ME3000: Independent Study 1

Final Report

Area Defence and Tag with Smart Drones

Student Name:

Chong Yu Quan

Student Number:

A0136286Y

Student Email:

chong.yuquan@u.nus.edu

May 14, 2022

Contents

List of Figures

Acknowledgements

I want to express my sincerest gratitude to Assistant Professor Guillaume Sartoretti for his patient guidance throughout my final year project. Through his invaluable advice, I have gained significant knowledge and broadened my horizons on various facets of multi-agent reinforcement learning, for which I am incredibly grateful to have the opportunity.

Chapter 1

Introduction

Unmanned air vehicles (UAV) have always been integral to defence systems for applications such as surveillance and logistics. They are usually a component of an unmanned aircraft system (UAS) that includes a ground-based controller and a system of communications with the UAV, with varying levels of partial autonomy. However, with the advent of artificial intelligence (AI), the system capabilities based on UAVs can expand towards full autonomy not only for a single agent but for multi-agent scenarios in addressing a common task. Following this new frontier of drone autonomy, the project aims to design and develop AI systems for drones to act intelligently and collaboratively to achieve common goals. With the potential to achieve beyond human-level performance while eliminating the need for human operators, such a system would be vital to any future defence infrastructure.

Multi-agent reinforcement learning (MARL) algorithms have shown great potential for developing such systems. For example, DeepMind's MARL algorithm, AlphaStar [3], trained to play the popular real-time strategy game, StarCraft II, is ranked at Grandmaster level for all three Starcraft races and above 99.8% of officially ranked human players. OpenAI's MARL algorithm, OpenAI Five [4], trained to play popular massively multiplayer online role-playing game (MMORPG) Dota 2, becomes the first AI to beat the world champions, Team OG, in an esports game. While those achievements are incredible, a closer inspection of the model architectures of the algorithms highlights several areas for exploration in future work that could potentially improve the algorithms' performance in general to various other tasks. Specifically, a prospective area to be explored would be the incorporation of Graph Neural Networks

(GNN) [5–10] into the model architecture. An existing issue with feed-forward neural networks in MARL algorithms is that they require feature vectors representing the state of the environment as inputs to the model, which often requires manual feature engineering. Given the universality of graph structures in describing complex data, GNNs can efficiently handle task-independent feature learning for machine learning.

1.1 Literature Review

1.1.1 Multi-Agent Reinforcement Learning

MARL algorithms generally fall into independent learning (IL), centralised multi-agent policy gradient, and value decomposition classes, where the latter two follow the Centralised Training Decentralised Execution (CDTE) framework. In IL, each agent learns independently and perceives the other agents as part of the environment. Examples of IL algorithms include the Independent synchronous Advantage Actor-Critic (IA2C) and Independent Proximal Policy Optimisation (IPPO) algorithms. IA2C is a version of the Advantage Actor-Critic (A2C) algorithm [11] for decentralised multi-agent reinforcement learning. Each agent has an actor model to approximate the policy and a critic model to approximate the value function. Both actor and critic are trained based on the history of local observations, actions and rewards the agent perceives to minimise the A2C loss. Similarly, IPPO is a version of the Proximal Policy Optimisation (PPO) algorithm [12] for decentralised multi-agent reinforcement learning. It has an identical model architecture to IA2C. The only difference is that IPPO uses a surrogate objective that constrains the relative change of the policy at each update, allowing for more update epochs using the same batch of trajectories. In contrast to IPPO, IA2C can only perform one update epoch per batch of trajectories to ensure that the training batch remains on-policy.

Compared to IL, CTDE allows sharing information during training, with decentralised execution based on the agent's local observations for policy generation. Hence, CTDE algorithms perform better in general than IL algorithms as they can leverage the joint information of all agents to train more efficiently. Therefore, the IL algorithms are not part of the considered

MARL algorithms for this project. The value decomposition subset of the CTDE algorithms focuses on decomposing the joint state-action value function into individual state-action value functions. Examples of value decomposition algorithms include Value Decomposition Networks (VDN) [13] and QMIX [14] algorithms. VDN aims to learn a linear decomposition of the joint state-action value (Q-value). Each agent maintains a network to approximate its Q-values. VDN decomposes the joint Q-value into the sum of individual Q-values. The joint state-action value function trains using the Deep Q Learning (DQN) [15] algorithm, where gradients of the joint temporal difference (TD) loss flow backwards to the network of each agent. QMIX builds on VDN to address a broader class of environments requiring a more complex decomposition. A parameterised mixing network computes the joint Q-value based on each agent's state-action value function. The mixing function requires that the optimal joint action, which maximises the joint Q-value, is the same as the combination of the individual actions maximising the Q-values of each agent. Like VDN, QMIX trains to minimise the DQN loss where the gradient backpropagates to the individual Q-values.

However, value-based algorithms cannot accommodate a continuous action space as that would require an infinite number of outputs from the model, hence necessitating the discretisation of the action space to work. Such discretisation is non-ideal as manual hyperparameter tuning requires significant effort. On the other hand, the centralised policy gradient methods subset of the CDTE algorithms is capable of accomodating a continuous actions space. Each agent consists of a decentralised actor model and a centralised critic model, optimised based on joint information between the agents. Examples of centralised policy gradient methods algorithms include the Multi-Agent Deep Deterministic Policy Gradients (MADDPG) [16], Multi-Agent A2C (MAA2C) and Multi-Agent PPO (MAPPO) [17]. MADDPG is a multi-agent version of the Deep Deterministic Policy Gradients (DDPG) [18] algorithm for MARL. The actor model trains on a replay buffer of local observations, and the critic model trains on the joint observation and action to approximate the joint state-action value function. Each agent minimises the deterministic policy gradient loss [19], where the differentiability of actions with respect to the actor's parameters is assumed, resulting in continuous action space. MAA2C is an actor-critic algorithm in which the critic learns the joint state value function compared to the state-action value function in MADDPG. It extends the existing on-policy actor-critic algorithm A2C by applying centralised critics conditioned on the state of the environment rather than the individual history of observations. MAPPO is an actor-critic algorithm that extends on IPPO in which the critic learns a joint state value function similar to MAA2C. In contrast to MAA2C, which can only perform one update epoch per training batch, MAPPO can utilise the same training batch of trajectories to perform several update epochs.

1.1.2 Graph Neural Networks

Nevertheless, the MARL algorithms stated in the previous section have model architectures based mainly on feed-forward neural networks requiring manual feature engineering to obtain the feature vectors as inputs. Otherwise, the model architecture could also include convolutional neural networks (CNN) based on grid structured data such as images. However, many complex data, such as molecular structures or social networks, do not fall into those two narrow categories. For such cases, a graph data structure would be the most appropriate in encapsulating the data. Hence, to extend existing neural networks for processing data represented in graph domains, the GNN was proposed. The features and related nodes define a node in a graph, and the target of GNN is to learn a state embedding, which contains the information of the neighbourhood for each node. The state embedding can then produce an output such as the feature vectors serving as inputs to the feed-forward neural networks.

In GNN, the propagation step and output step are vital in the model to obtain the hidden states of nodes or edges and variants of GNNs arises from differences in aggregators that gather information from each node's neighbours and updaters that update nodes' hidden states. GNN that focuses on convolutions in the graph domain falls between spectral approaches and non-spectral (spatial) approaches in general. Spectral approaches work with a spectral representation of the graphs, where the convolution operation operates in the Fourier domain by computing the eigendecomposition of the graph Laplacian. An example would be the Graph Convolutional Network (GCN) [6].

However, in general, spectral methods require learned filters that depend on the Laplacian eigenbasis dependent on the graph structure. Therefore, a model trained on a specific graph structure cannot translate well to graphs with different structures. Hence, spectral based GNN is non-ideal for this project. On the other hand, non-spectral approaches, such as GraphSAGE [7], define convolutions directly on the graph, operating on spatially close neighbours. While non-spectral approaches accommodate different graph structures, the primary challenge of non-spectral approaches is defining the convolution operation with differently sized neighbourhoods and maintaining the local invariance of CNNs.

Building on the non-spectral approaches, several works have implemented the gate mechanism like the Gated Recurrent Unit (GRU) [20] or Long Short Term Memory (LSTM) [21] in the propagation step to diminish the restrictions in the former GNN models and improve the long-term propagation of information across the graph structure. For example, the Gated Graph Neural Network (GGNN) [9] uses GRU in the propagation step, unrolls the recurrence for a fixed number of timesteps and uses backpropagation through time to compute gradients. Like the GRU, LSTM applied in the propagation process based on a tree or graph gives GNN such as Graph LSTM [10]. The stated gate mechanisms in GNN are worth exploring in cases of a potential large graph structure (due to a large number of drones) where the GNN would benefit significantly from the improved long-term propagation of information across the graph structure.

