Karlsruhe Institute of Technology

Swarm Reinforcement Learning with Graph Neural Networks

Bachelor's Thesis
of

Christian Burmeister

KIT Department of Informatics
Institute for Anthropomatics and Robotics (IAR)
Autonomous Learning Robots (ALR)

Referees: Prof. Dr. Techn. Gerhard Neumann
Prof. Dr. Ing. Tamim Asfour

Advisor: Niklas Freymuth

Duration: Juli 17st, 2021 — January 17st, 2022

Erklärung

Ich versichere hiermit, dass ich die Arbeit selbstständig verfasst habe, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, die wörtlich oder inhaltlich übernommenen Stellen als solche kenntlich gemacht habe und die Satzung des Karlsruher Instituts für Technologie zur Sicherung guter wissenschaftlicher Praxis beachtet habe.

Karlsruhe, den 17. January 2022

Christian Burmeister

Zusammenfassung

Das stehts wachsende Forschungsgebiet des Multi-Agent Reinforcement Learning (MARL) betrachtet Echtweltproblem in welchen mehrere Aktoren, genannt Agenten, entweder in kooperativer, kompetitiver oder in einer gemischten Art und Weise miteinander arbeiten müssen. Dies findet Anwendung bei der Beschreibung des Verhaltens von Ameisen, Bienen oder Vögeln. Ebenfalls findet dieser Ansatz für Suche und Rettung, sowie der Modellierung von Netzwerksystemen Verwendung. Hierfür müssen die Agenten etwaige große Menge an Beobachtungen über ihre Umwelt verarbeiten und lernen, um in der Lage zu sein die Aufgabe zu lösen. Somit ist eine effiziente Datenstruktur für die Observationen der Agenten unumgänglich, um auch große Mengen von Agenten lernen lassen zu können. Graph Neural Networks (GNN) sind Lernprozesse die entworfen wurden um auf als Graph konstrurierten Daten effektiv zu lernen. Jene eignen sich gut um die Kommunikation zwischen den Agenten zu repräsentieren. In dieser Arbeit möchten wir die Auswirkung und den Nutzen von GNNs auf das lernen von MARL Problemen untersuchen. Der Fokus liegt hierbei auf den Effekt mehrerer Durchläufe von GNN auf die Informationsübertragung in Umgebungen mit sehr geringer partieller Sichtbarkeit.

In dieser Arbeit stellen wir eine Trainingsarchitektur vor, welche den Proximal Policy Optimization (PPO) Algorithmus verwendet. Zusätzlich besitzt sie ein GNN, welcher problemlos auf beliebig viele Durchläufe skalieren kann. Dieser Ansatz basiert auf der bereits veröffentlichen Arbeit für eine Multi-Agent Deep Reinforcement Leaerning Architektur, welche man in Ruede et al. (2021) findet. Zudem wurde die Architektur um die Fähigkeit erweitert, mit heterogenen Graphen arbeiten zu können. Jene wird nämlich für unser Observationsmodell in komplexeren Aufgaben benötigt. Anschließend wird in mehreren Multi-Agent Aufgaben der Nutzen von mehreren GNN Durchläufen beurteilt.

Unsere Ergebnisse zeigen, dass ... (EINFÜGEN!)

Abstract

Multi-Agent Reinforcement Learning (MARL) is an ever expanding field of research which deals with real world problems that require multiple entities, called agents, to work cooperatively, competitively, or as a mixture of both. Practical applications range from the simulation of ants, bees and birds, coordinating network systems aswell as search-and-rescue operations. The agents need to learn to correctly use potentially large amounts of observation data to make informed decisions. Therefore an efficient way to process observation data is needed for effective learning that can scale to a large number of agents. Graph Neural Networks (GNN) allow for learning processes to work on graphs. Currently, they are a growing field of research. GNNs are a natural representation for modelling inter-agent communications. We want to investigate the effect of GNNs on learning and the resulting policy on MARL tasks. We will focus on the effect of multiple GNN passes to information propagation, especially in partial observability with a short range.

