation for IGs can be found in Appendix A.

Unlike Qmix which uses the global state s for central value decomposition QPD instead uses $\vec{o}_t$ which is the joint observation vector of all agents as a representation of the global state and $\vec{u}_t = \mathbf{u}$ as a vector representing the joint action of all agents. We apply this to central value functions (CVFs) by defining Q_{tot} as the sum of all path integrals (Yang et al., 2020a):

$$Q_{tot}(\vec{o}_t, \vec{u}_t) = \sum_{i=1}^{n} \sum_{x_j \in X_i} PathIntegratedGrads_j^{\tau_t^T}(\vec{o}_t, \vec{u}_t) \tag{2.18}$$

Where τ_t^T is the trajectory path from time t to T and X_i is the set of agent i's observation features and action dimensions.

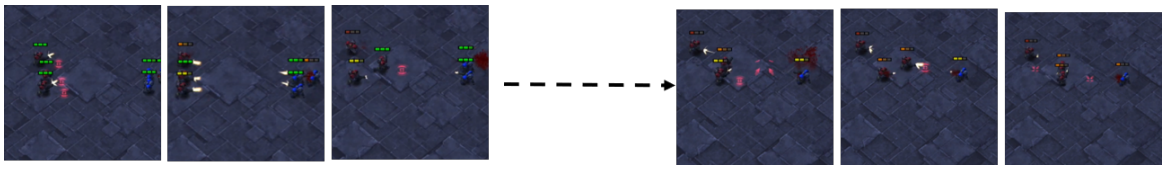


FIGURE 2.7: Example MARL trajectory path

By decomposing Q_{tot} following the real trajectory path, we get each agent's individual contribution to Q_{tot} based on its own local observation and action, which is then used to perform credit assignment.

Counterfactual Multi-Agent Policy Gradients (COMA)

The methods described above assume an agent acts greedily with respect to their own local Q-values and are applied to deterministic Q-learning based algorithms. For policy gradient (PG) algorithms Counterfactual Multi-Agent Policy Gradients (COMA) (Foerster et al., 2018) is the first method in MARL to develop multi-agent credit assignment for the case of a stochastic policy.

COMA (Foerster et al., 2018) illustrated in figure 2.8 uses a centralised critic that is used to learn a function $Q(s, \mathbf{u})$ which estimates Q-values for joint action $\mathbf{u}$ conditioned on central state s which is a representation of the ground truth state of the entire training environment not just the egocentric representation visible by independent the agents. An advantage function is learnt which marginalises out the current action u^a while keeping the actions of the other agents' actions u^{-a} fixed :

$$A^a(s, \mathbf{u}) = Q(s, \mathbf{u}) - \sum_{u'^a} \pi^a(u'^a|\tau^a) Q(s, (u^{-a}, u'^a)) \tag{2.19}$$

Where $A^a(s, u)$ is the counterfactual baseline for agent a and $Q(s, u)$ is the estimate Q-value for the joint action u conditioned on central state s. u^{-a} are the actions of all agents aside from a and u^a is the action of agent a. Essentially we compute the expected reward under state s given action u^a compared to the sum of all expected rewards given all other actions u'^a.

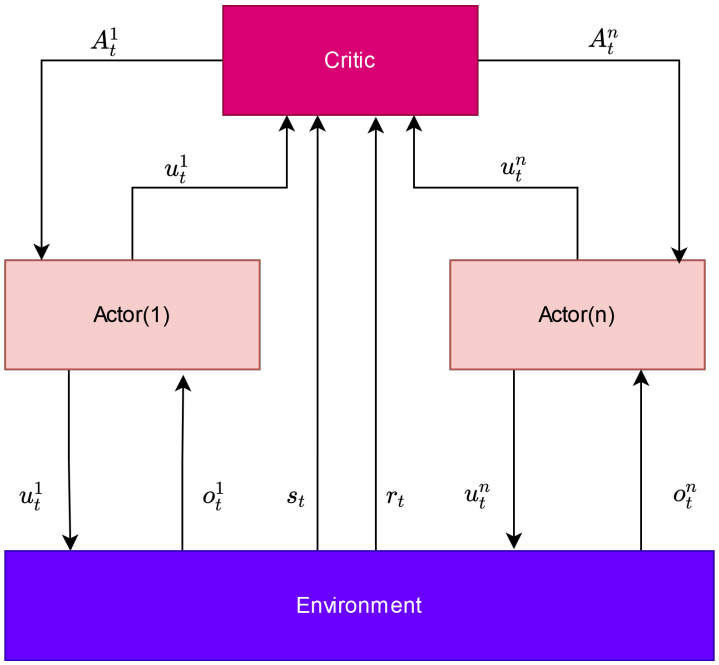


FIGURE 2.8: Relationship between COMA agents

2.4 Conclusion

This chapter provided the formalisms for the rest of the thesis. It describes the basic models to be used in this research and, testing environments which the agents willl be trained in.

Reinforcement Learning (RL) is a form of trial and error learning in which an agent attempts to maximise a numerical reward based on actions performed in an environment. It differs from supervised learning in that we aim to maximise a reward signal rather than train a model using a predefined dataset. RL algorithms are modelled as *Markov Decision Process* (MDP). Various algorithms are then used to penalise or reinforce decisions made by the agent. In cases where an agent only has partial observability of the environment, we instead model the environment as a *Partially Observable Markov Decision Process* (POMDP) and we attempt to learn the underlying distribution of a hidden state in order to solve it.

POMDPs are generalised into the multi-agent setting in the form of decentralised POMDPs (Dec-POMDPs) where multiple agents act under uncertainty based on partial observations. Dec-POMDPs require us to solve a joint policy. We can also no longer estimate the hidden state effectively as individual agent observations are now dependent on the behaviours of other agents. This setting also provides unique challenges depending on the reward signal for cooperative, zero-sum and general sum games. A game theoretical approach to determining optimal policy is the use of Nash Equilibria but as there are usually multiple Nash Equilibria for the same environment this method suffers from multiple local minima. MARL settings are usually too complex in terms of state space to be computationally tractable for game theoretical methods.

The tabular methods used in standard RL are impractical in large state spaces and therefore we make use of function approximators as a replacement. This has found great success in the form of Deep Q-learning and Policy Gradient methods which have expanded RL into continuous and high-dimensional spaces. Recently the use of regret minimization has also been expanded on with the use of neural

networks (NNs) and has shown remarkable robustness to partial observability compared to previous methods.

When deep learning is applied to the MA setting naive methods have been found to have limited success. In the case of cooperative games, it has been found that decomposing the global reward by calculating agent blame is effective. To do this we use a centralised value function to perform decomposition under an additive assumption (Sunehag et al., 2018). This additive assumption was later relaxed using Qmix (Rashid et al., 2018) which greatly improved performance in heterogeneous environments.

Chapter 3

Research Problem

multi-agent systems where agents work towards a common goal can be modelled as cooperative games where a single global reward is assigned to all agents. When using standard RL methods with independent learners we face the issue of credit assignment as determining the contribution of each agent towards the completion of the task is difficult (Busoniu et al., 2008). This is unfortunate as many practical tasks like coordination of railway networks and traffic lights for optimal transportation times can be modelled in this way.

3.1 Significance and Motivation

Central value functions (CVFs) can be used to decompose global rewards to assign contributions to individual agents under the assumption that agents can be heterogeneous and no communication is allowed at runtime to reduce processing overhead.

They have been found to greatly improve over purely independent reinforcement learning methods by exhibiting greater scaling towards large numbers of agents and in the solving of more complex environments as observed in the Starcraft multi-agent Competition (SMAC) (Samvelyan et al., 2019). Both value decomposition network (VDN) and Qmix can be seen to under-perform in the SMAC environment *bane _no _z* with a large number of redundant agents (Yang et al., 2020b) when compared to the other SMAC settings. In the Qatten paper (Yang et al., 2020b) the idea is put forward that this is because in both the linear and monotonic case it is difficult to assign rewards to agents where the weighting between the agents is so heavily biased towards a small number of the total set. They come to this conclusion by analysing the attention weights of their attention-based critic network which showed in order to solve this environment a high reward needed to be assigned to a small subset of the agents which are the most relevant to the success of the joint policy.

Additionally, all algorithms mentioned above require access to a global state in order to determine the value of the local observations of each agent. In complex environments a global state is not necessarily informative as the relevance of local observations cannot be easily related to the global view and in high-dimensional spaces, it becomes difficult to define (Yang et al., 2020a). An example of a complex setting with these issues is the *RoboCup* environment (Dizet et al., 2021) which relies on using simulated digital sensory data which emulates data recorded by a low-resolution video recorder. This recorded data is the local observations of the independent agents and represents their local partial observation of the environment. A concatenation of these local observations is still only a partial representation of the full state space and in the centralised training decentralised execution (CTDE) paradigm results in sub-optimal performance for most algorithms. Given the limitation of the observable space with real or realistic settings, it is not reliable to assume

that a reasonable state approximation can be made from only these limited local observations.

In this redundant case Policy Gradient (PG) algorithms are naturally able to more effectively solve these settings due to making use of stochastic policies which allow the agents more effective exploration ((Yu et al., 2021; Papoudakis et al., 2020)). Despite this utility, however, MARL research making use of PG-based algorithms has been significantly less than their Q-learning based counterparts. It was believed that the lower sample efficiency of PG algorithms is the reason for observing drastically lower results across most benchmarks compared to Q-learning methods ((Foerster et al., 2018; Samvelyan et al., 2019)). Even newer PG methods like LICA (Zhou et al., 2020) are not directly comparable to Q-learning methods as they require over 10x as many training samples to achieve comparable performance.

As PG methods are only able to use on-policy data which has been gathered from the current joint policy of the agents they inherently have lower sample efficiency than off-policy algorithms like Q-learning. In more difficult environments collecting large numbers of samples is computationally expensive and requires a large amount of parallel workers collecting data. Achieving this amount of data collection effectively requires the use of large-scale distributed training methods like IMPALA (Espeholt et al., 2018a) which require large amounts of compute to effectively leverage. However, the ability of Q-learning methods to make use of off-policy data is not as useful as it is in the single-agent setting. It has been noted that the massive replay buffers used in the single-agent case are not transferable to MARL (Hu et al., 2021) due to the rapid rate of relevancy decay of old data samples during training (Foerster et al., 2017). In some settings, it has been advantageous to remove the replay buffer entirely (Foerster et al., 2016) meaning that the sample efficiency advantage of Q-learning methods is not effective in the multi-agent setting.