Building on the success of the attention mechanism in many sequence-based tasks such as machine translation and machine reading, the Graph Attention Network (GAT) [8] proposed incorporates the attention mechanism into the propagation step. It computes the hidden states of each node by attending over its neighbours, following a self-attention strategy. The implementation of the attention mechanism in the GAT architecture is highly efficient due to parallelisation and highly applicable to graph nodes of different degrees and inductive learning problems. Hence, the GAT shows excellent promise in efficiently encapsulating the features of the state of the environment.

1.2 Gap, Objective and Significance

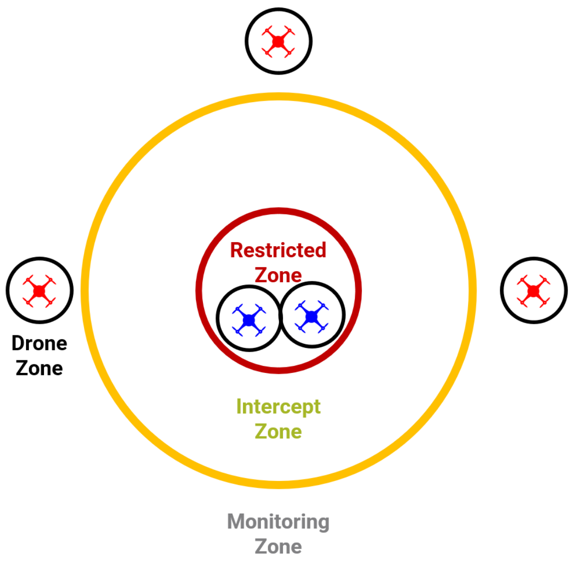
The project aims to design and develop an AI system for a team of smart drones to act intelligently and collaboratively to achieve the common goal of guarding a restricted zone with high-value assets against opposing teams of drones. It seeks to achieve the objective by incorporating GNNs into the latest state-of-the-art MARL algorithms in the development of the learning agents. The project plans to conduct the training of the learning agent in a virtual environment, scoping the possible scenarios to the cases of a one versus one and a two versus two scenarios. The lack of existing literature for the stated objective highlights this as an unexplored research area. The project hypothesises that incorporating the GNNs should improve state representation of the environment, consequently improving model performance. If successful, this project provides the basic building block towards developing such a system for real drones, which is extremely valuable to any future defence infrastructure.

Chapter 2

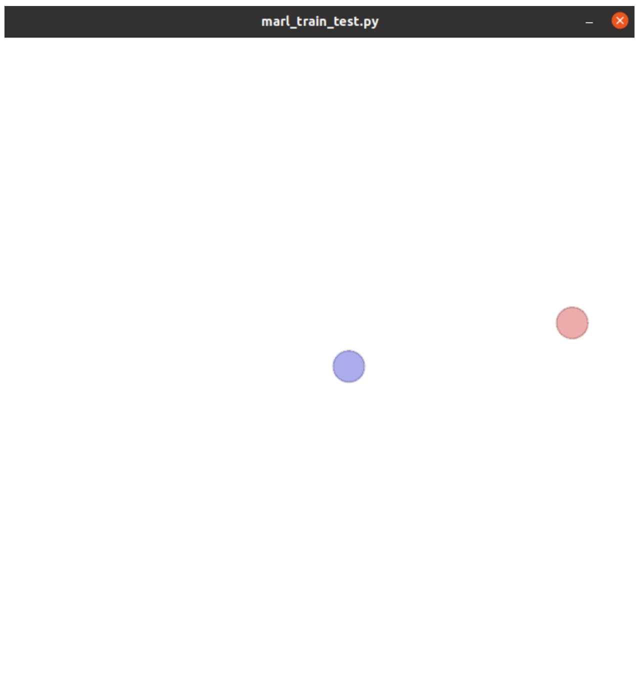
Methodology

The following sections highlight the problem statement before delving into the specific details of the three algorithms used in this project: MADDPG, MADDPGv2 and MAPPO.

2.1 Problem Statement



(A) Illustration of training environment



(B) MPE environment for 1 vs 1 case

Figure 2.1: Training environment

There are x blue drones and y red drones in the training environment and three zones: monitoring, intercept and restricted zones, as illustrated in Figure 2.1a. The objective of the blue drones is to keep the red drones away from the restricted drones by a specified time limit, while the objective of the red drones is to infiltrate the restricted zone. In addition, blue drones can disable red drones by colliding with them. A disabled drone cannot move at all (like a landmark) and cannot communicate. The environment randomly spawns the blue drones in the restricted zone and red drones in the monitoring zone. In addition, potential landmarks can be spawned randomly between the restricted and intercept zones.

Each drone is assumed to have a computer vision system that provides obstacle detection with depth perception, translating to a drone zone where the drone is perceptive of its local surroundings. Furthermore, the environment has a radar that works for the blue drones that detect all entities (e.g. opponents and landmarks) with a specified radar noise. Both blue and red drones also know their teammates' specific kinematic states (e.g. position and velocity). If a drone's teammates are in its drone zone, the drone receives information with no noise under the assumption of a perfect drone vision system. Else, information received would include a specified noise.

The training environment developed based on the stated scenario above utilises the Multi-Agent Particle Environment (MPE) [16,22] founded upon the OpenAI Gym [23], a simple multi-agent, 2D particle world with continuous observation, discrete action space, and fundamental simulated physics. Figure 2.1b highlights the rendered environment during training.

2.2 Reinforcement Learning Cast

This section highlights the details of the observations (o_i), actions (a_i) and rewards (r_i) for an learning agent i. Firstly, the kinematic state of an agent i is defined as the coordinates of the agent relative to the geometric centre, (p^i_x, p^i_y), and the velocity of the agent, (v^i_x, v^i_y). The communication state of an agent i consist of the number of opposing agents in agent i drone zone, n_i. The observation of agent i, o_i, consists of the following: the time elapsed in the episode, coordinates of landmarks (if any) relative to the geometric centre, the kinematic state of agent i without any radar noise, the communication state of agent i, the kinematic states of agent i teammates (with radar noise if beyond drone zone) and communication states of agent i's teammates. In addition, if agent i is a blue drone, o_i includes the kinematic states with radar noise and the communication states of the opponents as well. If agent i is a red drone, zeros are appended instead.

The actions of agent i, a_i, consists kinematic actions in terms of forces, (F_x^i, F_y^i) and communication actions in terms of number of opposing agents, n_i, in agent i's drone zone. The rewards for agent i, r_i, follows a sparse reward structure where rewards are shared across all agents of the same team. If the red drones succeed in their objective, a large positive reward (e.g. 10) is given to all red drones, and a large negative reward (e.g. -10) is given to all blue drones. The opposite occurs if blue drones succeed in their objective. A common reward for red and blue drones is a large negative reward for all drones in the same team as the drone that exits the screen.

2.3 Actor Architecture

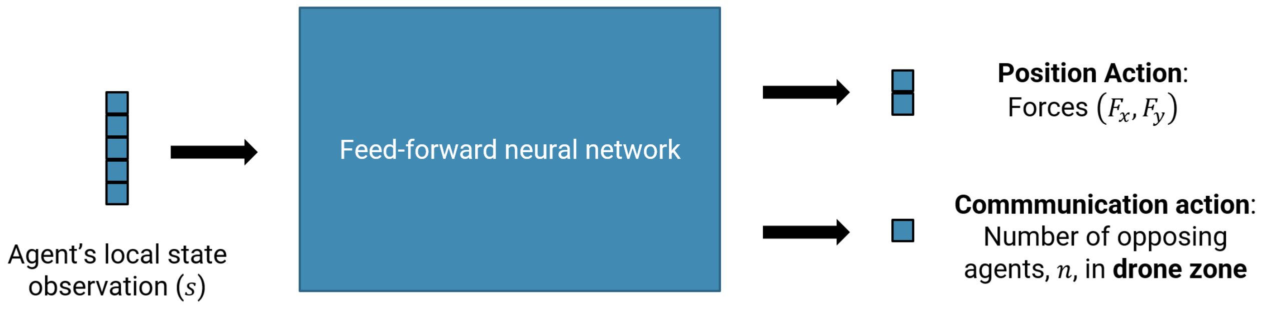

Agent's local state observation (s)
Feed-forward neural network
Position Action: Forces (F_x, F_y)
Commmunication action: Number of opposing agents, n, in **drone zone**

FIGURE 2.2: Architecture of the actor model

The architecture for the actor model follows a standard feed-forward neural network. The inputs to the neural network are the agent's local state observation, s, which consist of the time elapsed during the episode, landmark positions, the agent's, its teammates' and its opponents' kinematic and communication states, with the specified radar noise stated previously included accordingly. For invalid observations, such as that of the blue drones' kinematic state outside of a red drone's drone zone, the observations are replaced with zeros instead. The outputs are the position and communication actions, which are the forces in the x and y direction, (F_x, F_y), and the number of opposing agents, n, in the agent's drone zone. The communication actions are equivalent to the communication state of the agent.