This work will introduce an architecture that uses Proximal Policy Optimization (PPO) for training with a graph base that allows scaling to multiple GNN passes. Our approach is based on the previosly proposed multi-agent deep reinforcement learning architecture found in Ruede et al. (2021). Furthermore we expand the architecture to work on heterogeneous graphs, which is required for our observation model of more complex tasks. We then evaluate this architecture in multiple multi-agent tasks to show the benefit of multiple GNN passes.

Our results show that ... Results/Conclusion

- Summarize main result (most important): improvements on policy fitness for multiple hops. little culling => small effect. tight culling => larger effect. Effect larger for more complex tasks and in heterogeneous. evaluate compared to robin?
- Main Conclusions: GNN good fit for describing inter agent communication and cooperation. MARL can greatly benefit from multiple hops.

Table of Contents

Chapter 1.

Introduction

Multi-Agent Systems has several real-world applications. Any System that requires multiple agents to work together or against each other can be modeled in a Multi-Agent fashion. For example, in Chu et al. (2020) the authors define a traffic control system in which they create a grid of traffic lights, where each of them is an agent. They then learn the agents to optimize minimal sum of queue and waiting times at the intersections. Similarly, robot swarms can also be used to solve pathfinding problems in indoor, maze-like scenarios as discussed in Aurangzeb et al. (2013). Given the structure of the environment communication is severly limited.

Recent applications and research in MARL and GNN.

- GNN really hot right now! Alot of research in this topic.
- MARL: Robin Paper (this thesis is based on his!).
- Multi-Agent RL: RTS Games: Zhou et al. (2021).
- GNN: Graph Convolutional reinforcement learning.

In Multi-Agent Reinforcement Learning we have multiple agents that work together or against in an environment. We use adapted versions of the Reinforcement Learning techniques used for single agents. Graph Neural Networks are learning methods for Neural Networks. The function they learn are designed in a way to preserve Graph structure. They take graphs as Inputs and have graphs as outputs. In our ...???

What is GNN, What is MARL, what can GNNs it do for MARL? (the main thing we want to talk about, more conceptually)

- MARL: Multiple Agents are trying to learn in an environment using RL Methods. Adapting RL algorithms designed for single agent environments for MAS.
- GNN:
- What can GNN do for MARL:

What is my approach I want to talk about here? What was our goal?

- Approach: Based on Robin Paper => Expand on his with GNN! (this is a bachelor thesis) Ruede et al. (2021)
- restriction on MARL, restriction about agents, environment etc. (Robin, 2.4)
- Goal: Show that MARL especially under harsh communication ranges can benefit alot from GNN structures with multiple hops.

My work relative to other work. What has other research focused on?

- Robin Paper => Expand on his with GNN! (this is a bachelor thesis) Ruede et al. (2021)
- And Graph Convolutional reinforcement learning paper.

This thesis is structured like this: Firstly we will go over the necessary preliminaries and notation needed to understand this thesis in Chapter 2. This includes Reinforcement Learning basics in the single-agent case, then adresses how it is applied in multi-agent tasks and is followed by a brief overview of Graph Neural Networks. In Chapter 3 related work will be discussed. The subsequent Chapter 4 is focused on the details of our proposed model. It will describe our policy architecture respect to both homogeneous and heterogeneous Graph Neural Networks. Our experiments, including the general setup and the specific tasks that we trained our mode on, are described in Chapter 5. The goal of our expermiments, their interpretation and evaluation of our results can be found in Chapter 6. We will complete this thesis by distilling the conclusions about our findings and provide future avenues for improvements in Chapter 7.

Chapter 2.

Preliminaries

2.1. Notation

The following table contains common used notation through the thesis. It features mathematical, Reinforcement Learning (RL) and Graph Neural Network (GNN) notation.