3.2 Research Aims and Objectives

There is a need to create models that can more accurately determine the value of local state-action pairs in multi-agent systems without the need for a global state representation that can accommodate for redundancies in real-world settings. This research aims to improve CVFs to better deal with large numbers of redundant agents by using layerwise relevance propagation (LRP) (Montavon et al., 2019) as an alternate method to perform value function decomposition in order to remove the requirement of a central global state. There are also environments where Q-value based factorisation methods prove inadequate like highly sparse reward environments (Papoudakis et al., 2020) while policy gradient (PG) algorithms are able to easily solve them. We aim to produce methods to increase the sample efficiency of naive policy gradient algorithms that can be applied to centralised advantage actor-critic (A2C) algorithms with minimal additional implementation complexity over the naive method.

The above aims of this research project will be achieved through the following objectives:

- Create a set of environments of varying levels of redundancy in the SMAC simulator.
- Incorporate LRP into the central critic used to perform value function decomposition to train MARL algorithms for better credit assignment in environments with high numbers of redundant agents.

- Create a sample efficient centralised A2C algorithm with reduced variance compared with baseline A2C.
- Analyse the performance of our proposed algorithm(s) against the state of the art (SOTA) to determine the success of implementation.

3.3 Research Questions

Can layer-wise relevance propagation be used to decompose value functions in highly redundant environments for better credit assignment while using only local agent information?

Can the naive centralised actor-critic's performance be brought to near the levels of the popular Qmix algorithm using only simple off-policy data reuse?

3.4 Outline

The following chapter will provide more insight into the research methodology for this thesis along with the plan of how to produce the desired outcomes.

The research methodology will deal with the theoretical component in the form of research design and pseudo-code explaining the implementation of our methods.

Chapter 4

Research Methodology

4.1 Introduction

This research has the goal of improving methods to use central action-value functions (CVFs) to train agents to solve multi-agent tasks. We present two methods from our research: relevance decomposition network (RDN) and cycling replay buffer (CRB).

RDN is a method that makes use of layerwise relevance propagation (LRP) to decompose the global value function learnt by a critic to it's local value functions. These decomposed value functions are calculated by the addition of all relevance scores associated with the information per agent. These decomposed rewards are used to train the local independent agents.

CRB uses the addition of a naive cycling replay buffer to improve the sample efficiency of PG algorithms without the use of off policy update correction like V-trace (Espeholt et al., 2018b). The buffer removes the oldest entries of data as new data is added and then the agents are trained on the entire stored dataset. We also introduce a KL divergence mask to accommodate data that is too far off-policy which would induce destabilisation.

4.1.1 Research Design

Relevance Decomposition Network (RDN)

We propose RDN to perform credit assignment between ad-hoc agents in the cooperative multi-agent setting under the assumption of a linear relationship between the individual local rewards and the shared global reward. However unlike most central training decentralised execution (CTDE) methods which use learned decomposition as an end-to-end system RDN separates learning of the global reward function from the local Q values of the agents. We provide an overall diagram of the framework below in figure 4.1 of RDN. It consists of a central critic which takes the inputs $\vec{o}_t$ and $\vec{u}_t$ which are vectors containing the observations of all local agents and their actions taken at each timestep respectively. Agents are represented practically using Deep Recurrent Q Networks (DRQN) (Hausknecht and Stone, 2015) which take a local observation from the environment o which is the egocentric partial observation of the independent agent. The critic is used to train the agents by using LRP (Montavon et al., 2019) to calculate the relevance of the local observations and actions to perform credit assignment.

Layer-Wise Relevance Propagation (LRP) is a method that has been developed to interpret neural network based machine learning models (Montavon et al., 2019). It assigns a relevance or importance value to each neuron of the neural network (NN) which described how much it contributed o the final network output. This is done

by propagating the prediction of the NN $f(x)$ backwards through the NN using a set of propagation rules. These rules are described in more detail in Appendix A.

The propagation of values by LRP is subject to a conservation property where the value assigned to a neuron must be redistributed to the immediately preceding layer in an equal amount. Formally let j and k be two consecutive layers of the neural network. Propagating relevance scores M_k at a given layer onto neurons of the lower layer is achieved by applying the rule:

$$M_j = \sum_k \frac{z_{jk}}{\sum_j z_{jk}} M_k. \tag{4.1}$$

Where z_{jk} models the extent to which neuron j has contributed to making the neuron k relevant with $z_k = \epsilon + \sum_{0,j} avj \cdot \rho(w_{jk})$ where ϵ is a small constant to aid numerical stability, ρ is an arbitrary function applied to the weights of the neurons like an activation function in the NN layer and v is the output from neuron j in the earlier layer of the NN to neuron k. If using the rule above for all neurons in a network we can verify the layer-wise conservation property $M_j = \sum_k \frac{z_{jk}}{\sum_j z_{jk}} M_k$ where M_j is a vector of relevance scores with each neuron $i's$ relevance scores indexed as M_{ji}, and by extension the global conservative property $\sum_i M_i = f(x)$.

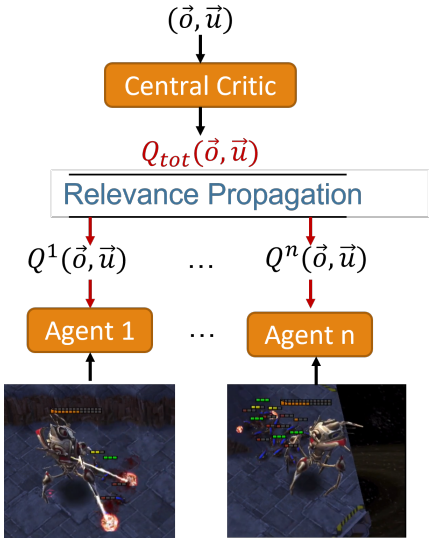


FIGURE 4.1: Diagram of RDN framework

The details of the algorithm are shown in Algorithm 1. Lines 2-9 show how the decentralised agents interact with the environment to gather data where $Q^i(o_{t,i,.})$ is the Q value of independent agent i at timestep t, h_t^i is the hidden state of agent i at timestep t, $o_{t,i,.}$ is the local observation of agent i at timestep t,$u_{t,i}$ is the action taken by agent i at timestep t and $\pi_i(Q^i(o_{t,i,.}), \epsilon(e))$ is the policy π of agent i dictated by a Q value function and an $\epsilon - greedy$ exploration strategy.

Lines 11 to 15 calculate the expected total Q value at each timestep using a critic network parameterised by θ^c and a target Q value using a target critic parameterised by $\tilde{\theta^c}$ whose parameters are copied over from the critic network every 200 updates. The critic network is updated using loss:

$$L(\theta^c) = E_{\vec{o},\vec{u},r,\vec{o}'}[(Q_{tot}^{\theta^c}(o_1, ..., o_n, u_1, ..., u_n) - y)^2] \tag{4.2}$$

Where $y = r + \gamma(Q_{tot}^{\tilde{\theta}^c}(o_1^{'}, ..., o_n^{'}, u_1^{'}, ..., u_n^{'}))$ and θ^c is the critic's parameters and $\tilde{\theta}^c$ is the target critic parameters, which are reset every C training epochs

Lines 16 to 21 update the independent agents' parameters θ^i. The total expected Q value for each timestep is decomposed into independent target Q values $\tilde{\theta}^i$ and the DRQNs (Hausknecht and Stone, 2015) which act as the agent networks are trained using the loss $L(\theta^i) = E_{\vec{o},\vec{u},r,\vec{o}'}[(Q^{i,\theta^i}(o_i, u_i) - \tilde{Q}^i)^2]$ (Sandholm and Crites, 1995) and $\tilde{Q}^i = \sum_i M_{in}$ where $\tilde{Q}^i$ is the Q target for agent i and $\sum_i M_{in}$ is the sum of all relevance values associated with agent i. Comparatively, VDN only trains all agents using the sum of the total Q value as a joint loss in an end-to-end fashion with no critic network to learn the joint value function. Qmix directly takes the Q value of each agent and the ground truth state space as the critic inputs and then relies on back-propagation to implicitly perform value factorisation to train the local agents.

A characteristic of LRP is it maintains a conservative calculation between layers of the neural network (Montavon et al., 2019). As such the relevance values from later layers are included completely in the calculation of the relevance values of the layers closer to the input. Essentially we can assume that the total sum of the relevance values is approximately the same as the output of the NN when LRP is used. therefore we can equate Q_{tot} to the sum of relevance scores as $Q_{tot} \approx \sum_j \sum_i M_{in}$ or alternatively state that Q_{tot} is approximately the same as the sum of all relevance values across all agents.

Algorithm 1: Relevance Decomposition Network (RDN)

1 **Initialize** Critic network θ^c. target critic $\tilde{\theta}^c$, DRQN $\theta^\pi = (\theta^1, ..., \theta^n)$
2 **for** *each training episode e* **do**
3 s_0 = initial state, t = 0, $h_0^i = 0$ for each agent i
4 **while** $s_t \neq$ *terminal* ***and*** $t < T$ **do**
5 $t = t + 1$ **for** *each agent i* **do**
6 $Q^i(o_{t,i,.}), h_t^i = DRQN(o_{t,i}, h_{t-1}^i; \theta^i)$
7 Sample $u_{t,i}$ from $\pi_i(Q^i(o_{t,i,.}), \epsilon(e))$
8 Execute the joint actions $(u_{t,1}, u_{t,2}, ..., u_{t,n})$
9 Receive the global reward r_t and next state s_{t+1}
10 Add new episode to replay buffer and sample a batch of episodes
11 **for** *b in batch* **do**
12 **for** $t = 1$ *to* T **do**
13 Calculate targets y_t using $\tilde{\theta}^c$
14 Calculate expected values Q_t^{tot} using θ^c
15 Update critic parameters θ^c with loss $L(\theta^c)$
16 **for** *b in batch* **do**
17 **for** $t = 1$ *to* T **do**
18 Calculate independent Q values Q^{i,θ^i}
19 Calculate total Q target from updated critic parameters θ^c
20 Decompose the total Q target into independent Q targets $\tilde{Q}^i$
21 Update DRQN parameters θ^π with loss $L(\theta^\pi)$

Extended-Masked Centralised Actor-Critic (EMCAC)

Although there have been large strides in developing better MARL algorithms in recent years most are focused on off-policy Q-learning based algorithms as policy gradient (PG) methods to produce surprisingly poor results (Samvelyan et al., 2019). The poor performance is strange given comparable benchmarks in the single agent setting (Schulman et al., 2015). Although more complex PG methods have been developed for MARL settings such as LICA they are not directly comparable to the Q-learning based methods as they require up to 10x as many samples to achieve similar performance making them impractical to train until the point where they achieve comparable performance to Q-learning based methods (Hu et al., 2021).