2.4 Critic Architecture

The architecture of the critic model generally consists of the Graph Attention Network Version 2 (GATv2) [24], Graph Multiset Transformer (GMT) [1] and a standard feed-forward neural network as shown in Figure 2.3 and 2.4. For MADDPG, the inputs consist of the global observation, $\boldsymbol{x} = [o_1, o_2, \ldots, o_n], o_i \in \mathcal{S}$, which are the local observations of all agents in the same team in a fully connected graph as well as the actions of all agents, $\boldsymbol{y} = [a_1, a_2, \ldots, a_n], a_i \in \mathcal{A}$. On the other hand, for MAPPO, the inputs are only the global observation, x. Hence, the MADDPG critic model outputs the centralised state-action value, $Q(\boldsymbol{x}, \boldsymbol{y})$, while the MAPPO critic model outputs the centralised state value $V(\boldsymbol{x})$. The following subsections elaborate on GATv2 and GMT in detail.

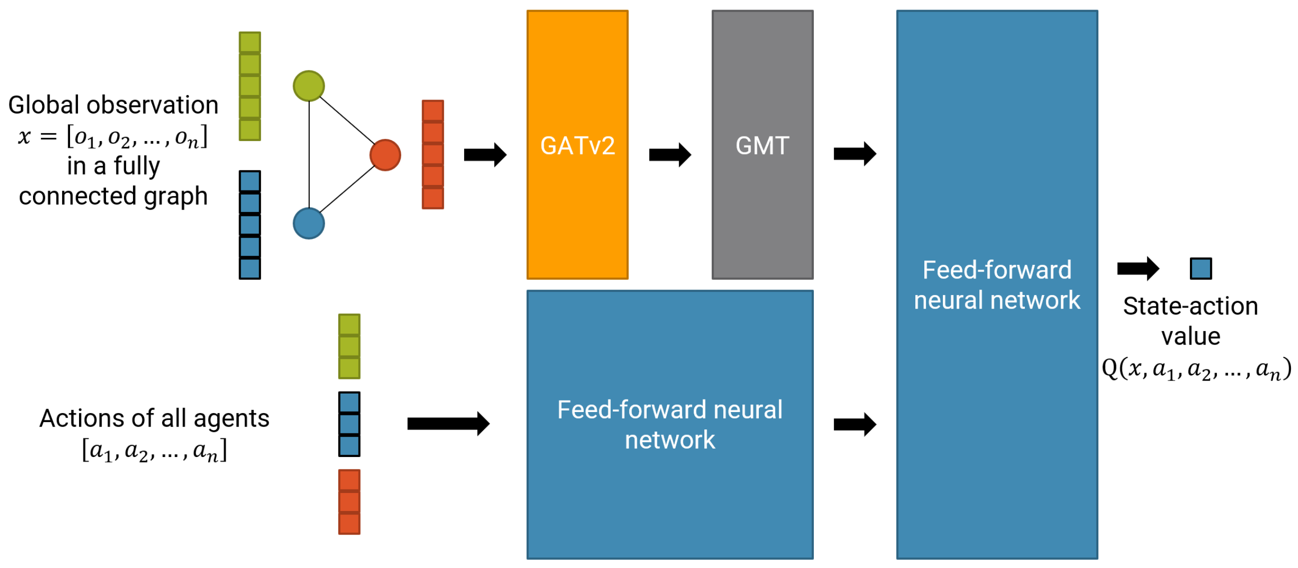


FIGURE 2.3: Architecture of the MADDPG critic model

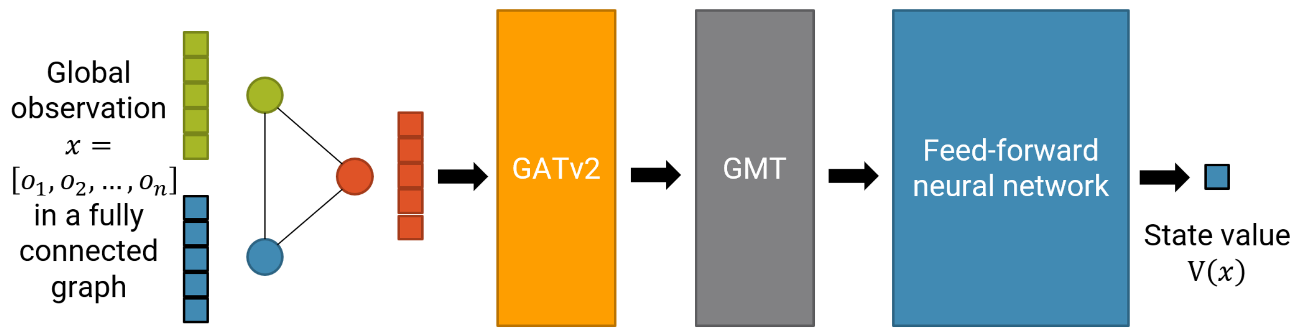


FIGURE 2.4: Architecture of the MAPPO critic model

2.4.1 GNN

In general, a graph, $\mathcal{G}$, is represented by the node set $\mathcal{V}$ with $|\mathcal{V}| = n$ nodes and its adjacency matrix $\boldsymbol{A} \in \{0,1\}^{n\times n}$. Each node has c dimensional node features, $\boldsymbol{X} \in \mathbb{R}^{n\times c}$. A GNN layer updates every node representation by aggregating its neighbors' representations as illustrated in message-passing function shown in equation 2.1.

$$\boldsymbol{H}_u^{(l+1)} = \text{UPDATE}^{(l)}\left(\boldsymbol{H}_u^{(l)}, \text{AGGREGATE}^{(l)}\left(\{\boldsymbol{H}_v^{(l)}, \forall v \in \mathcal{N}(u)\}\right)\right) \tag{2.1}$$

$\boldsymbol{H}_u^{(l+1)} \in \mathbb{R}^{n\times d}$ are the node features computed after l steps of the GNN that can be simplified as $\boldsymbol{H}^{(l+1)} = \text{GNN}^{(l)}\left(\boldsymbol{H}^{(l)}, \boldsymbol{A}^{(l)}\right)$. $\mathcal{N}(u)$ highlights the set of neighbouring nodes of u and $\boldsymbol{H}_u^{(1)} = \boldsymbol{X}_u$. UPDATE and AGGREGATE are different functions that typically distinguishes one type of GNN from the other. For example, GraphSAGE [7] uses a neural network as UPDATE and an element-wise mean as AGGREGATE.

2.4.2 GATv2

As many popular GNN architectures weigh all neighbours, $\forall v \in \mathcal{N}(u)$, with equal weightage through their AGGREGATE functions, GAT [8] utilises a form of learned weighted average for representations of $\mathcal{N}(u)$ to address this limitation. It first devises a scoring function, $e : \mathbb{R}^{1\times d} \times \mathbb{R}^{1\times d} \to \mathbb{R}$ that computes a score for every edge (u, v), which indicates the importance of the features of the neighbor v to the node u as shown in equation 2.2, where $\boldsymbol{h}_u \in \mathbb{R}^{1\times d}$ and $\boldsymbol{h}_v \in \mathbb{R}^{1\times d}$ are the node features for nodes u and v respectively, $\boldsymbol{a} \in \mathbb{R}^{2d'\times 1}$, $\boldsymbol{W} \in \mathbb{R}^{d\times d'}$ and $\|$ denotes vector concatenation.

$$e(\boldsymbol{h}_u, \boldsymbol{h}_v) = \text{LeakyReLU}(\boldsymbol{a}^\top \cdot [\boldsymbol{h}_u\boldsymbol{W} \,\|\, \boldsymbol{h}_v\boldsymbol{W}]) \tag{2.2}$$

The attention coefficients, α_{uv}, are then obtained as shown in equation 2.3 through normalisation using the softmax function across all neighbours, $\forall v \in \mathcal{N}(u)$.

$$\alpha_{uv} = \text{softmax}(e(\boldsymbol{h}_u, \boldsymbol{h}_v)) = \frac{\exp(e(\boldsymbol{h}_u, \boldsymbol{h}_v))}{\sum_{v'\in\mathcal{N}(u)} \exp(e(\boldsymbol{h}_u, \boldsymbol{h}_{v'}))} \tag{2.3}$$