Table 2.1.: Overview of mathematical, RL and GNN notation

Symbol	Description
$\oplus$ or $\otimes$	Aggregation
a	Agent
act	Action
π	Policy
r	Reward
s	State
o	Observation
x_v	Node-features
x_e	Edge-features
X_g	Global-features
$\mathcal{N}_i$	Neighborhood for agent i

2.2. Reinforcement Learning

In Reinforcement Learning (RL), an agent interacts with an environment to iteratively learn about it's task. Every timestep it uses an action a and the environment generates a reward r, which is used to update the strategy of the agent, called policy π. The policy π can be deterministic $\pi(s) = a$, or probabilistic $\pi(a|s) = \mathbb{P}[A_t = a|S_t = s]$. Rewards can be dense (outputed at avery step), or sparse (outputed irregularly). It also generates an observation o. The observation is based on the entire information of the environment state S^e (fully observable) or part of the environment state S^e (partially observable). In both cases it is a potentially noisy mapping of the state. It may still be different from the state, even if fully observable. This is used by π to generate a new action for the next time-step.

Single-Agent Reinforcement Learning in fully observable environments are formally described using a Markov Decision Process (MDP). An MDP relies on the Markov property. It defines that the current state is enough to infer future states through a transition probablitity. So the history of previous states is not needed. Therefore it can be described by the 4-tuple (S, A, T, R), where:

- S is a set of states called the state space.
- A is a set of actions called action space.
- $T(s, a, s')$ is the state transition function. For a given action a and state s it outputs the probability of landing in state s'. It's probability is denoted with $\mathbb{P}(S'|S, a)$.
- $R(s, a, s')$ is the expected immediate reward function. It returns the reward for transitioning from state s to state s'. It may also include the action.

If the environment is partially observable an MDP can be expanded to a partially observable MDP (POMDP). The policy now takes the observation as input and not the actual environment state. The observation is based on given agent and includes subjective partial information. We can formaly describe a POMDP as a tuple (S, A, O, P, R, Z), where we have in addition to MDPs:

- O a finite set of observations. An observation may be any potentially noisy mapping of the state. As such, it may contain incomplete or subjective partial information.
- $Z^a_{s'o}$ an observation function which gives us the probability of observation o from the agent's perspective for a new environment state s'.

To evaluate a policy we are able to use two functions, the state-value function and action-value function. The state-value function $V_\pi(s)$ tells us the expected future reward of a given start state s and a policy π. This is the expected sum of the rewards we will get following the policy starting from s. The sum is further discounted by the discount rate $\gamma \in [0, 1]$. It is used to discount rewards that are futher in the future, any may be interpreted as the probability of a rollout continuing after any given step. The value function may be described as

$$V_\pi(s) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r_t | S_t = s \right]$$

On the other hand we have the action-value function $Q_\pi(s,a)$, also known as Q-function. It tells us the discounted expected future reward for a policy π, an state s and an action a. This describes how good it is to be in a given state and to use a given action, so this can be formulated as

$$Q_\pi(s,a) = \mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t r_t | S_t = s, A_t = a]$$

So an optimal policy π^* maximizes the value-function and the action-value-function. For any nontrivial task like continuous environments that have a lot of states, a direct representation of π is not practical. The action-state space explodes in size and therefore an approximation is required.

In this thesis, we consider a parameterized policy pi, whose parameters are optimized using policy gradient methods. The optimal policy is denoted as π^*. In Actor-Critic methods we optimize two participants. The actor which controls how the agent behaves is based on the policy π. The critic measures how good the current policy is by using the value-function. While learning, the actor tries to improve the policy by using the advice from the critic, which learns how to correctly critique the policy of the actor.

In order to reduce variance when training we can use the advantage function $A^\pi(s,a)$. It describes how much extra reward the agent could get by taking a given action.

$$A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$$

This is called Advantage Actor Critic (A2C). Common Actor-Critic algorithms include Proximal Policy Optimization (PPO) (Schulman et al., 2017), Trust Region Policy Optimization (TRPO) (Schulman et al., 2015), Trust Region Layers (PG-TRL) (Otto et al., 2021).