We assess the effect of 2 small changes made to the standard centralised actor-critic (CAC) to improve sample efficiency and learning stability. The first is the use of a small off-policy training data buffer. This replay buffer does not contain a large number of samples to be drawn from during training time like that is sued in Q-learning. Instead, it is of the fixed size which we want to use for the batched training. This allows us to train on a greater number of total samples without requiring additional parallel workers to collect data. We illustrate how this works in figure 4.2 where the buffer of fixed size is used and when the buffer is full and new data is entered the oldest batch of data is removed. With this, we can maintain a larger training set of data without very old data destabilising the training.

The second addition is the use of a KL divergence mask. The KL divergence mask is used to suppress updates from data that deviates too far from the current policy. Data that is too far off-policy causes instability during training if used naively (Wang et al., 2016). During training the, data in the buffer is re-evaluated using the average policy found during training. If the policy of the data as evaluated by the average policy deviates from the current policy by a predetermined bound then the advantage function for the update of that data point is set to 0 and it is suppressed during training. Finally, we make use of a mixed on and off-policy training regime similar to the one used to train the phasic policy gradient (PPG) (Cobbe et al., 2021) where the critic network is given additional training during the training loop using off-policy data. Critic networks have been shown empirically to be robust to using off-policy data and with additional training can accelerate the rate at which the agent networks are able to reach stable policies (Cobbe et al., 2021).

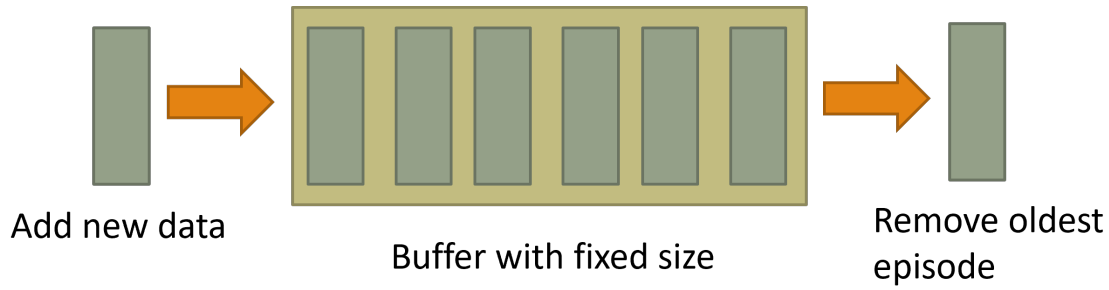


FIGURE 4.2: Diagram of Cycling Replay Buffer

The details of the algorithm are shown in Algorithm 2 below. Lines 2-9 show how the decentralised agents interact with the environment to gather data where $\pi^i(o_{t,i,.})$ is the probability distribution for all actions of independent agent i at timestep t, h^i_t is the hidden state of agent i at timestep t, $o_{t,i,.}$ is the local observation of agent i at timestep t,$u_{t,i}$ is the action taken by agent i at timestep t and $\pi_i(o_{t,i,.}), \epsilon(e))$ is the

policy π of agent i produced by the softmax function and an ϵ is the parametrisation term for the bounded softmax that can be used with policy gradients (Foerster et al., 2018) .

Lines 14 to 18 calculate the expected total value at each timestep using a critic network parameterised by θ^c and a target Q value using a target critic parameterised by $\tilde{\theta^c}$ whose parameters are copied over from the critic network every 200 updates.The critic network is updated using loss

$$L(\theta^c) = E_{\vec{o},\vec{u},r,\vec{o}'}[(Q_{tot}^{\theta^c}(o_1, ..., o_n, u_1, ..., u_n) - y)^2] \tag{4.3}$$

Where $y = r + \gamma(Q_{tot}^{\theta'^c}(o_1^{'}, ..., o_n^{'}, u_1^{'}, ..., u_n^{'}))$ and θ^c is the critic's parameters and $\theta^{'c}$ is the target critic parameters, which are reset every C training epochs. The policy network is updated using the loss $\nabla_\theta J(\theta) = \sum_{t=0}^{T-1} \nabla_\theta log_{\pi_\theta} A_t$ where $A_t = G_t - V_t$ with G_t being the total discounted rewards at the current timestep and V_t being the expected value of the current timestep. We also mask out the advantage function where the KL divergence exceeds some predefined threshold K which makes the advantage function a bounded function where $A_t = G_t - V_t$ where $KL(\theta^\pi || \tilde{\theta^\pi}) < K$ where, K is a hyperparameter defining a maximum KL divergence between the current and average policy from training represented by a target policy network. Functionally the PG updates is made into a bounded function w.r.t the KL divergence between the current and average target policy which prevents large updates from using the off-policy data from destabilising learning.

Lines 21 to 25 are the off-policy update step of the algorithm which is functionally the same as the on-policy stage but only the critic network is trained and the critic parameters are updated for a second time with loss $L(\theta^c)$.

Algorithm 2: Extended-Masked Central Actor-Critic (EM-CAC)

Initialize offline replay buffer D, semi-online replay buffer D' Critic network, θ^c. target critic $\tilde{\theta^c}$, policy $\theta^\pi = (\theta^1, ..., \theta^n)$,target policy $\theta^{\tilde{\pi}} = (\theta^1, ..., \theta^n)$
for *each training episode e* **do**
  s_0 = initial state, t =0, $h_0^i = 0$ for each agent i
  while $s_t \neq$ *terminal* ***and*** $t < T$ **do**
    $t = t + 1$ **for** *each agent i* **do**
      $\pi^i(o_{t,i,.}), h_t^i = policy(o_{t,i}, h_{t-1}^i; \theta^i)$
      Sample $u_{t,i}$ from $\pi_i(o_{t,i,.}), \epsilon(e))$
    execute the joint actions$(u_{t,1}, u_{t,2}, ..., u_{t,n})$
    receive the global reward r_t and next state s_{t+1}
  Add new episode to offline replay buffer and sample a batch of episodes
  Add new episode to semi-online replay buffer
  Remove oldest episode in replay buffer
  # On-policy stage
  for *e in batch* **do**
    for $t = 1$ *to* T **do**
      Calculate critic targets y_t using $\tilde{\theta^c}$
      calculate expected critic values Q_t^{tot} using θ^c
      calculate $td(\lambda)$ returns
      calculate KL divergence between θ^π and and $\tilde{\theta^\pi}$
  Update critic parameters θ^c with loss $L(\theta^c)$
  Update policy network θ^π with loss $L(\theta^\pi)$
  # Off-policy stage **for** *e in batch* **do**
    for $t = 1$ *to* T **do**
      Calculate critic targets y_t using $\tilde{\theta^c}$
      calculate expected critic values Q_t^{tot} using θ^c
      calculate $td(\lambda)$ returns
  Update critic parameters θ^c with loss $L(\theta^c)$

4.2 Conclusion

An objective of this research is to allow CVF methods to be more effective in environments with high numbers of redundant agents where state information is limited to more accurately reflect real-world conditions. Also, we aim to provide simple to implement augmentations to the basic CAC algorithm to improve both the sample efficiency and stability of policy gradient algorithms to be more comparable to their Q-learning counterparts

We have accomplished our first objective with our algorithm RDN which uses LRP as a way to more effectively assign credit in environments with high numbers of redundant agents. The second objective is completed with the use of EM-CAC which extends the online replay buffer to make better use of training data and a KL divergence mask to stabilise training.

Initial testing is done using a simple 2-step grid world for a simple proof of concept in an environment that is trivial to solve for a human and in *SMAC* to show the effectiveness of our methods in more complex domains.

Chapter 5

Experiments

5.1 Introduction

The previous chapter introduced the research hypothesis and the method by which we intend to test it. In this section, we will consider the practical execution of this research and the results obtained from it.

For RDN we are primarily concerned with how increasing numbers of redundant agents make it difficult to converge on good policies for the two primary value factorisation methods in VDN and Qmix. We first test these algorithms along with independent Q-learning (IQL) on a simple 2-step matrix game to show that even in simple cases Qmix and VDN are unable to accurately converge to optimal policies and misassign credit to agents. We then created a set of maps based on the *bane_vs_bane* where VDN and Qmix were found to exhibit high variance in testing. We create 4 different versions of the scenario where we decrease the number of redundant agents on the RL-based agent.

For extended-masked centralised actor-critic (EM-CAC) we only use a subset of the total SMAC site to test out enhancements using two maps centralised actor-critic (CAC) is able to solve to determine that EM-CAC does not cause instability in cases where CAC is already effective and two more difficult maps where CAC typically fails to achieve high performance within 5 million time-steps. We perform an ablation study of EM-CAC showing the value of each enhancement individually and all together and record the KL diverge of the EM-CAC method to show how KL divergence is kept to a minimum across the replay buffer. Finally, we record results with different numbers of off-policy data to show how much can be introduced into the learning cycle.

5.2 Testing Environments

2 step matrix game

The most simple form of cooperative sum game is a simple 2 step matrix game. For this game, we can easily guarantee game theoretical convergence to determine the optimal policy for each state-action pair. Thus we use this to determine the effectiveness of each algorithm in the most optimal use case (Yu et al., 2019).

In step one agent one chooses a row and agent two a column simultaneously. These choices determine the matrix game to be played. In step two, agents again choose a row and column within the chosen matrix game. We illustrate this game in figure 5.1 below where the agents have chosen to play game three in the bottom right which is the second column and second row. Then in step two, the agents choose column and row 1 in the top left of game four. After step two they have been given

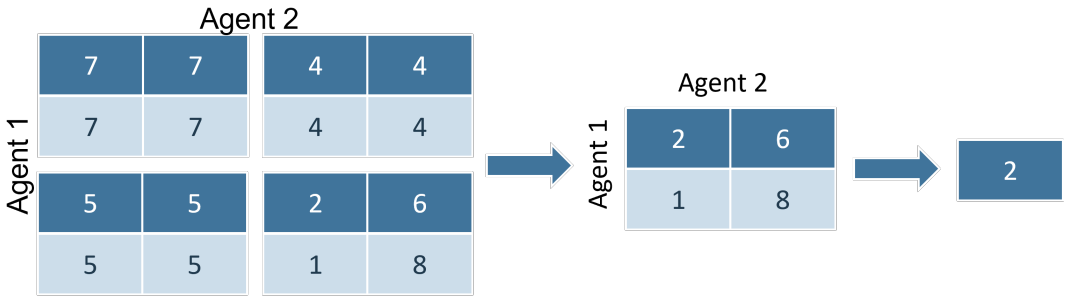


FIGURE 5.1: 2 player matrix game: player 1 chooses a row and player 2 chooses a column per step for 2 time-steps. The location of the players in the final timestep determines the global reward assigned to the episode

a reward of 2 as the assigned value of their position in that game. This payoff is a global reward and is assigned to both agents.