GAT then computes the weighted average of the transformed features of the neighbour nodes using the attention coefficients followed by a non-linearity function, σ, as shown in equation 2.4.

$$\boldsymbol{h}'_u = \sigma\left(\sum_{v\in\mathcal{N}(u)} \alpha_{uv}\cdot \boldsymbol{h}_v\boldsymbol{W}\right) \tag{2.4}$$

However, GAT has its limitations as its shown by Brody et al. [24] to compute only static attention rather than dynamic attention. Static attention is defined as a (possibly finite) family of scoring functions $\mathcal{F} \subseteq (\mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R})$ that computes the static scoring for a given set of key vectors $\mathbb{K} = \{\boldsymbol{k}_1, \ldots, \boldsymbol{k}_n\} \subset \mathbb{R}^d$ and query vectors $\mathbb{Q} = \{\boldsymbol{q}_1, \ldots, \boldsymbol{q}_n\} \subset \mathbb{R}^d$, if for every $f \in \mathcal{F}$ there exist a maximal key $v_f \in [n]$ such that for every query $u \in [m]$ and key $v \in [n]$ it holds that $f(\boldsymbol{q}_u, \boldsymbol{k}_{v_f}) \geq f(\boldsymbol{q}_u, \boldsymbol{k}_v)$. Dynamic attention is defined as a (possibly finite) family of scoring functions $\mathcal{F} \subseteq (\mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R})$ that computes the dynamic scoring for a given set of key vectors $\mathbb{K} = \{\boldsymbol{k}_1, \ldots, \boldsymbol{k}_n\} \subset \mathbb{R}^d$ and query vectors $\mathbb{Q} = \{\boldsymbol{q}_1, \ldots, \boldsymbol{q}_n\} \subset \mathbb{R}^d$, if for every mapping $\varphi : [m] \to [n]$ there exists $f \in \mathcal{F}$ such that for every query $u \in [m]$ and key $v_{\neq\psi(u)} \in [n]$, it holds that $f(\boldsymbol{q}_u, \boldsymbol{k}_{\varphi(u)}) > f(\boldsymbol{q}_u, \boldsymbol{k}_v)$.

Static attention is limited in expressivity because every function $f \in \mathcal{F}$ has a key that is always selected regardless of the query, hence unable scenarios where different keys have different relevance to different queries. On the other hand, dynamic attention can select every key $\varphi(u)$ using the query u by making $f(\boldsymbol{q}_u, \boldsymbol{k}_{\varphi(u)})$ maximal in $\{f(\boldsymbol{q}_u, \boldsymbol{k}_v)|v \in [n]\}$. The fact that GAT computes static attention can be illustrated from equation 2.2 by rewriting it to equation 2.5, where the learned parameter, $\boldsymbol{a}$, is represented as a concatenation of $\boldsymbol{a}_1, \boldsymbol{a}_2 \in \mathbb{R}^{d'\times 1}$, $\boldsymbol{a} = [\boldsymbol{a}_1 \parallel \boldsymbol{a}_2]$.

$$e(\boldsymbol{h}_u, \boldsymbol{h}_v) = \text{LeakyReLU}(\boldsymbol{a}_1^\top \boldsymbol{h}_u\boldsymbol{W} + \boldsymbol{a}_2^\top \boldsymbol{h}_v\boldsymbol{W}) \tag{2.5}$$

Since $\mathcal{V}$ is finite, it can be observed from equation 2.5 that there exists a node $v_{max} \in \mathcal{V}$ such that $\boldsymbol{a}_2^\top \boldsymbol{h}_{v_{max}}\boldsymbol{W}$ is maximal for all nodes $v \in \mathcal{V}$. Due to monotonicity of LeakyReLu and softmax, for every query node $u \in \mathcal{V}$, the node $\boldsymbol{h}_{v_{max}}$ results in the maximal value of its attention coefficients distribution $\{\alpha_{uv}|v \in \mathcal{N}(u)\}$, which corresponds to the definition of static attention. According to Brody et al. [24], the issue for GAT lies in the fact that the learned layers $\boldsymbol{W}$ and $\boldsymbol{a}$ are applied consecutively that effectively makes it a single layer. To address this issue, they recommended applying the $\boldsymbol{a}$ layer after the non- linearity, and the $\boldsymbol{W}$ layer after concatenation of the input node features. This achieves dynamic attention (stated in this report without proof) to give rise to GATv2 and is shown in equation 2.6.

$$e(\boldsymbol{h}_u, \boldsymbol{h}_v) = \boldsymbol{a}^\top \text{LeakyReLU}([\boldsymbol{h}_u \parallel \boldsymbol{h}_v] \cdot \boldsymbol{W}) \tag{2.6}$$

2.4.3 GMT

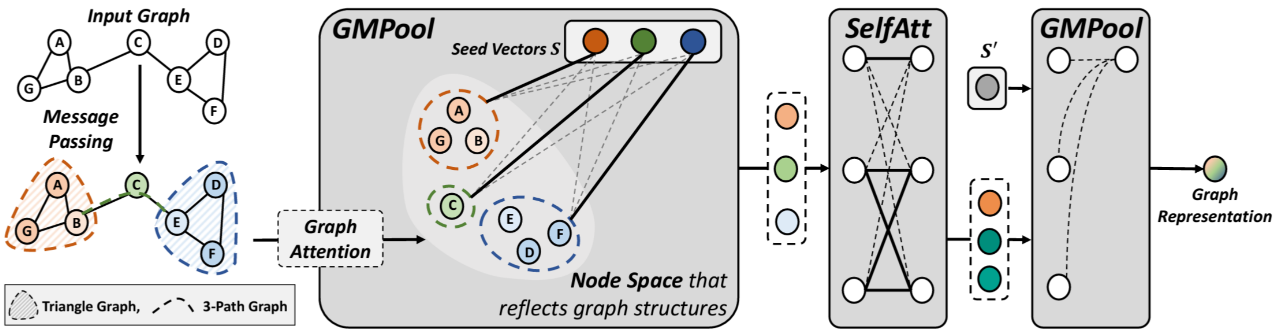


FIGURE 2.5: Architecture of the Graph Multiset Transformer adapted from [1]

As shown in Figure 2.5, the overall architecture of GMT by Baek et al. [1] consists of two key components: Graph Multiset Pooling (GMPool) and Self-Attention function (SelfAtt), which the following subsections elaborate.

GMPool

Like GATv2, GMPool utilises the attention mechanism as a central component of the pooling mechanism to address the limitations of simple pooling methods (e.g. mean) in distinguishing vital nodes. The output of the attention function, Att, is shown in equation 2.7, where $\boldsymbol{Q} \in \mathbb{R}^{n_q \times d_k}$, $\boldsymbol{K} \in \mathbb{R}^{n \times d_k}$ and $\boldsymbol{V} \in \mathbb{R}^{n \times d_v}$ are the query, key and value inputs respectively, d_k and d_v are the dimensions of the key and value vector respectively, n_q is the number of query vectors, n is the number of input nodes and w is an activation function.

$$\text{Att}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = w\left(\boldsymbol{Q}\boldsymbol{K}^\top\right)\boldsymbol{V} \tag{2.7}$$

Going beyond a single attention, multi-head attention [25] can be implemented by linearly projecting $\boldsymbol{Q}$, $\boldsymbol{K}$, $\boldsymbol{V}$ h times to obtain h different subspaces. The output of the multi-head attention function, MH, is shown in equation 2.8, where $\boldsymbol{W}_i^Q \in \mathbb{R}^{d_k \times d_k}$, $\boldsymbol{W}_i^K \in \mathbb{R}^{d_k \times d_k}$, $\boldsymbol{W}_i^V \in \mathbb{R}^{d_v \times d_v}$, $\boldsymbol{W}^O \in \mathbb{R}^{hd_v \times d_m}$ and d_m are the output dimensions of the MH function.

$$\begin{gathered} O_i = \mathrm{Att}(\boldsymbol{Q}\boldsymbol{W}_i^Q, \boldsymbol{K}\boldsymbol{W}_i^K, \boldsymbol{V}\boldsymbol{W}_i^V) \\ \mathrm{MH}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = [O_1 \parallel \cdots \parallel O_h]\boldsymbol{W}^O \end{gathered} \tag{2.8}$$

However, the MH function is suboptimal in generating $\boldsymbol{K}$ and $\boldsymbol{V}$ as they are linearly projected from H. To address this limitation, Baek et al. defined a novel graph multi-head attention function (GMH) [1]. With node features, $\boldsymbol{H}$ and adjacency information, $\boldsymbol{A}$, GMH utilises GNN to leverage on the graph structure to generate key and value that contains neighbouring information of the graph compared to the linearly projected $\boldsymbol{K}\boldsymbol{W}_i^K$ and $\boldsymbol{V}\boldsymbol{W}_i^V$ in equation 2.8. The computation of GMH is shown in equation 2.9.