2.3. Multi-Agent Reinforcement Learning

In this thesis we explore Multi-Agent Reinforcement Learning (MARL) or Swarm Reinforcement Learning (SwarmRL). In a Multi-Agent System (MAS) n agents influence the environment. The set of actions for all agents at a given timestep is called the joint actions $A = \langle a_1, ..., a_n \rangle$, where a_i is the action of agent i. Each agent has it's own observation and we therefore have a joint observation $O = \langle o_1, ..., o_n \rangle$. There are a few extensions to POMDPs

for multi-agent environments, for example decentralized POMDPs (Dec-POMDP). The main difference to POMDPs can be seen in the joint observations and joint actions.

- I is the set of n agents.
- S is a set of states.
- A is a set of joint actions.
- $T(s,a,s')$ is the state transition function. For a given joint action a and state s it gives the probability of landing in state s'.
- O is a set of joint observations.
- $Z^a_{s'o}$ is an observation function which gives us the probability of an observation o from an agent a's perspective for a new environment state s'.
- $R(s,a,s')$ is the expected immediate reward function. It returns the reward for transitioning from state s to state s'. It may also include the action.

2.4. Neural Networks

The Neural Network is the basis for every kind of approximate function learning. Each neuron can hold a real value. Layers are mulitple neurons in a row, where neurons in a given layer are used as inputs for the neurons of the next layer. The new value y is a weighted sum, with weights w, of the neurons of the previous layer it is connected to plus a bias b. This is describes a linear function. The result is then put through an activation function ϕ to control value ranges.

$$y = \phi(\mathbf{W}^T\mathbf{x} + \mathbf{b})$$

If each of the neurons of the previous layer is connected to each node of the next layer we call that fully connected. The first layer is the input of the function and the last layer is the output. A Multi-Layer-Perceptron has multiple fully connected layers, where layers i receives input from layer $i-1$ and in turn produces the input for layer $i+1$.

2.5. Message Passing GNN

In this thesis a graph is given as $G = (V,E,X_v,X_e,X_u)$. We have n Nodes $|V| = n$ with a node feature Matrix $X_v \in \mathbb{R}^{n\times k}$ that contains k features for each node. They describe attributes of a given node for a given task. Edges are defined as $v_i, v_j = e \in E \subseteq V \times V$, $|E| = l$ with their own edge feature Matrix $X_e \in \mathbb{R}^{l\times m}$. We additionally use global features $X_u \in \mathbb{R}^{q\times r}$ that are used to define features for the entire graph. The neighborhood of a node v_i is defined as every node v_j, where $i \neq j$, which is connected to v_i via an edge. Which is denoted as $\mathcal{N}_i = \{v_j \mid (v_i, v_j) \in E\}$.

In Graph Neural Networks (GNNs), we want to learn functions that run on graphs as inputs. The function should have the same result for two isomorphic graphs. Edges are represented using an adjacency matrix A, where

$$a_{ij} = \begin{cases} 1 & (i,j) \in E \\ 0 & otherwise \end{cases}$$

Nodes and edges have features, called node-features X_v and edge-features X_e. To preserve isomorphic graphs, we want the function f of our neural network to be permuation equivariant, i.e.

$$f(PX_v, PAP^T) = Pf(X_v, A) \text{ for a given permuation} P$$

In order to do that we define a function $g(x_{v_i}, \mathcal{N}_i)$, that uses a node-feature and the neighborhood $\mathcal{N}_i$ of a node v_i. It is called the local aggregation function. Function g should not depend of the order of nodes in the neighborhood, in other words it should be permutation invariant $g(x_{v_i}, P\mathcal{N}_i) = g(x_{v_i}, \mathcal{N}_i)$. We can then construct f using g so that f achieves permutation equivariance in the following way

$$f(X_v, A) = \begin{pmatrix} g(x_{v_1}, \mathcal{N}_1) \\ g(x_{v_2}, \mathcal{N}_2) \\ \vdots \\ g(x_{v_n}, \mathcal{N}_n) \end{pmatrix}$$

f is called a single Graph Neural Network block or a single GNN hop. In practice f is learned by using an MLP Neural Network. By applying this function multiple times to the graph, information can flow between multiple neighborhoods, called multiple GNN hops. A node is able to get information about the neighbors of his neighbor. The local aggregation function g can have one of three flavors.