Independent Q-learning methods struggle to solve these simple tasks as agents do not have communication methods to determine the actions of each other (**QMIX**). As a result, they should favour choosing game 1 which has the highest average payoff irrespective of the action of the other agent rather than game 3 which presents a higher potential reward but relies on coordinated decision making.

Starcraft Multi Agent Competition (SMAC)

MARL studies lacked a cohesive testing environment akin to the Atari Learning Environment (ALE) (Bellemare et al., 2013) suite used in single-agent DRL. The ALE suite provides an interface to a large number of Atari 2600 games and has become a common benchmarking tool for DRL.

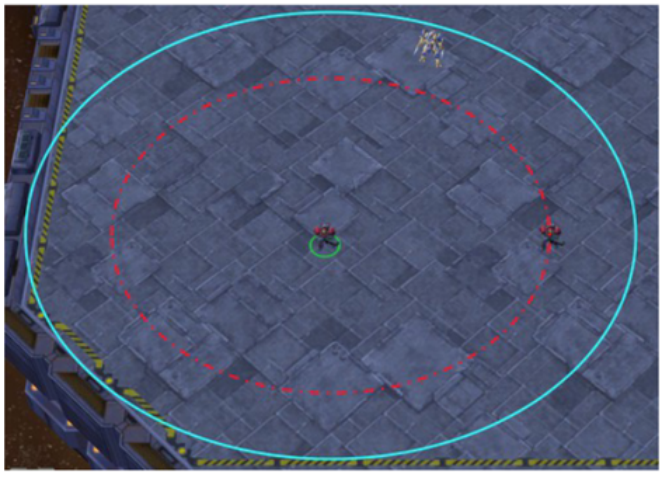


FIGURE 5.2: Boundaries showing the vision range or visible state information of the agent (blue) and attack range (red) of a marine unit in the SMAC training environment

In 2019 SMAC (Samvelyan et al., 2019) was released. It is a platform for the

testing and training of agents under the central training and decentralised execution paradigm. It focuses on smaller-scale micromanagement tasks rather than attempting to solve the entire game of Starcraft2 (SC2). For example, in the most basic scenario *3m* each team controls 3 marine units and wins by defeating all the enemy units. In this scenario and all others, each member of the team is treated as an individual agent. The main task to learn in this scenario is to have all units on a team attack the same enemy on another team at the same time to reduce the defeat the enemy agents more quickly. Agents are given a partially observable local state illustrated in figure 5.2 and must act off of that during testing. Information is formatted as a vector for each agent containing the IDs, health, shields and distances or all agents in observable range. Agents outside of the visible space are replaced with a vector of zeros. Due to this, we do not need to learn to perform image recognition to use this environment which simplifies the learning process .Agents are only able to observe a local field of view within a circle around their position. They can only observe other agents if they in the line of sight or alive so they have to learn to distinguish if other agents have died or left the line of sight.

The actions available to all agents are a move in an assigned direction, attack an enemy unit with a specific enemy ID, stop all actions and perform no action for that timestep. In the premade scenarios the total action space ranged between 7 and 70 depending on the number of enemy units and the built-in macro actions like attack and move at the same time are disabled. When agents act at each timestep they are only able to choose a single action to execute at that moment. Actions are recorded as a one-hot vector encoding of the chosen action. However the, training environment does not accommodate illegal actions like units trying to attack enemies in vision range but outside of attack range. Users must manually mask out illegal actions during training and execution time from a vector of legal moves that is generated at each timestep from the environment.

Replays can be viewed using the standard SC2 replay system and new maps made using the Blizzard map maker tool [1]. Each new map made must be manually, added to the training environment and the number of units and total timesteps for the scenario set. Each scenario runs for a maximum number of timesteps after which it will terminate without all player or enemy units being destroyed. This means it is possible for agents to receive a loss for a scenario without all friendly units being destroyed if they do not complete the task in time. Due to agents trained in SMAC typically being ad-hoc agents which do not communicate with one another the player agents can be overall successful in destroying the majority of the enemy units but if the enemy unit moves out of range of the vision of the majority of units then the team of agents can fail to relocate it causing them to lose by the timer running out.

Large-scale performance summaries have been performed on a large array of maps which provides a standardized benchmarking tool which is not available in other environments ((Samvelyan et al., 2019; Papoudakis et al., 2020)).

[1] https://www.blizzard.com/en-us/apps/battle.net/desktop

5.3 Methodology

5.3.1 Baselines

We consider several algorithms for use as baselines in our experiments. Independent Q-Learning (IQL) represents a naive approach to solving multi-agent tasks but is still able to show reasonable performance in certain environments (Foerster, 2018). QMIX is currently the most popular algorithm to our knowledge and many newer MARL algorithms like ROMA (Wang et al., 2020) and MAVEN (Mahajan et al., 2019) are auxiliary augmentations to improve learning for specialised settings like sparse reward settings.

5.3.2 Training environments

SMAC uses Starcraft 2 in order to create a series of micromanagement tasks using decentralised controllers (Samvelyan et al., 2019). Enemies are controlled by pre-designed heuristics. Proper micromanagement is required to maximise the damage dealt to enemy units and achieve the optimal reward. Units are required to learn a wide array of skills including focusing fire, kiting and efficient pathing. It has become a common benchmark for many SOTA MARL algorithms in the fully cooperative setting.

5.3.3 Network architecture and hyperparameter configurations

RDN

For the SMAC environments and the 2-step matrix game, we use the same hyperparameters used in the original SMAC benchmarking paper (Samvelyan et al., 2019). The architecture of the agents is a standard deep recurrent Q network (DRQN) (Hausknecht and Stone, 2015) with a gated-recurrent unit (GRU) layer with a 64 cell hidden state followed by 2 fully connected layers with 32 nodes each and a final fully connected layer with nodes equal to the size of the action space of an individual agent.

The critic network is a feed forward neural network with 2 dense layers of 64 nodes each and an output layer of size 1.

We use a γ value of 0.99 in all environments. An $\epsilon - greedy$ exploration strategy is used with an initial ϵ value of 1 which is annealed to 0.05 over 50k timesteps for SMAC and 2000 steps for the matrix game.

To increase learning speed we share parameters across all individual Q networks and use a single network to represent all agents. A one-hot encoding of the agent type is concatenated into each agent's observations. By using the agent type as part of the observation space we can train a single shared neural network to represent all agents which have been shown to improve learning speed significantly with minor performance loss (Papoudakis et al., 2020). Both the critic and the agent networks are trained using Adam with a learning rate of $5x10^{-4}$. The replay buffer for SMAC contains the most recent 1000 trajectories and is sampled with a batch size of 32. For the 2-step matrix game we only keep 200 trajectories and use a batch size of 4. Target networks for the critic are updated every 200 training epochs.

We test our method every 100 training episodes for SMAC and 10 for the matrix game for 20 test episodes with exploration disabled. We evaluate performance based on the percentage of wins per set of test episodes. We conduct 8 separate training runs in order to gather the data from the experiments.

Agent networks are given an input consisting of only the most recent observation, previous action and their identity as a one-hot encoding. The critic for RDN is given a concatenation of all agents' previous 4 observations, current actions and their one hot encoded identities. For QMIX the critic takes in the ground truth state and the expected Q values of the independent agents as in the original QMIX paper (Rashid et al., 2018).

5.3.4 EM-CAC

We use the same hyper-parameters used in the original SMAC benchmarking paper(Samvelyan et al., 2019). The architecture of the agent networks is standard deep recurrent Q network (DRQN) (Hausknecht and Stone, 2015) with gated-recurrent unit (GRU) layer with a 64 cell hidden state followed by 2 fully connected layers with 32 nodes each and a final fully connected layer with nodes equal to the action space of the individual agent. The final output layer is input into a softmax function to generate a probability distribution as a representation of the agent policy.

The critic network is a feed forward neural network with 2 dense layers of 64 nodes each and an output size of 1.

We use a γ value of 0.99 in all environments. For Qmix an $\epsilon - greedy$ exploration is used with an initial value of ϵ 1 which is then annealed to 0.05 over 50k timesteps. We do not use a bounded softmax to encourage greater exploration for the policy gradient (PG) models as done in the original COMA (Foerster et al., 2018) papers as we find this to have a negligible effect on overall results while introducing unneeded additional hyperparameters.

To increase learning speed we share parameters across all agent networks and use a single network to represent all agents. A one-hot encoding of the agent type is concatenated into each agent's observations. By using the agent type as part of the observation space we can train a single shared neural network to represent all agents which have been shown to improve learning speed significantly with minor performance loss (Papoudakis et al., 2020). Both the critic and the agent networks are trained using Adam with a learning rate of $5x10^{-4}$. The EM-CAC contains a total of 32 trajectories with 4 trajectories being sampled per training step with parallel workers. The off-policy replay buffer contains the last 5000 trajectories. The target network for the critic is updated every 200 training epochs with the policy target network being updated using a weighted average of all past policies using Polyak averaging (**polyak1992acceleration)**.

We test our method every 1000 timesteps and evaluate the performance by taking the average winrate of over 8 trial runs.

5.4 Relevance Decomposition Network

5.4.1 2 step matrix game

All methods are quick to come to convergence in this simple environment. However our method is the only one able to fully converge on the correct solution with no variance as seen in figure 5.3. The independent Q-learning method overall converges a better policy than QMIX as training progresses. This could indicate that the independent method is able to more effectively perform exploration due to the training of agents not being linked to the central critic. As QMIX agents are reliant on the factorised signal from the central critic if the critic is performing inaccurate decomposition then agents are unable to learn an effective policy as they are receiving a poor representation of the actual reward signal. For independent agents, the reward signal is based on the joint policy at each timestep therefore even without value factorisation agents are able to eventually determine a higher reward policy.