$$\begin{gathered} O_i = \mathrm{Att}(\boldsymbol{Q}\boldsymbol{W}_i^Q, \mathrm{GNN}_i^K(\boldsymbol{H}, \boldsymbol{A}), \mathrm{GNN}_i^V(\boldsymbol{H}, \boldsymbol{A})) \\ \mathrm{GMH}(\boldsymbol{Q}, \boldsymbol{H}, \boldsymbol{A}) = [O_1 \parallel \cdots \parallel O_h]\boldsymbol{W}^O \end{gathered} \tag{2.9}$$

With GMH, Baek et al. defined GMPool [1] that considers the interactions between k seed vectors (queries) in $\boldsymbol{S} \in \mathbb{R}^{k \times d}$, a parameterised seed matrix optimised in an end-to-end fashion, and n nodes (keys) to compress n nodes to k representative nodes based on their attention similarities between queries and keys. GMPool is shown in equation 2.10, where rFF is any row-wise feedforward layer that processes each individual row independently and identically and LN is layer normalisation [26].

$$\begin{gathered} \boldsymbol{Z} = \mathrm{LN}(\boldsymbol{S} + \mathrm{GMH}(\boldsymbol{S}, \boldsymbol{H}, \boldsymbol{A})) \\ \mathrm{GMPool}_k(\boldsymbol{H}, \boldsymbol{A}) = \mathrm{LN}(\boldsymbol{Z} + \mathrm{rFF}(\boldsymbol{Z})) \end{gathered} \tag{2.10}$$

SelfAtt

$$\begin{gathered} \boldsymbol{Z} = \mathrm{LN}(\boldsymbol{H} + \mathrm{GMH}(\boldsymbol{H}, \boldsymbol{H}, \boldsymbol{H})) \\ \mathrm{SelfAtt}(\boldsymbol{H}) = \mathrm{LN}(\boldsymbol{Z} + \mathrm{rFF}(\boldsymbol{Z})) \end{gathered} \tag{2.11}$$

While GMPool condenses n nodes to k representative nodes. it does not consider relationships between nodes. To address this limitation, Baek et al. defined SelfAtt as shown in equation 2.11 that encapsulates the inter-relationships between n nodes by using the node embeddings, $\boldsymbol{H}$, to generate both key and query.

Overall Architecture

$$\text{Pooling}(\boldsymbol{H}, \boldsymbol{A}) = \text{GMPool}_1(\text{SelfAtt}(\text{GMPool}_k(\boldsymbol{H}, \boldsymbol{A})), \boldsymbol{A}') \tag{2.12}$$

The overall architecture of GMT is shown in equation 2.12, where GMT aggregates the features into a single vector form, Pooling : $\boldsymbol{H}, \boldsymbol{A} \rightarrow \boldsymbol{h}_G \in \mathbb{R}^{1\times d}$. The order of operation of the GMT involves first condensing the entire graph, $\mathcal{G}$, into k representative nodes using GMPool. SelfAtt is then used to account for the interactions between the k representative nodes. The entire graph representation is then obtained using GMPool with $k = 1$, where $\boldsymbol{A}' \in \mathbb{R}^{k\times k}$ is the identity or coarsened adjacency matrix as $\boldsymbol{A}$ should be updated after condensing n nodes to k representative nodes with GMPool$_\text{k}$.

2.5 MADDPG

For a scenario with N agents, the set of the agent polices that are typically stochastic are given by $\boldsymbol{\pi}_\theta = \{\boldsymbol{\pi}_{\theta_1}, \ldots, \boldsymbol{\pi}_{\theta_N}\}$, with the corresponding policies parameters given by the set $\boldsymbol{\theta} = \{\theta_1, \ldots, \theta_N\}$. From the policy gradient theorem [27], the gradient of the expected return for agent i, $J(\boldsymbol{\pi}_{\theta_i}) = \mathbb{E}(R_i)$ is shown in equation 2.13, where $p^{\boldsymbol{\pi}_\theta}$ is the stationary distribution of states under $\boldsymbol{\pi}_\theta$.

$$\nabla_{\theta_i} J(\boldsymbol{\pi}_{\theta_i}) = \mathbb{E}_{\boldsymbol{x}\sim p^{\boldsymbol{\pi}_\theta}, a_i \sim \boldsymbol{\pi}_{\theta_i}} [\nabla_{\theta_i} \text{log} \boldsymbol{\pi}_{\theta_i}(a_i|o_i) Q_i^{\boldsymbol{\pi}_\theta}(\boldsymbol{x}, a_1, \ldots, a_N)] \tag{2.13}$$

For deterministic policies, a similar theorem, deterministic policy gradient theorem [19], can be applied. For N continuous deterministic policies, $\boldsymbol{\mu}_\theta = \{\boldsymbol{\mu}_{\theta_1}, \ldots, \boldsymbol{\mu}_{\theta_N}\}$ with parameters θ_i, the deterministic policy gradient is shown in equation 2.14, where $\mathcal{D}$ is the experience replay buffer containing experiences of all agents stored as tuples $(\boldsymbol{x}, \boldsymbol{x}', a_1, \ldots, a_N, r_1, \ldots, r_N)$ and sampled in a off-policy manner with policy β.

$$\nabla_{\theta_i} J(\boldsymbol{\mu}_{\theta_i}) = \mathbb{E}_{\boldsymbol{x}, a \sim \mathcal{D}^\beta} \left[\nabla_{\theta_i} \boldsymbol{\mu}_{\theta_i}(a_i | o_i) \nabla_{a_i} Q_i^{\boldsymbol{\mu}}(\boldsymbol{x}, a_1, \ldots, a_N)|_{a_i = \boldsymbol{\mu}_{\theta_i}(o_i)} \right] \tag{2.14}$$

$Q_i^{\boldsymbol{\mu}}$ is updated using equation 2.15, where $\boldsymbol{\mu}'_\theta = \{\boldsymbol{\mu}_{\theta'_1}, \ldots, \boldsymbol{\mu}_{\theta'_N}\}$ is the set of target policies with delayed parameters, $\boldsymbol{\theta}' = \{\theta'_1, \ldots, \theta'_N\}$.

$$\begin{aligned} y &= r_i + \gamma Q_i^{\boldsymbol{\mu}'_\theta}(\boldsymbol{x}', a_1, \ldots, a_N)|_{a_j = \boldsymbol{\mu}'_{\theta_j}(o_j)} \\ \mathcal{L}(\theta_i) &= \mathbb{E}_{\boldsymbol{x}, a, r, \boldsymbol{x}'} \left[(Q^{\boldsymbol{\mu}_\theta}(\boldsymbol{x}, a_1, \ldots, a_N) - y)^2 \right] \end{aligned} \tag{2.15}$$

2.6 MADDPGv2

MADDPGv2 is a proposed extension of MADDPG in order to deal with sparsity of rewards from the environment without the need for complicated and domain-specific reward engineering. MADDPGv2 incorporates the concept of multi-goal learning through the usage of Universal Value Function Approximators (UVFA) [28] coupled with Hindsight Experience Replay (HER) [2]. Results from Andrychowicz et al. [2] highlights that training an agent to perform multiple tasks can be easier than training it to perform only one task, even if there is only one task we would like the agent to perform.