$$\begin{aligned} \text{Convolutional: } g(x_i) &= \phi(x_{v_i}, \oplus_{j \in \mathcal{N}_i} c_{ij}\ \psi(x_{v_j})). \\ \text{Attentional: } g(x_i) &= \phi(x_{v_i}, \oplus_{j \in \mathcal{N}_i} a(x_i, x_j)\ \psi(x_{v_j})). \\ \text{Message-Passing: } g(x_i) &= \phi(x_{v_i}, \oplus_{j \in \mathcal{N}_i} \psi(x_{v_i}, x_{v_j})). \end{aligned}$$

In Convolutional GNNs we use constants c_{ij} for each edge which states how much Node j influences the features of Node i. This turns the neighborhood aggregation into a weighted sum. So it is only dependend on the structure of the graph. Attentional GNNs are similar, but the weights are now a function $a(x_i, x_j)$ of Node i and Node j that is learnable. In Message-passing GNNs we have no weights, but the neighbor feature transformation function ψ now

takes both node features into account $\psi(x_{v_i}, x_{v_j})$. This way the sender and receiver work together to create messages that are used across an edge.

Chapter 3.

Related Work

20 referenced papers. 2-3 sections

- RL
  - Swarm RL (max, Robin)
  - PPO Actor-Critic
  - TRL
  - homogeneous vs heterogeneous agents.
- GNN
  - GNNs
  - GraphNets: Learning to Simulate Complex Physics with Graph Networks.
  - GCNs
  - GATs
  - MeshGraphNets
  - heterogeneous GNNs (Pytorch Geometric) and their papers.

Chapter 4.

Swarm Reinforcement Learning with Graph Neural Networks

First we will introduce the policy architecture used as the base. It is designed to work with PPO and uses the latent node features of the observation for the actor and critic. This base is then expanded with a Graph Neural Network structure that allows multiple hops through a stack of message passing blocks. To support heterogeneous graphs, needed for tasks like pursuit, the architecture is extended to work on multiple graphs as inputs. Most of the changes needed for heterogeneous graphs is located in the edge-, node- and global-modules respectively.

4.1. PPO - Policy Architecture

An overview of the policy architecture as desribed below can be seen in Figure 4.1.

Our environments can have different observations structured as a graph $G = (X, E, U)$. For now we will talk about homogeneous graphs. Each agent a_k uses information it knows about itself and what it knows about it's neighboring agents $a_i \in \mathcal{N}_k$. The neighborhood is determined either by the culling method kNN or using the euclidean distance. The specific data for each environment is described in Chapter 5. Information about a_k is used as the node-features x_k for the graph. The edges E describe neighborhood relations for a given $\mathcal{N}_k$. For an edge $\{x_i, x_j\} = e \in E$ edge-features on the other hand define the individual observations from a_i about the state of the agent a_j. Ruede et al. (2021) uses a similar observation structure implicitly. Edge-features are created using an observe() function. They explicitly calculate the neighborhood for each agent that uses the policy.