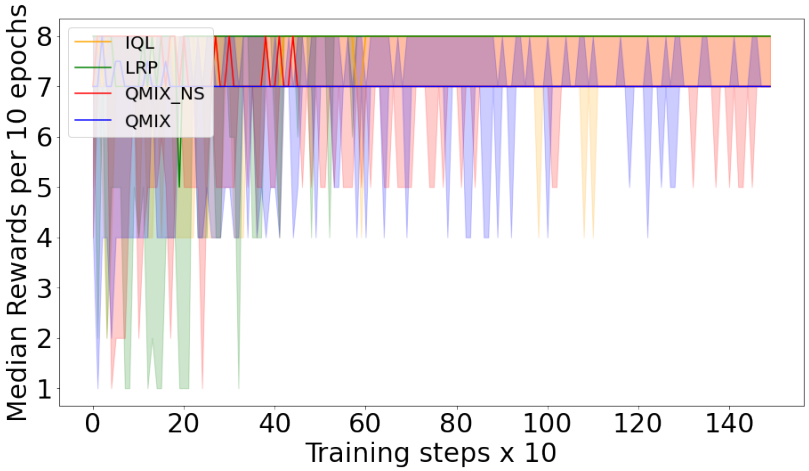


FIGURE 5.3: Median reward from 8 training runs on a 2 step grid world where shaded region is the 25th to 75th percentile range of rewards

5.4.2 SMAC

We evaluate our algorithm in both scenarios where all agents in the team are of the same unit type (homogenous) and where different agents of different unit types (heterogenous) must collaborate to solve the scenario.

TABLE 5.1: Table displaying the number of units on our agent's team and if the scenario is symmetric or not. Asymmetric environments have the same number of enemy agents as bane_vs_bane and our new environments are marked with a *

Scenario Name	Zerglings	Banelings	Symmetric
bane_no_z*	0	4	False
bane_small*	10	4	False
bane_med*	15	4	False
bane_vs_bane	20	4	True

We make use of 4 environments from *the Starcraft Multi-Agent Challenge* (SMAC) which are listed in table 5.1 along with the number of each unit in our agents' team and whether each scenario is symmetrical or not. In the asymmetrical scenarios, the team of enemy agents contains more units. In all cases, the enemy agent team contains 20 zerglings and 4 banelings whereas we alter the number of units on our agents' team to observe the effect on the performance of reducing the number of redundant agents. The preexisting map from *the Starcraft Multi-Agent Challenge* (SMAC) we make use of is the *bane_vs_bane map*. In this map, **both** sides have 20 zerglings and 4 banelings. The most optimal policy for this environment has the zerglings move out of the way so as to not obstruct the banelings' movement. This is illustrated in a sample rollout for the *bane_vs_bane* map. The agents with green bodies are the banelings and can be seen to move forward to destroy the enemy units. Banelings upon contact with enemy units will explode and deal damage to all enemy units near them. The other units are the zerglings. In figure 5.4 we can see that the zerglings move backwards to make room for the banelings to move into the enemy units. Ultimately the only important action the zerglings undertake is making way for the banelings to have access to the enemy units.

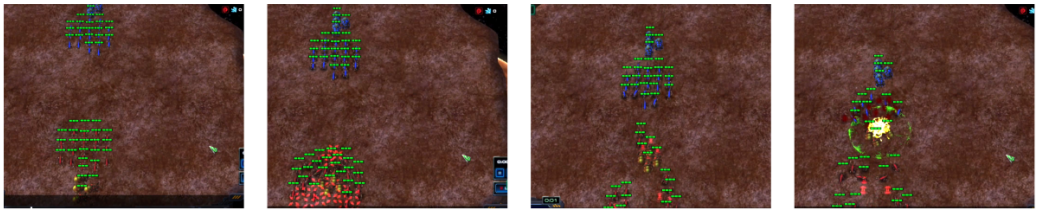


FIGURE 5.4: Example rollout in the *bane vs bane setting*

We make use of 3 additional variants of the original map which are illustrated below in 5.5. *bane_med*, *bane_small* and *bane_no_z* which have respectively decreasing numbers of zerglings which are the redundant agent in this scenario. We use a diminishing number of redundant agents in order to show how unneeded agents attempting to form cooperative policies are actually detrimental to the overall policy of the environment. Only bane_vs_bane is a symmetrical environment meaning the same number of each type of enemy and ally agent are present. In all other cases, we use the original parameter settings from banevsbane for the number of enemy units spawned while only varying the allied unit composition.

FIGURE 5.5: bane _med, _small and no_z maps

From the figure 5.6 we can see that across all maps RDN outperforms all baselines although the performance of VDN improves as the number of redundant agents is reduced. Interestingly Qmix is outperformed by Qmix_NS, a variant of Qmix that does not use state information, in *bane_med* and in *bane _no _z* indicating that in cases where there are an intermediate number of redundant agents the central state already begins to become uninformative making accurate multi-agent credit assignment difficult.

From figure 5.7 Qmix is shown to have a significantly higher average variance than all other algorithms which only use the local inputs. This is likely due to the increasing size of the ground truth state with respect to the increasing number of agents required to cooperate under the environments. Qmix_NS obtains low variance in *bane _no _z* however this is due to achieving poor results overall and therefore low fluctuations in the rewards achieved. Figures 5.8 and 5.9 confirm this as we can see for RDN and VDN the difference between the 7th and 25th percentile winrates is minor as both algorithms are unaffected by an increasingly complex state space as the number of redundant agents increases. Comparatively, Qmix displays large variance on all 4 environments as the high number of redundant agents makes the central state difficult to interpret and as the Qmix.

In the case where all redundant agents are removed, VDN performs to a similar level to LRP.

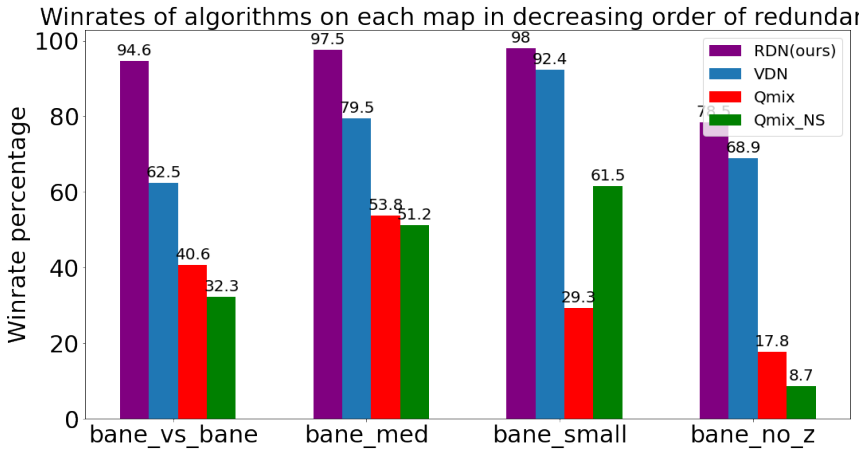


FIGURE 5.6: Percentage winrates from most number of redundant agents (left) to least redundant agents (right) for all tested algorithms

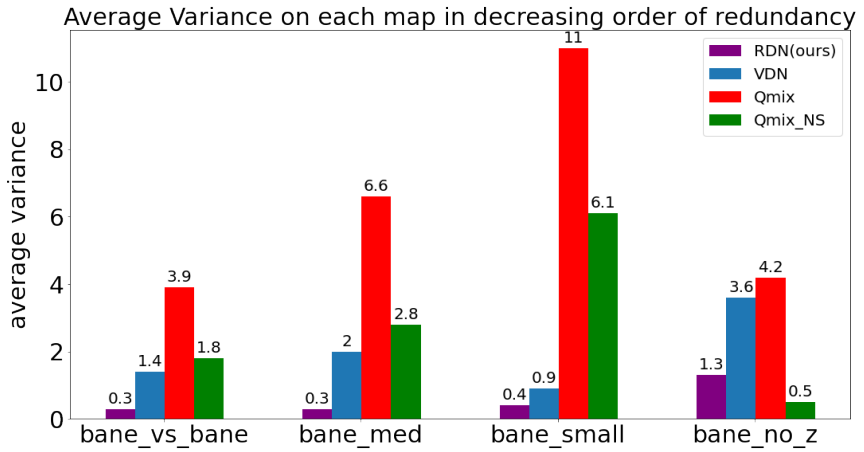


FIGURE 5.7: variance from most number of redundant agents (left) to least redundant agents (right) for all tested algorithms

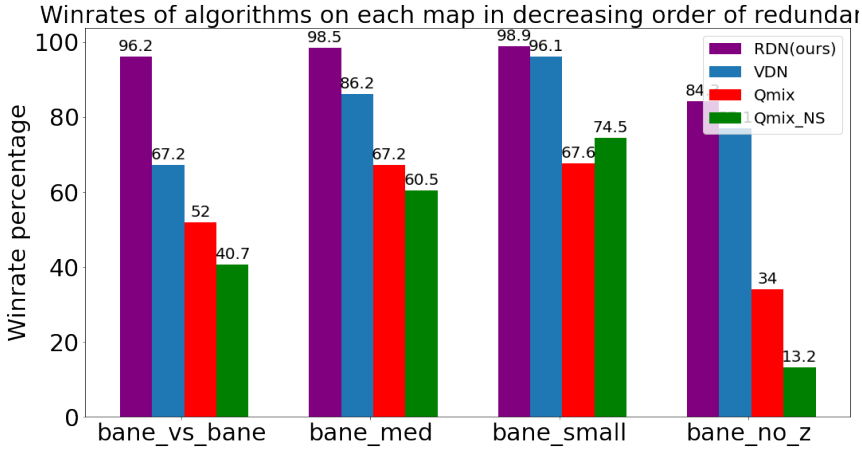


FIGURE 5.8: 75th percentile winrates from most number of redundant agents (left) to least redundant agents (right) for all tested algorithms

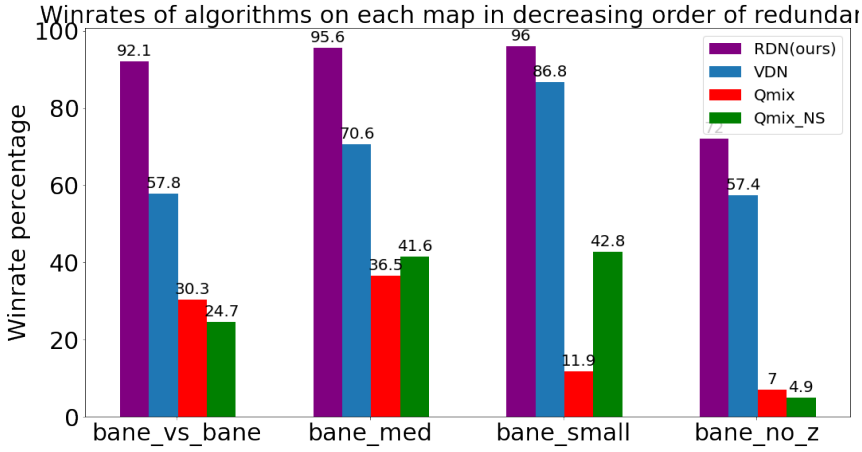


FIGURE 5.9: 25th percentile winrates from most number of redundant agents (left) to least redundant agents (right) for all tested algorithms

5.5 Extended-Masked Centralised Actor-Critic (EM-CAC)

We evaluate our algorithms in a mixture of heterogeneous environments where teams of mixed agent types must collaborate using their own traits to successfully complete the scenario, homogeneous environments where agents are all of the same type and symmetrical environments where both enemy and allied agent teams are the same and asymmetrical environments where enemy and allied agent teams are different. The groupings of these scenarios are outlined in the original SMAC paper (Samvelyan et al., 2019). Below we showcase a summary of our findings in a tabular format comparing the mean and medians of the naive-CAC method, CAC with extended replay buffer and EM-CAC with the replay buffer and the KL divergence mask. These algorithms are compared against the well-known COMA algorithm which is the most popular policy gradient based MARL method and Qmix which forms the basis for most MARL Q-learning based algorithms in the value-decomposition space.