Let $\mathcal{G}$ be the space of all possible goals, where every goal $g \in \mathcal{G}$ corresponds to a reward function $r_g : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$. Every episode begins by sampling a state-goal pair from some distribution $p(\boldsymbol{x}_0, g)$, where the goal remains constant for the whole episode. At every timestep, t, the agent i gets the state input o_i as well as the current goal, g, $\boldsymbol{\pi}_{\theta_i} : \mathcal{S} \times \mathcal{G} \rightarrow \mathcal{A}$ and obtains the reward $r_t = r_g(o_i^t, g)$. Similarly, state-action value, $Q^{\boldsymbol{\mu}}(\boldsymbol{x}, a_1, \ldots, a_N)$, does not depend only on a state-action pair but also on a goal as well, $Q^{\boldsymbol{\mu}}(\boldsymbol{x}, a_1, \ldots, a_N, g)$.

```
Algorithm 1 Hindsight Experience Replay (HER)
  Given:
    • an off-policy RL algorithm $\mathbb{A}$,                    ▷ e.g. DQN, DDPG, NAF, SDQN
    • a strategy $\mathbb{S}$ for sampling goals for replay,      ▷ e.g. $\mathbb{S}(s_0, \ldots, s_T) = m(s_T)$
    • a reward function $r : \mathcal{S} \times \mathcal{A} \times \mathcal{G} \to \mathbb{R}$.   ▷ e.g. $r(s, a, g) = -[f_g(s) = 0]$
  Initialize $\mathbb{A}$                                          ▷ e.g. initialize neural networks
  Initialize replay buffer $R$
  for episode = $1, M$ do
      Sample a goal $g$ and an initial state $s_0$.
      for $t = 0, T - 1$ do
          Sample an action $a_t$ using the behavioral policy from $\mathbb{A}$:
                  $a_t \leftarrow \pi_b(s_t||g)$                   ▷ || denotes concatenation
          Execute the action $a_t$ and observe a new state $s_{t+1}$
      end for
      for $t = 0, T - 1$ do
          $r_t := r(s_t, a_t, g)$
          Store the transition $(s_t||g,\ a_t,\ r_t,\ s_{t+1}||g)$ in $R$      ▷ standard experience replay
          Sample a set of additional goals for replay $G := \mathbb{S}$(current episode)
          for $g' \in G$ do
              $r' := r(s_t, a_t, g')$
              Store the transition $(s_t||g',\ a_t,\ r',\ s_{t+1}||g')$ in $R$     ▷ HER
          end for
      end for
      for $t = 1, N$ do
          Sample a minibatch $B$ from the replay buffer $R$
          Perform one step of optimization using $\mathbb{A}$ and minibatch $B$
      end for
  end for
```

FIGURE 2.6: Pseudocode for HER for a single agent adapted from [2]

In HER, every set of experience from an episode is stored not only with the original goal used for the episode but also with a subset of other goals and their corresponding rewards. The pseudocode for HER for a single agent shown in Figure 2.6 formally expresses the concept. It is important to note that the goal being pursued in each episode influences an agent's actions but not the environment dynamics. Hence, each trajectory can be replayed with an arbitrary goal assuming that an off-policy algorithm such as MADDPG is used.

2.7 MAPPO

A drawback of policy gradients methods is that they often empirically result in extensive destructive policy updates without gradient clipping methods. MAPPO addresses this issue by introducing surrogate losses that can be computed and differentiated with a minor change to a typical policy gradient implementation. In MAPPO [17], the critic network generating the centralised state value $V_{\phi_i}(\boldsymbol{x})$, with parameters ϕ_i for agent i, is trained to minimise the loss function, $\mathcal{L}(\phi_i)$, shown in equation 2.16, where B is the batch size and $\hat{R}_j$ is the discounted reward-to-go that is the discounted sum of rewards after timestep j.

$$\begin{aligned}\mathcal{L}'(\phi_i) &= \Big(\text{clip}\left(V_{\phi_i}(\boldsymbol{x}_j), V^{old}_{\phi_i}(\boldsymbol{x}_j) - \epsilon, V^{old}_{\phi_i}(\boldsymbol{x}_j) + \epsilon\right) - \hat{R}_j\Big)^2 \\ \mathcal{L}(\phi_i) &= \frac{1}{B}\sum_{j=1}^{B} \max\left(\left(V_{\phi_i}(\boldsymbol{x}_j) - \hat{R}_j\right)^2, \mathcal{L}'(\phi_i)\right)\end{aligned} \tag{2.16}$$

The actor network generating the policy, $\boldsymbol{\pi}_{\theta_i}$, with parameters θ_i for agent i, is trained to maximise the loss function, $\mathcal{L}(\theta_i)$, shown in equation 2.17, where B is the batch size, $r_j^{\theta_i}$ is the policy probability ratio for agent i at timestep j, A_j^i is the is an estimator of the advantage function for agent i at timestep j, ϵ is the policy probability ratio clipping hyperparaeter, S is the policy entropy and σ is the entropy coefficient hyperparameter.

$$\begin{aligned}\mathcal{L}(\theta_i) = &\frac{1}{B}\sum_{j=1}^{B} \min\left(r_j^{\theta_i} A_j^i, \text{clip}(r_j^{\theta_i}, 1-\epsilon, 1+\epsilon) A_j^i\right) \\ &+ \sigma\frac{1}{B}\sum_{j=1}^{B} S\left(\boldsymbol{\pi}_{\theta_i}\left(o_j^i\right)\right)\end{aligned} \tag{2.17}$$

The definition for A_j^i is given in equation 2.18, which is a truncated version of the Generalised Advantage Estimation (GAE) [29]. The hyperparameter λ controls the trade-off between the bias and variance for A_j^i. If $\lambda = 0$, we obtain the temporal difference (TD) advantage estimate, δ_t^i, which is high in bias but low in variance. If $\lambda = 1$, we obtain an extended advantage estimate that is low in bias but high in variance.

$$\begin{aligned}\delta_t^i &= r_t^i + \gamma V_{\phi_i}(\boldsymbol{x}_{t+1}) - V_{\phi_i}(\boldsymbol{x}_t) \\ A_t^i &= \delta_t^i + (\gamma\lambda)\delta_{t+1}^i + \cdots + (\gamma\lambda)^{T-t+1}\delta_{T-1}^i\end{aligned} \tag{2.18}$$

The definition of $r_j^{\theta_i}$ is given in equation 2.19. The first term in the min term in equation 2.17

represents the surrogate loss used in trust region policy optimization (TRPO) [30]. The second term in the min term modifies the surrogate objective by removing the incentive for $r_j^{\theta_i}$ for leaving the interval $[1-\epsilon, 1+\epsilon]$. The min term then takes the lower bound between the unclipped and clipped objectives to penalise large policy updates in general.

$$r_j^{\theta_i} = \frac{\boldsymbol{\pi}_{\theta_i}(a_j^i|o_j^i)}{\boldsymbol{\pi}_{\theta_i^{old}}(a_j^i|o_j^i)} \tag{2.19}$$

Chapter 3

Results & Discussion

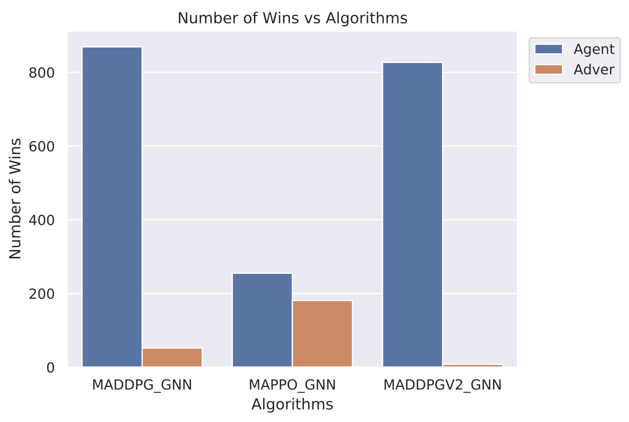
The following sections details the experiments performed with in-depth discussions of the results.

3.1 Experiments

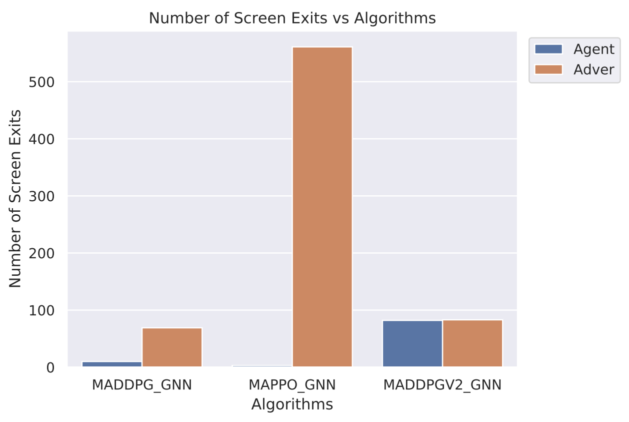
3.1.1 Preliminary experiments

1 vs 1

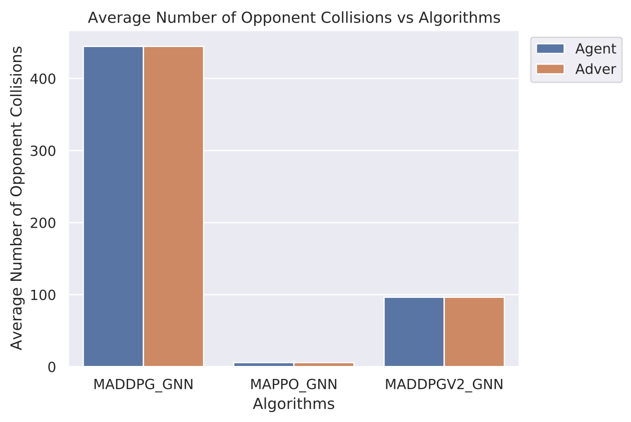
Preliminary studies for the 1 vs 1 case were conducted to select the best algorithm available to focus on. Figure 3.1 highlights the results for 1000 episodes for the blue (Agent) and red (Adver) drones for the three algorithms after training for 5000 episodes with an episode time-step limit of 30 time-steps. The results from Figure 3.1a show that the blue drones succeed at a significantly higher rate than the red drones for all three algorithms with the exception of MAPPO. Hence, the optimal policy for the red drones can be reasoned to be significantly much more challenging to achieve than the blue drones' optimal policy. This can be generally attributed to the blue drone's ability to disable the red drone, resulting in the red drone's objective being significantly more difficult to achieve. In addition to having comparatively lower success rate relative to MADDPG and MADDPGv2, MAPPO also has a tendency to exit the screen with zero collisions between red and blue drones, as shown in Figure 3.1b and 3.1c. Hence, it is highly likely that the wins accrued by the MAPPO red and blue drones result from random luck rather than deliberate policy. The lack of convergence to optimality for red and