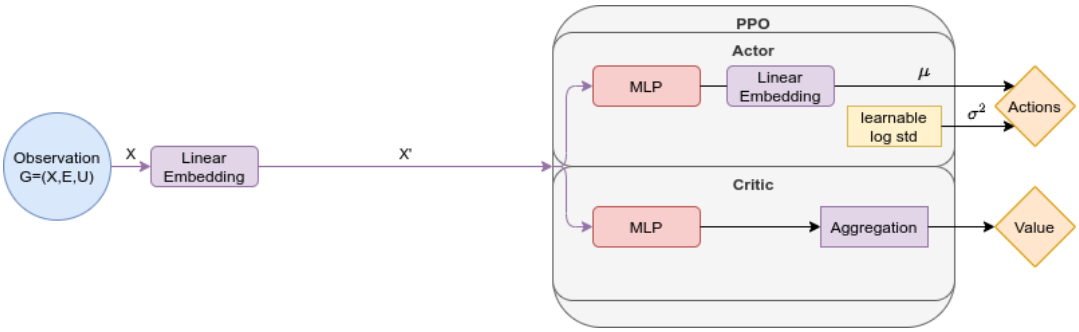


Figure 4.1.: Policy architecture used for Swarm Reinforcement Learning with Graph Neural Networks. The node features of the observation graph is the only data used. Here the actor and critic use the same embedding. It is put into a latent dimension and then used by the PPO Network to calculate the action as a distribution as well as the value.

For testing and debugging purposes, the Graph Neural Network (GNN) can be disabled. When it is, only the node-features are used. Once the observation graph is constructed, the node-features are encoded into a latent-space representation. For this embedding we use a linear layer. We allow for independet embedding for the actor and critic. They can either share the same embedding or have different embeddings. The Multi-Layer Perceptron (MLP) used in Ruede et al. (2021) is used as an encoding step and to process each element of an aggregation group. Processing is not needed for our embedding, as our GNN is responsible for that, so a simple linear transformation is enough.

The learning head for Proximal Policy Optimization (PPO) is composed of the actor describing the policy π, which gets us the joint action act $= \langle \text{act}_1, ..., \text{act}_n \rangle$ and the critic calculating the running advantage A. Our actor uses an MLP, followed by another linear transformation. It will reduce the dimension from the latent dimension back to the action dimension. This is then used as the mean μ for a Diagonal Gaussian distribution to support non deterministic policies. The variance σ^2 uses a learnable logarithmic standard deviation. The critic also uses an MLP with an output-dimension of 1. This defines a value per node in the graph. In Ruede et al. (2021) the authors define a value for each agent. This helps to identify which agent was responsible for the reward. But this is not how the PPO Algorithm was originally designed to work for single-agent environments. Our architecture can use an aggregation to output a single value per graph. This can either be $\min(x)$, $\max(x)$, or $\text{mean}(x)$. Through exploration the agents are still able to identify over time how much an agent was responsible for a given reward. We parameterized node-wise values and graph-wise values for our approach.

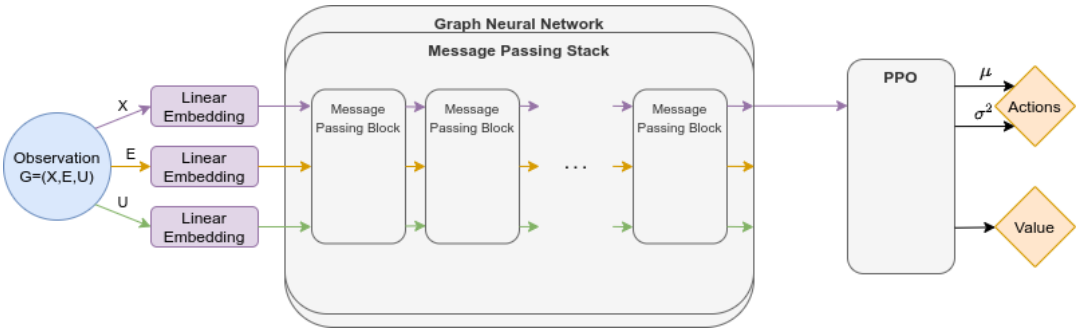


Figure 4.2.: Here a GNN is added as a step before the policy architecture. The entire observation is put into a latent space and then passed through multiple message passing hops. The outputs of one hop is used as the input of the next hop. This is coordinated through the message passing stack.

4.2. Homogeneous Message-passing GNN Architecture

An overview of the message passing architecture for homogeneous graphs as described below can be seen in Figure 4.2.

When enabling the GNN, it is added as a step between the policy architecture and observation. node-features, edge-features and global-features are now embedded into a latent-space linearly, similar to above. The weights between the encoders are not shared. Global-features can be used to explicitly give the swarm global information. In our experiments this was not needed.