From table 5.2 we can see that COMA exhibits generally poor performance across all testing scenarios and is surprisingly outperformed by even the naive-CAC method.

TABLE 5.2: Median and mean winrates for the last 10 sets of evaluation episodes on 4 SMAC environments

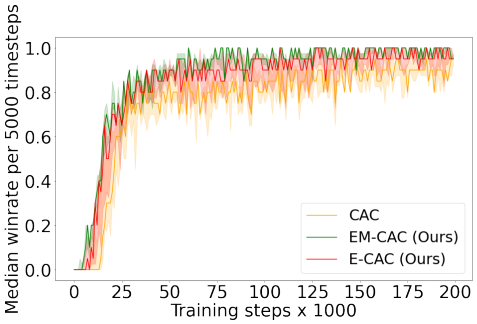
	CAC	**E-CAC**	**EM-CAC**	COMA	Qmix
MMM2	**29/35**	**43/55**	**62/65**	0/0	64/70
2c_vs_64zg	**97/100**	**95/100**	**97/100**	15/20	97/100
3s5z	**15/5**	**48/63**	**93/93**	1/0	85/90
1c3s5z	**89/90**	**95/100**	**98/100**	30	94/95

This is in line with more recent studies that show it is only applicable to a small range of low agent count environments (Papoudakis et al., 2020). From left to right we can see that the basic CAC implementation is only able to achieve good performance on the 1c3s5z and 2c_vs_64zg Whereas our fully augmented EM-CAC model is able to match the performance of Qmix on all 4 scenarios. We can see that without the mask in use variance is high with the results of the E-CAC model where the MMM2 and 3s5z results for the mean and median vary by over 10% showing that that KL divergence mask was able to stabilise the algorithm effectively during training. Interestingly the 2c_vs_64zg is classified as a hard environment in SMAC despite all algorithms aside from COMA being able to achieve a near 100% winrate on it.

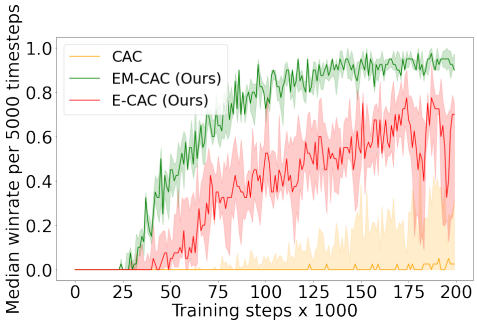
We further analyse the performance curves of the CAC based algorithms to show more clearly how the performance between the CAC and the EM-CAC method differs. From figures 5.11b and 5.10b we can see that our method EM-CAC obtains greater performance in the given 2 million training steps when compared to the naive CAC (Foerster et al., 2018) method. As the difficulty of the environment increases the increased sample efficiency of EM-CAC becomes more noticeable. In the most difficult environment *MMM2* this performance gap is most noticeable. All 3 tested algorithms show high variance in winrate however, looking at the reward curves in figures 5.13b and 5.12b instead shows that although winrate variance is high **EM-CAC** learns a high reward policy with lower variance much faster than the naive models. Additionally, we can see that As training continues the KL divergence mask greatly improves the training stability of the algorithm and not only the sample efficiency. In the *3s5z* we see similar results where the **EM-CAC** model is able to converge to a higher reward and a more policy. In the 2 easier environments in figures 5.10a and 5.11a we find that all versions of the algorithm converge to approximately equally optimal solutions.

It is important to inspect the reward curves along with winrates when using the SMAC testing environment as success or failure in its scenarios are treated as binary. Either agents have a total reward over the threshold and are successful or they are awarded a fail. By analysing the reward curves we can show that naive-CAC method does not effectively learn in the harder environments as the reward curves start to become flat before an effective policy is reached. Additionally, it shows that some of the variance observed in the winrates is the result of small deviations in policy that affect the average winrate across evaluations but do not affect the overall reward as greatly as would be expected.

In these easier environments where observing the reward curve over training, we can see in figures 5.12a and 5.13a that using a naive mixed buffer does induce greater variance in training even if it converges to a similar solution. The minor variance in these simpler tasks is offset by the large improvement in more complex ones and that when the KL divergence mask is used variance is constrained to low levels.

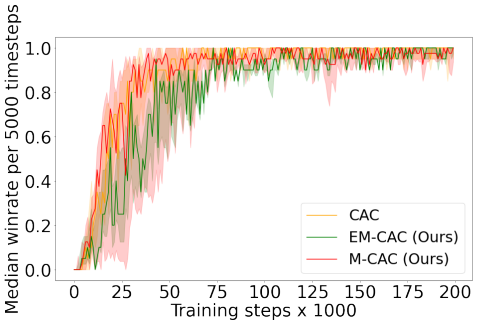


(A) Percentage winrates on the 1c3s5z map

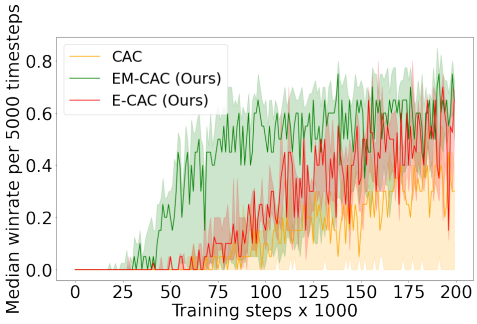


(B) Percentage winrates on the 3s5z map

FIGURE 5.10: Percentage winrates on an easy and medium SMAC map

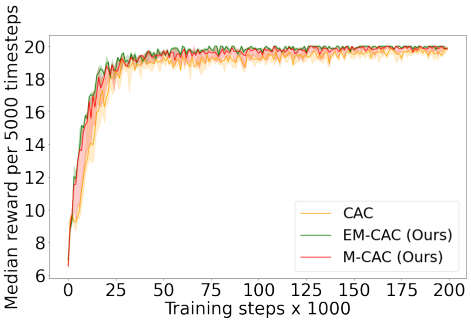


(A) Percentage winrates on the 2c_vs _64z map

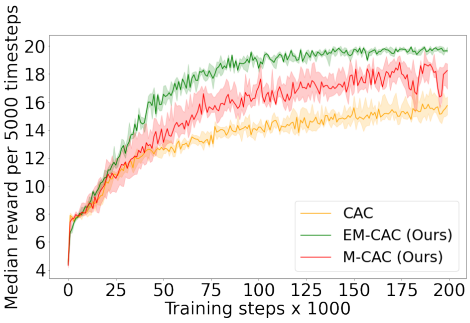


(B) Percentage winrates on the MMM2 map

FIGURE 5.11: Percentage winrates on a hard and superhard SMAC map

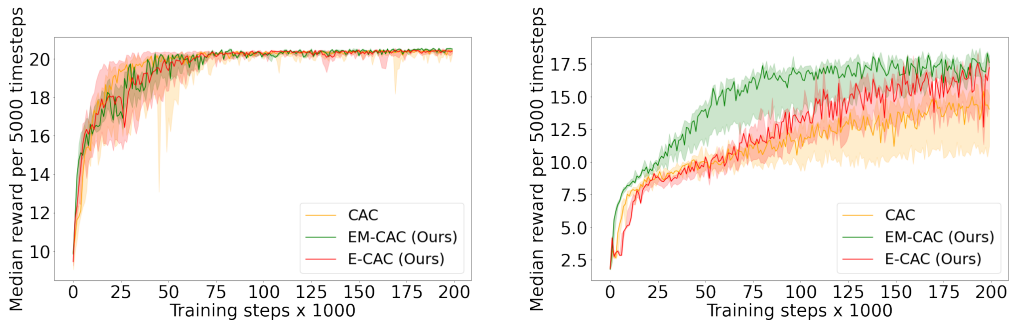


(A) Total rewards obtained on the 1c3s5z map



(B) Total rewards obtained on the 3s5z map

FIGURE 5.12: Reward over training steps on an easy and medium SMAC map



(A) Total rewards obtained on the 2c_vs _64z map (B) Percentage winrates on the MMM2 map

FIGURE 5.13: Reward over training time on a hard and superhard SMAC map

Chapter 6

Conclusion

In single-agent RL we model environments as MDPs which are then extended to POMDPs in the case of partial observability. We can generalise these into the multi-agent space using Dec-POMDPs for the case of multiple agents with partial observability.

In cooperative games we use a joint global reward. To determine individual contributions to the reward we perform value decomposition. It has been found that by conditioning on a global state in the context of local observations we can separate the global reward across agents which leads to better credit assignment. Typically this is achieved through the use of a centralised critic.

The most popular MARL value factorisation algorithm Qmix requires that the ground truth central state be available for use during training in order to achieve high performance and exhibits poor performance in scenarios with a high number of redundant agents where independent Q-learning is able to solve the environment. Additionally due to the poor sample efficiency of Policy Gradient (PG) algorithms they have seen limited development despite being naturally able to solve for these settings with only a naive centralised actor-critic (CAC).

For better credit assignment with high numbers of redundant agents and without the use of the global state we propose the method relevance decomposition network (RDN) which makes use of the Layerwise Relevance Propagation (LRP) method of network explainability to decompose the global reward without the use of the global state.

For a more sample efficient and more stable PG method, we present a modified version of the naive CAC method called extended-masked CAC (EM-CAC). This method makes use of a small amount of off-policy data to increase sample sizes during training and a KL divergence mask to suppress divergent data which would destabilise training.

We make use of 2 environments of varying difficulty in the form of the Starcraft multi agent challenge (SMAC) and a 2 step matrix game. The SMAC environment has multiple custom scenarios of differing difficulty to train on. For RDN we made our own environments with varying numbers of redundant agents to analyse the relationship between performance and redundancy in the MARL space.

Our method RDN is able to consistently converge on the simple matrix game and is largely unaffected by changes in the number of redundant agents across the custom scenarios. Qmix and VDN both exhibited large variance but VDN did show improved performance as the number of redundant agents was decreased.