(A) Wins



(B) Screen exits

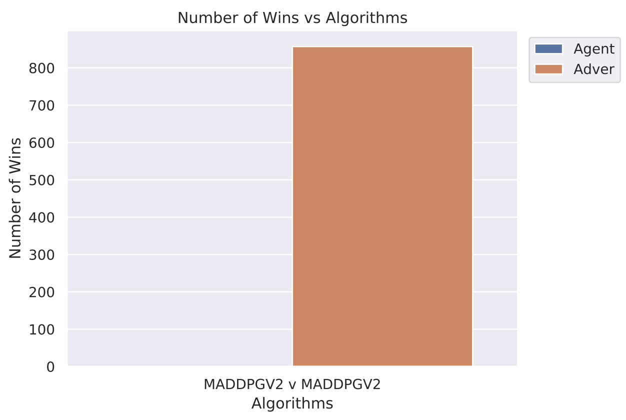


(C) Average number of collisions with opponents (per agent)

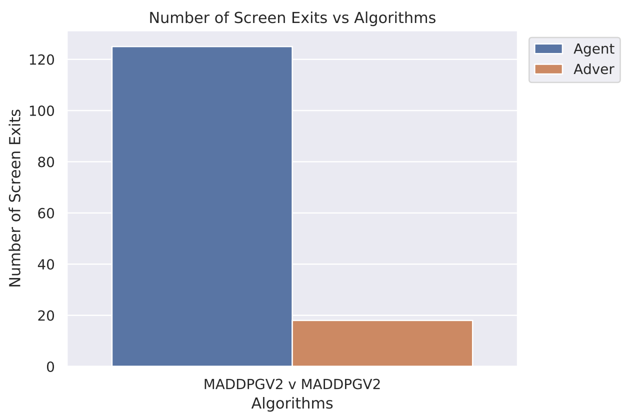
FIGURE 3.1: Key results for preliminary studies

blue drones trained using MAPPO may be because the policy clipping significantly slows down learning for red and blue drones. Hence, for MAPPO to become a viable algorithm, significant time and effort must be invested to tune the hyperparameters. Nevertheless, the policy for the red and blue drones trained using MADDPG and MADDPGv2 are reasonably optimal given their success rate and ability to avoid exiting the screen, as shown in Figure 3.1a and 3.1b. Among these two algorithms, MADDPGv2 is selected as the key algorithm of focus under as HER was found to enhance the quality of the replay buffer for the red drones. Various evasive manoeuvres to throw the blue drone of the chase executed by the red drone were found for MADDPGv2 but not for MADDPG when rendering the environment. The significantly lower number of average number of collisions with opponents per agent for MADDPGv2 compared to MADDPG provides strong evidence for this claim as shown in Figure 3.1c.

3.1.2 Multi-Agent (2 vs 2)



(A) Wins

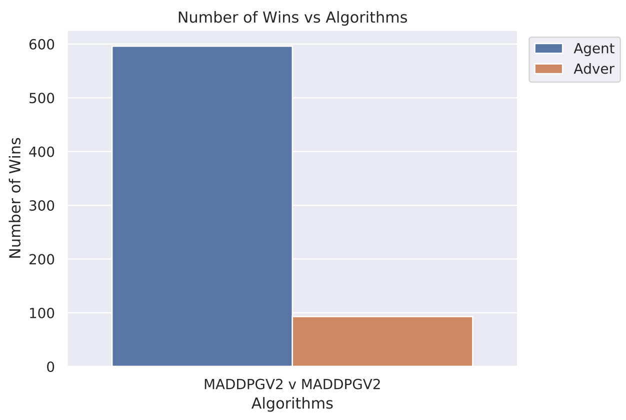


(B) Screen exits

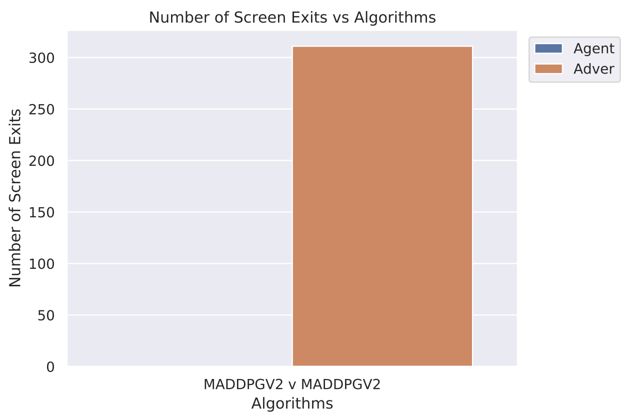
FIGURE 3.2: 2 vs 2 results usng MADDPGv2

Figure 3.2 highlights the results for the 2 vs 2 experiments for 1000 episodes after training for 10000 episodes using MADDPGv2, where the results contradict the initial hypothesis. From Figure 3.2a, it can be observed that the red drones still completely dominates the blue drones in general. The relatively high number of screen exits as shown in Figure 3.2b for the blue drones confirms that the policy for the blue drones is far from optimality.

3 vs 3



(A) Wins



(B) Screen exits

FIGURE 3.3: 3 vs 3 results usng MADDPGv2

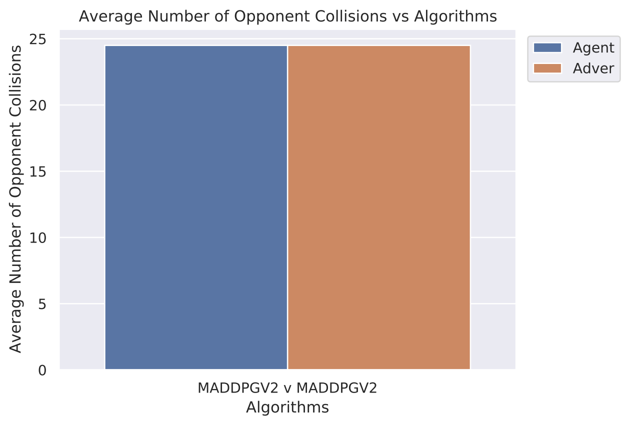
Figure 3.3 highlights the results for the 3 vs 3 experiments for 1000 episodes after training for 10000 episodes using MADDPGv2. From Figure 3.3a, it can be observed the blue drones demonstrates a high level of dominance over red drones unlike all previous experiments.

`images/1_vs_1_10_time_steps_average_number_of_opp_collisions_vs_algorithms_barplot.pdf`

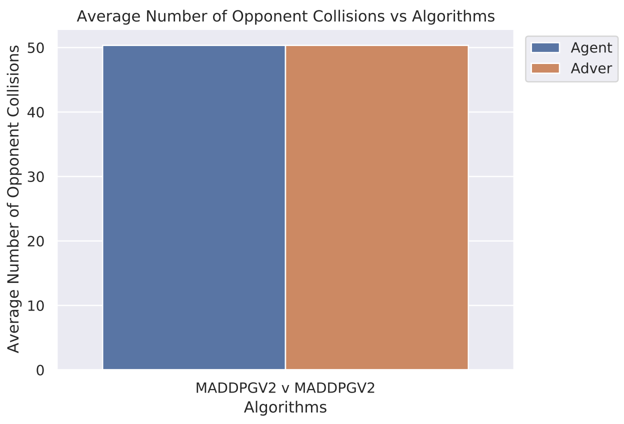
(A) 1 vs 1 (10 time-steps)

`images/1_vs_1_20_time_steps_average_number_of_opp_collisions_vs_algorithms_barplot.pdf`

(B) 1 vs 1 (20 time-steps)



(C) 2 vs 2



(D) 3 vs 3

Figure 3.4: Average number of collisions with opponent

The plots that highlights the average number of collisions per agent over 1000 episodes as shown in Figure 3.4 corroborates with the hypothesis mentioned above. It can be easily observed that the 3 vs 3 case using MADDPGv2 have the highest number of collisions with opponent of approximately 50 collisions per agent, with the 1 vs 1 case using MADDPGv2 (10 time-steps) trailing closely behind. Hence, having a significantly large frequency of collisions with the red drones is extremely valuable in helping blue drones learn. A potential reason for the 1 vs 1 case using MADDPGv2 not having better performance for the blue agents would be due to the fact that the 1 vs 1 case cannot leverage on the collaboration between the blue drones to defend the restricted zone. However, it is important to note that the red drones have a sizeable amount of screen exits as shown in Figure'3.3b. Hence, it is highly possible that the policy for the red drones is still far from optimality. Therefore, further training beyond 10000 episodes can be conducted in the future experiments to verify this.