EM-CAC shows that even the naive use of off-policy data in the training of PG algorithms is able to improve performance on hard-to-solve tasks. On easier tasks, the off-policy data does introduce additional variance in training in earlier stages. The KL divergence mask had a dual effect of increasing stability during training by reducing the effect of divergent updates and in combination with the off-policy

data further improved sample efficiency. This method reduces the costly nature of gathering on-policy samples and allows CAC similar performance to Qmix without requiring a larger amount of training data.

We produced the novel algorithm RDN which is able to more effectively solve highly redundant environments in the MARL setting and show how redundant agents negatively impact the performance of existing MARL algorithms by making the ground truth state increasingly difficult to interpret and train from. We also have produced a simple method which allows a naive CAC to match the performance of Qmix on 4 different SMAC tasks without requiring the additional number of training steps that would be required for PG methods.

Appendix A

Network explanation techniques

A.1 Layer-Wise Relevance Propagation (LRP)

LRP (Montavon et al., 2019) is a network explanation method that can be applied to neural network (NN) models, where inputs can be some arbitrary type of data like images, videos or text. LRP functions by propagating the prediction $f(x)$ backwards in the NN using a set of local propagation rules.

The propagation method implemented by LRP is subject to the "conservation property", where what has been recieved by the neuron of the NN must be redistributed to the immediate lower layer in equal amount.Formally let j and k be two consecutive layers of the neural network. Propagating relevance scores M_k at a given layer onto neurons of the lower layer is achieved by applying the rule:

$$M_j = \sum_k \frac{z_{jk}}{\sum_j z_{jk}} M_k. \tag{A.1}$$

Where z_{jk} models the extent to which neuron j has contributed to make the neuron k relevant with $z_k = \epsilon + \sum_{0,j} avj \cdot \rho(w_{jk})$ where ϵ is a small constant to aid numerical stability, ρ is an arbitrary function applied to the weights of the neurons like an activation function in the NN layer and v is the output from neuron j in the earlier layer of the NN to neuron k. If using the rule above for all neurons in a network we can verify the layer-wise conservation property $M_j = \sum_k \frac{z_{jk}}{\sum_j z_{jk}} M_k$ where M_j is a vector of relevance scores with each neuron $i's$ relevance scores indexed as M_{ji}, and by extension the global conservative property $\sum_i M_i = f(x)$. The overall LRP method is illustrated below in figure A.1.

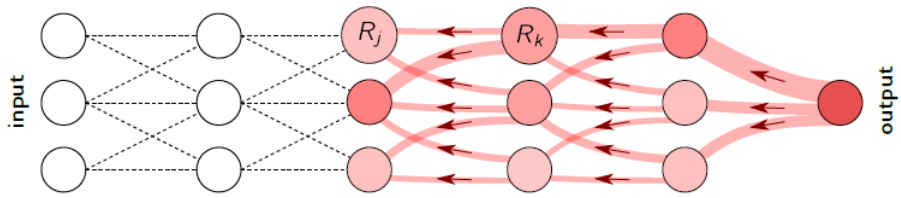


FIGURE A.1: Illustration of LRP method. Each neuron redistributes ton the lower layer the same amount as it received from the higher layer

LRP has also been applied to discover biases in common ML models and datasets (Bach et al., 2015) (Samek et al., 2017). This has also been applied to extract new insights from effective ML models and to find relevant features for audio source localisation (Perotin et al., 2019).

A.1.1 LRP Rules for Deep Rectifier Networks

In it's original publication, LRP was considered specifically for the use of NN with rectifier (ReLu) nonlinearities as these are the most common form of NN used in most research. For example in computer vision and reinforcement learning. Deep rectifier networks are composed of neurons of the type:

$$a_k = max(0, \sum_{0,j} a_j w_{jk}) \tag{A.2}$$

Where a is the activation function used in the NN layers and w is the layer weights. The sum $\sum_{0,j}$ runs over all lower-layer actions a_j, along with an extra neuron which represents the bias. Three primary propagation rules are used to calculate the relevance scores for this type of NN.

The first is the **Basic Rule (LRP-0)** (Montavon et al., 2019). This is the simplest LRP rule and redistributes in direct proportion to each input to the neuron activation function $M_j = \sum_k \frac{a_j w_{jk}}{\sum_{0,j} a_j w_{jk}} M_k$. As the gradients of NNs are typically noisy, other more robust propagation rules were designed based on LRP-0. The **Epsilon Rule (LRP-ϵ)** is the first enhancement to LRP-0 and is the rule used in Relevance Decomposition Network (RDN) (Singh & Rosman, 2021):

$$M_j = \sum_k \frac{a_j w_{jk}}{\epsilon + \sum_{0,j} a_j w_{jk}} M_k \tag{A.3}$$

The term ϵ added to the denominator of the LRP-0 calculation which adds stability to the relevance calculations by minimizing the effect of smaller scores. This leads to sparser explanations which contain less noise.

The third version of the LRP propagation rule is **Gamma Rule (LRP-γ)** (Montavon et al., 2019), which is designed to favor the effect of positive over negative relevance contributions:

$$M_j = \sum_k \frac{a_j \cdot (w_{jk} + \gamma w_{jk}^+)}{\sum_{0,j} a_j \cdot (w_{jk} + \gamma w_{jk}^+)} M_k \tag{A.4}$$

Where γ is a parameter used to control how much positive relevance contributions are favored. As *gamma* increases, negative contributions are minimized. And the prevalence of positive contributions limits how large positive and negative relevances can grow during propagation. This helps produce more stable explanations.

A.2 Integrated Gradients (IGs)

IGs are another method for network explanation however, unlike LRP they rely upon the gradients associated with network outputs to determine a relevance score (Sundararajan et al., 2017). Using gradients instead of hand-crafted rules makes them implementation invariant and, can therefore be applied to a NN irrespective of it's type. This is done by measuring the difference in the network response between the current output $f(x)$ and some series of linearly interpolated baselines.

For IGs, we are assumed to have a function $F : R^n \rightarrow [0, 1]$ that represents a NN. Let x be the current input to the NN, and $x' \in R^n$ be some baseline input. Typically the baseline is some input that results in a zero response from the NN. In the case of image detection models, for example, this could be a black image, and for RL models a terminal state.

IGs consider the straightline path from the baseline x' to the input x, and compute gradients at all points along this path. These gradients are accumulated to calculate integrated gradients. More concretely, IGs are defined as the path integral of the gradients along the straightline path from baseline x' to input x (Sundararajan et al., 2017).

The integrated gradient along the i^{th} dimension for an input x and baseline x' is defined as follows:

$$IntegratedGrads_i(x) ::= (x_i = x'_i) \times \int_{\alpha=0}^{1} \frac{\partial F(x' + \alpha \times (x - x'))}{\partial x_i} \tag{A.5}$$

Where $\frac{F(x)}{\partial x_i}$ is the gradient of function $F(x)$ along the i^th dimension of the input. IGs also have a useful property in that sum of the IGs is equal to the difference between the outputs produced by the input and baseline. This is defined as $\sum^{n}_{i=1} IntegratedGrads_i(x) = F(x) - F(x')$. To aggregate these gradients we monotonically interpolate between two points between the baseline and input. IGs make use of a straightline interpolation, but other paths also exist which is illustrated in figure A.2.

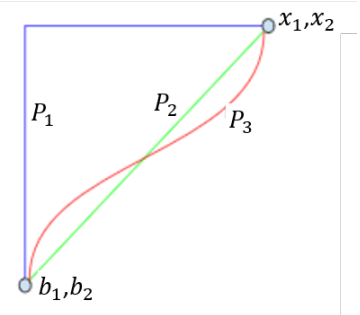


FIGURE A.2: Three paths between baseline (b_1, b_2) and input (x_1, x_2). Each path corresponds to a different attribution method. The straight line path P_2 is the path used by integrated gradients

The IGs are aggregated as a path integrated gradient using the method defined below:

- Let $x \in R^n$ be the input.
- $\gamma = (\gamma_1, ..., \gamma_n) : [0, 1] \rightarrow R^n$ be a smooth function specifying a set of counterfactuals (CFs) which are alternative inputs from some interpolation between the baseline and the actual input used to test the ANN's response to certain inputs.
- $\{\gamma(\alpha) | 0 \leq \alpha \leq 1\}$ is a set of CFs
- F is an artificial neural network

$$PathIntegratedGrads_j(x) ::= \int_{\alpha=0}^{1} \frac{\partial F(\gamma(\alpha))}{\partial \gamma_j(\alpha)} \frac{\partial \gamma_j(\alpha)}{\partial \alpha} d\alpha \tag{A.6}$$

Bibliography

Sunehag, P., Lever, G., Gruslys, A., Czarnecki, W. M., Zambaldi, V., Jaderberg, M., Lanctot, M., Sonnerat, N., Leibo, J. Z., Tuyls, K., & Graepel, T. Value-decomposition networks for cooperative multi-agent learning based on team reward. In: *Proceedings of the 17th international conference on autonomous agents and multiagent systems*. AAMAS '18. Stockholm, Sweden: International Foundation for Autonomous Agents; Multiagent Systems, 2018, 2085–2087.

Claus, C., & Boutilier, C. The dynamics of reinforcement learning in cooperative multiagent systems. In: *Proceedings of the fifteenth national/tenth conference on artificial intelligence/innovative applications of artificial intelligence*. AAAI '98/IAAI '98. Madison, Wisconsin, USA: American Association for Artificial Intelligence, 1998, 746–752. ISBN: 0262510987.

Foerster, J. N. (2018). *Deep multi-agent reinforcement learning* (Doctoral dissertation). University of Oxford. University of Oxford.

Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: An introduction 2nd ed.

Grzes, M., & Kudenko, D. Theoretical and empirical analysis of reward shaping in reinforcement learning. In: *2009 international conference on machine learning and applications*. 2009, 337–344. https://doi.org/10.1109/ICMLA.2009.33.

Foerster, J. N., Farquhar, G., Afouras, T., Nardelli, N., & Whiteson, S. Counterfactual multi-agent policy gradients. In: *Thirty-second aaai conference on artificial intelligence*. 2018.

Rashid, T., Samvelyan, M., de Witt, C. S., Farquhar, G., Foerster, J. N., & Whiteson, S. Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning. In: *Icml*. 2018.

Schwartz, J. T., & Sharir, M. (1983). On the "piano movers'" problem i. the case of a two-dimensional rigid polygonal body moving amidst polygonal barriers. *Communications on pure and applied mathematics*, *36*(3), 345–398.

Samvelyan, M., Rashid, T., de Witt, C. S., Farquhar, G., Nardelli, N., Rudner, T. G. J., Hung, C.-M., Torr, P. H. S., Foerster, J., & Whiteson, S. (2019). The StarCraft Multi-Agent Challenge. *CoRR*, *abs/1902.04043*.

Yu, C., Velu, A., Vinitsky, E., Wang, Y., Bayen, A., & Wu, Y. (2021, March). The surprising effectiveness of mappo in cooperative, multi-agent games.

Papoudakis, G., Christianos, F., Schäfer, L., & Albrecht, S. V. Benchmarking multi-agent deep reinforcement learning algorithms in cooperative tasks. In: *Proceedings of the neural information processing systems track on datasets and benchmarks (neurips)*. 2021. http://arxiv.org/abs/2006.07869

Zhou, M., Liu, Z., Sui, P., Li, Y., & Chung, Y. Y. (2020). Learning implicit credit assignment for cooperative multi-agent reinforcement learning. *Advances in Neural Information Processing Systems*, *33*, 11853–11864.

Montavon, G., Binder, A., Lapuschkin, S., Samek, W., & Müller, K.-R. (2019). Layer-wise relevance propagation: An overview. *Explainable AI: interpreting, explaining and visualizing deep learning*, 193–209.

Yang, Y., Hao, J., Liao, B., Shao, K., Chen, G., Liu, W., & Tang, H. (2020b). Qatten: A general framework for cooperative multiagent reinforcement learning. *arXiv preprint arXiv:2002.03939*.

Arjona-Medina, J. A., Gillhofer, M., Widrich, M., Unterthiner, T., Brandstetter, J., & Hochreiter, S. Rudder: Return decomposition for delayed rewards. In: *Advances in neural information processing systems*. 2019, 13544–13555.

Wang, Z., Bapst, V., Heess, N., Mnih, V., Munos, R., Kavukcuoglu, K., & de Freitas, N. (2016). Sample efficient actor-critic with experience replay. *arXiv preprint arXiv:1611.01224*.

Singh, S, & Rosman, B. The challenge of redundancy on multi-agent value factorisation. In: *Neurips workshop on cooperative ai*. 2021.

Hausknecht, M., & Stone, P. (2015). Deep recurrent q-learning for partially observable mdps. *arXiv preprint arXiv:1507.06527*.

Kaelbling, L. P., Littman, M. L., & Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, *101*(1), 99–134. https://doi.org/https://doi.org/10.1016/S0004-3702(98)00023-X

Cassandra, A. R. A survey of pomdp applications. In: *Working notes of aaai 1998 fall symposium on planning with partially observable markov decision processes*. *1724*. 1998.

Singh, A. G. J. P. (1995). Learning to act using real-time dynamic programming. *Artificial Intelligence*, *72*(1-2), 81–138.

Busoniu, L., Babuska, R., & De Schutter, B. (2008). A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, *38*(2), 156–172. https://doi.org/10.1109/TSMCC.2007.913919

Matignon, L., Laurent, G., & Fort-Piat, N. (2012). Independent reinforcement learners in cooperative markov games: A survey regarding coordination problems. *The Knowledge Engineering Review*, *27*, 1 –31. https://doi.org/10.1017/S0269888912000057

Nowé, A., Vrancx, P., & Hauwere, Y.-M. D. Game theory and multi-agent reinforcement learning. In: *Reinforcement learning*. 2012.

Tuyls, K., & Weiss, G. (2012). Multiagent learning: Basics, challenges, and prospects. *Ai Magazine*, *33*, 41–52. https://doi.org/10.1609/aimag.v33i3.2426

Fudenberg, D., & Tirole, J. (1991). *Game theory*. MIT press.

Oliehoek, F. A. (2012). Decentralized pomdps. In M. Wiering & M. van Otterlo (Eds.), *Reinforcement learning: State-of-the-art* (pp. 471–503). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-27645-3_15

Lowe, R., Wu, Y., Tamar, A., Harb, J., Abbeel, P., & Mordatch, I. (2017). Multi-agent actor-critic for mixed cooperative-competitive environments. *Neural Information Processing Systems (NIPS)*.

Albawi, S., Mohammed, T. A., & Al-Zawi, S. Understanding of a convolutional neural network. In: *2017 international conference on engineering and technology (icet)*. 2017, 1–6. https://doi.org/10.1109/ICEngTechnol.2017.8308186.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013a). Playing atari with deep reinforcement learning. https://doi.org/10.48550/ARXIV.1312.5602

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013b). Playing atari with deep reinforcement learning.

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

Bellemare, M. G., Naddaf, Y., Veness, J., & Bowling, M. (2013). The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, *47*, 253–279. https://doi.org/10.1613/jair.3912

Bellemare, M. G., Naddaf, Y., Veness, J., & Bowling, M. (2013). The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, *47*, 253–279.

Schaul, T., Quan, J., Antonoglou, I., & Silver, D. (2015). Prioritized experience replay.

Hessel, M., Modayil, J., van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., & Silver, D. (2017). Rainbow: Combining improvements in deep reinforcement learning.

Watkins, C. (1989). *Learning from delayed rewards* (Techinical Report ECS-LFCS-98-397,PhD Thesis). King's College, Cambridge.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, *8*(3-4), 229–256.

Schulman, J., Moritz, P., Levine, S., Jordan, M., & Abbeel, P. (2018). High-dimensional continuous control using generalized advantage estimation.

Sutton, R. S., McAllester, D., Singh, S., & Mansour, Y. (1999). Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, *12*.

Schulman, J., Moritz, P., Levine, S., Jordan, M., & Abbeel, P. (2015). High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*.

Singh, A., Jain, T., & Sukhbaatar, S. (2018). Learning when to communicate at scale in multiagent cooperative and competitive tasks. *arXiv preprint arXiv:1812.09755*.

Tampuu, A., Matiisen, T., Kodelja, D., Kuzovkin, I., Korjus, K., Aru, J., Aru, J., & Vicente, R. (2015). Multiagent cooperation and competition with deep reinforcement learning.

Foerster, J., Nardelli, N., Farquhar, G., Afouras, T., Torr, P. H. S., Kohli, P., & Whiteson, S. Stabilising experience replay for deep multi-agent reinforcement learning. In: *Proceedings of the 34th international conference on machine learning - volume 70*. ICML'17. Sydney, NSW, Australia: JMLR.org, 2017, 1146–1155.

Littman, M. (2001). Value-function reinforcement learning in markov games. *Cognitive Systems Research*, *2*, 55–66. https://doi.org/10.1016/S1389-0417(01)00015-8

Littman, M. L. Markov games as a framework for multi-agent reinforcement learning. In: *Proceedings of the eleventh international conference on international conference on machine learning*. ICML'94. New Brunswick, NJ, USA: Morgan Kaufmann Publishers Inc., 1994, 157–163. ISBN: 1558603352.

Ha, D., Dai, A., & Le, Q. V. (2016). Hypernetworks. *arXiv preprint arXiv:1609.09106*.

Gilpin, L. H., Bau, D., Yuan, B. Z., Bajwa, A., Specter, M., & Kagal, L. (2019). Explaining explanations: An overview of interpretability of machine learning.

Yang, Y., Hao, J., Chen, G., Tang, H., Chen, Y., Hu, Y., Fan, C., & Wei, Z. (2020a). Q-value path decomposition for deep multiagent reinforcement learning. *arXiv preprint arXiv:2002.03950*.

Dizet, A., Bono, C. L., Legeleux, A., Buche, C., et al. (2021). Robocup@ home education 2020 best performance: Robobreizh, a modular approach. *arXiv preprint arXiv:2107.02978*.

Papoudakis, G., Christianos, F., Schäfer, L., & Albrecht, S. V. (2020). Comparative evaluation of multi-agent deep reinforcement learning algorithms. *CoRR*, *abs/2006.07869*. https://arxiv.org/abs/2006.07869

Espeholt, L., Soyer, H., Munos, R., Simonyan, K., Mnih, V., Ward, T., Doron, Y., Firoiu, V., Harley, T., Dunning, I., et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In: *International conference on machine learning*. PMLR. 2018, 1407–1416.

Hu, J., Jiang, S., Harding, S. A., Wu, H., & wei Liao, S. (2021). Revisiting the monotonicity constraint in cooperative multi-agent reinforcement learning.

Foerster, J. N., Assael, Y. M., de Freitas, N., & Whiteson, S. (2016). Learning to communicate to solve riddles with deep distributed recurrent q-networks.

Espeholt, L., Soyer, H., Munos, R., Simonyan, K., Mnih, V., Ward, T., Doron, Y., Firoiu, V., Harley, T., Dunning, I., et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In: *International conference on machine learning*. PMLR. 2018, 1407–1416.

Sandholm, T. W., & Crites, R. H. On multiagent q-learning in a semi-competitive domain. In: *International joint conference on artificial intelligence*. Springer. 1995, 191–205.

Cobbe, K. W., Hilton, J., Klimov, O., & Schulman, J. Phasic policy gradient. In: *International conference on machine learning*. PMLR. 2021, 2020–2027.

Yu, R., Shi, Z., Wang, X., Wang, R., Liu, B., Hou, X., Lai, H., & An, B. (2019). Inducing cooperation via team regret minimization based multi-agent deep reinforcement learning. *arXiv preprint arXiv:1911.07712*.

Wang, T., Dong, H., Lesser, V., & Zhang, C. Roma: Multi-agent reinforcement learning with emergent roles. In: *Proceedings of the 37th international conference on machine learning*. 2020.

Mahajan, A., Rashid, T., Samvelyan, M., & Whiteson, S. (2019). Maven: Multi-agent variational exploration. *arXiv preprint arXiv:1910.07483*.

Bach, S., Binder, A., Montavon, G., Müller, K., & Samek, W. (2015). Analyzing classifiers: Fisher vectors and deep neural networks. *CoRR*, *abs/1512.00172*. http://arxiv.org/abs/1512.00172

Samek, W., Binder, A., Lapuschkin, S., & Müller, K.-R. Understanding and comparing deep neural networks for age and gender classification. In: *2017 ieee international conference on computer vision workshops (iccvw)*. 2017, 1629–1638. https://doi.org/10.1109/ICCVW.2017.191.

Perotin, L., Serizel, R., Vincent, E., & Guerin, A. (2019). Crnn-based multiple doa estimation using acoustic intensity features for ambisonics recordings. *IEEE Journal of Selected Topics in Signal Processing*, *PP*, 1–1. https://doi.org/10.1109/JSTSP.2019.2900164

Sundararajan, M., Taly, A., & Yan, Q. (2017). Axiomatic attribution for deep networks. https://doi.org/10.48550/ARXIV.1703.01365