3.2 Conclusion

In conclusion, dealing with sparse rewards remains an enormous challenge to any RL problem, especially when two multi-agent systems are training to learn to achieve two different objectives of significant difference in difficulty. A solution to the problem of sparse rewards would be to delve deep into reward engineering and craft specific learning curricula for the blue drones to assist them in their learning. While such a solution may produce the necessary improvements, trial and error with evaluations are highly tedious and time-consuming while requiring domain-specific knowledge. Implementing a sparse reward structure based on the target goal while having a framework generating the necessary incentives automatically to train the learning agent to converge to optimality is ideal. The project's best attempt towards that goal was the novel implementation of the MADDPGv2 algorithm that introduces a form of an implicit curriculum while providing guiding positive reward signals. While MADDPGv2 have shown relative success (for blue drones) in 3 vs 3 cases for 10 time-steps, it would be exceptionally challenging to generalise that for longer time-steps. Hence, given the limited success of the MADDPGv2 algorithm, there is much room for future work regarding this problem. Nevertheless, there have been many exciting work that seeks to tackle the issue of sparse rewards in RL [31,32]. Future work would do well to find inspiration from the work of others to be incorporated with the current existing work.

Bibliography

[1] J. Baek, M. Kang, and S. J. Hwang, “Accurate learning of graph representations with graph multiset pooling,” *arXiv preprint arXiv:2102.11533*, 2021.

[2] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. Pieter Abbeel, and W. Zaremba, “Hindsight experience replay,” *Advances in neural information processing systems*, vol. 30, 2017.

[3] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev *et al.*, “Grandmaster level in starcraft ii using multi-agent reinforcement learning,” *Nature*, vol. 575, no. 7782, pp. 350–354, 2019.

[4] C. Berner, G. Brockman, B. Chan, V. Cheung, P. Debiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse *et al.*, “Dota 2 with large scale deep reinforcement learning,” *arXiv preprint arXiv:1912.06680*, 2019.

[5] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, “The graph neural network model,” *IEEE transactions on neural networks*, vol. 20, no. 1, pp. 61–80, 2008.

[6] T. N. Kipf and M. Welling, “Semi-supervised classification with graph convolutional networks,” *arXiv preprint arXiv:1609.02907*, 2016.

[7] W. L. Hamilton, R. Ying, and J. Leskovec, “Inductive representation learning on large graphs,” in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 2017, pp. 1025–1035.

[8] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, “Graph attention networks,” *arXiv preprint arXiv:1710.10903*, 2017.

[9] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel, "Gated graph sequence neural networks," *arXiv preprint arXiv:1511.05493*, 2015.

[10] N. Peng, H. Poon, C. Quirk, K. Toutanova, and W.-t. Yih, "Cross-sentence n-ary relation extraction with graph lstms," *Transactions of the Association for Computational Linguistics*, vol. 5, pp. 101–115, 2017.

[11] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning.* PMLR, 2016, pp. 1928–1937.

[12] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[13] P. Sunehag, G. Lever, A. Gruslys, W. M. Czarnecki, V. Zambaldi, M. Jaderberg, M. Lanctot, N. Sonnerat, J. Z. Leibo, K. Tuyls *et al.*, "Value-decomposition networks for cooperative multi-agent learning," *arXiv preprint arXiv:1706.05296*, 2017.

[14] T. Rashid, M. Samvelyan, C. Schroeder, G. Farquhar, J. Foerster, and S. Whiteson, "Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning," in *International Conference on Machine Learning.* PMLR, 2018, pp. 4295–4304.

[15] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

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

[17] Y. Chao, A. VELU, E. VINITSKY *et al.*, "The surprising effectiveness of ppo in cooperative, multi-agent games," *arXiv preprint arXiv:2103.01955*, 2021.

[18] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[19] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *International conference on machine learning.* PMLR, 2014, pp. 387–395.

[20] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[21] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[22] I. Mordatch and P. Abbeel, "Emergence of grounded compositional language in multi-agent populations," *arXiv preprint arXiv:1703.04908*, 2017.

[23] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016.

[24] S. Brody, U. Alon, and E. Yahav, "How attentive are graph attention networks?" *arXiv preprint arXiv:2105.14491*, 2021.

[25] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[26] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *arXiv preprint arXiv:1607.06450*, 2016.

[27] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," *Advances in neural information processing systems*, vol. 12, 1999.

[28] T. Schaul, D. Horgan, K. Gregor, and D. Silver, "Universal value function approximators," in *International conference on machine learning.* PMLR, 2015, pp. 1312–1320.

[29] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," *arXiv preprint arXiv:1506.02438*, 2015.

[30] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International conference on machine learning.* PMLR, 2015, pp. 1889–1897.

[31] M. Jaderberg, V. Mnih, W. M. Czarnecki, T. Schaul, J. Z. Leibo, D. Silver, and K. Kavukcuoglu, "Reinforcement learning with unsupervised auxiliary tasks," *arXiv preprint arXiv:1611.05397*, 2016.

[32] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell, "Curiosity-driven exploration by self-supervised prediction," in *International conference on machine learning.* PMLR, 2017, pp. 2778–2787.

Appendix A

Experiment Details

A.1 Hyperparameters

The following sections highlights the hyperparameters used in the experiments. All relevant parameters with dimensions are in SI units.

A.1.1 MPE

The parameters for the environment are as follows: restricted radius of 0.2, intercept radius of 0.7, radar noise for position of 0.1, radar noise for velocity 0.5, reward constant of 10 and landmark size of 0.05. Both the blue and red drones have the same parameters as follows: drone radius of 0.25, drone size of 0.075, density of 25, mass of 1, maximum acceleration of 4, maximum speed of 1, noise for position actions (forces) of 1, noise for communication action of 1, range of position actions (forces) of 1.

A.1.2 MADDPG

The generic parameters of MADDPG are as follows: discount rate of 0.99, learning rate for actor and critic of 0.0005, memory size of 10000, batch size of 128. The architecture for blue and red drones are the identical. The architecture for the actor model consist of a neural network with 3 feed-forward layers of with 128 neurons each. For architecture of the critic, GATv2 has 3 layers of message passing with only 1 head that has 128 neurons per node in the graph. The GMT in the critic model uses GATv2 as the GNN for convolution, with hidden and output dimensions of 128 neurons. GMT follows a graph pooling ratio of 0.25 with 4 heads. The critic also has 2 neural networks consisting of 2 feed-forward layers with 64 neurons each for both the position (forces) and communication actions. The concatenated outputs then goes through a neural network consisting of 2 feed-forward layers of with 128 neurons each.

A.1.3 MADDPGv2

MADDPGv2 follows the same generic parameters and architecture as MADDPG. MADDPG follows four goal sampling strategy: "episode", "future", "goal distribution version 1", "goal distribution version 2". For the "episode" goal sampling strategy, the goals are sampled from the replay buffer with k random goals coming from the same episode as the transition being replayed. For the "future" goal sampling strategy, the goals are sampled from the replay buffer with k random goals coming from the same episode as the transition being replayed and were observed after it. For the "goal distribution version 1" goal sampling strategy, k goals are sampled randomly from specific goal distribution. For the "goal distribution version 2" goal sampling strategy, k goals are sampled from specific goal distribution with a win-loss adjusted probability distribution that samples harder goals from the current if the agent if is performing well on the current goal and vice-versa. For MADDPGv2, the goal sampling strategy of "goal distribution version 2" is used for both blue and red drones with $k = 4$.

A.1.4 MAPPO

MAPPO follows the same generic paramters and architecture as MADDPG. The following are the specific parameters for MAPPO: policy clip coefficient is 0.2, batch size of 20, number of epochs of 5 and $\lambda = 0.99$ for GAE.