We chose to use a GNN structure that uses stacks of GNN blocks. Each block defines a single message-passing hop. The stack is an array of blocks that are used sequentially. This way we are able to support multiple GNN hops. We do want to note, that Ruede et al. (2021) model architecture for a single aggregation group can be described via the function:

$$f(a_i) = \text{decoder}(\text{selfObserve}(a_i), \bigoplus_{j \in \mathcal{N}_i} \text{encoder}(\text{observe}(a_i, a_j)))$$

As can be seen, this already implements a common message-passing GNN. In comparison, our architecture adds the ability to process multiple GNN hops. The GNN block is composed of an edge-, node-, and global-module, each responsible for the corresponding features. A diagram of the relationship between these blocks is outlined in Figure 4.3. The functions used by the modules are as following:

$$\text{Edge-Module: } x_{e'} = f_e(x_v, x_u, x_e, x_g)$$
$$\text{Node-Module: } x_{v'} = f_v(x_v, \oplus_{\{e'=(v,u)||_v\}} x_{e'}, x_g)$$
$$\text{Global-Module: } x_{g'} = f_g(\oplus_V x_{v'}, \oplus_E x_{e'}, x_g)$$

Similar to the embedding, our architecture allows for two GNN modules for the value and critc. Each of them have seperate embedding and a seperate GNN module. This allows both to process observations differently. Each of the aggregation functions used in the entire network are shared and can be set to $\min(x)$, $\max(x)$, or $\text{mean}(x)$. Furthermore the input of a GNN block can be added to the output. These residual connections help with the vanishing/exploding gradients problems by breaking up the multiplications.

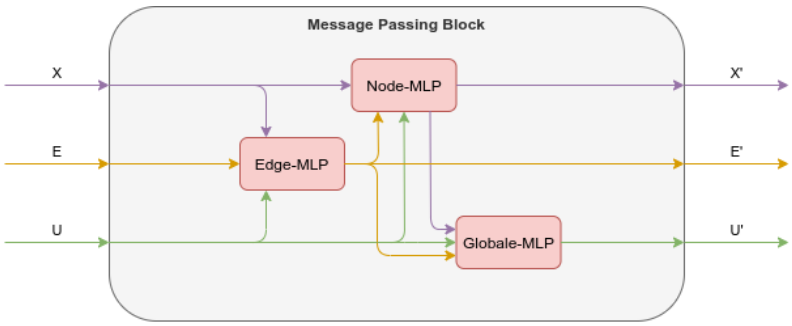


Figure 4.3.: This is a detailed view of one message passing block. It uses a edge-, node- and global-module to compute a single message passing hop.

4.3. Heterogeneous Message-passing GNN Architecture

An overview of the message passing architecture for heterogeneous graphs as described below can be seen in Figure 4.4.

This extension is used when an environment needs multiple distinct information types. By using heterogeneous graphs we are able to support multiple aggregation groups as in Ruede et al. (2021). An heterogeneous graph $G = (X, E, U)$ is composed of multiple node-types X_i, $i = 1, ..., m$ where $X = \bigcup_{i=1}^{m} X_i$ and multiple edge-types E_i, $i = 1, ..., m^2$ where $E = \bigcup_{i=1}^{m^2} E_i$. An edge-type is uniquely identified by its source node-type and destination node-type where $\forall E_i : \exists j, k \leq m : E_i = (X_j \times X_k)$. If the source- and destination-node-types are not the same i.e. $j \neq k$, we can either define the edge-types as directed, or make it undirected. In the undirected case not all node-type combinations are edge-types, because $(X_j \times X_k) = (X_k \times X_j)$. Here the edge-type of $(X_j \times X_k)$ is sufficient. in the directed case $(X_j \times X_k) \neq (X_k \times X_j)$ and both types are needed. For experiments, both variations are parameterized in our architecture. If $j = k$ the edge-type can be directed or undirected, which depends on the culling method used.

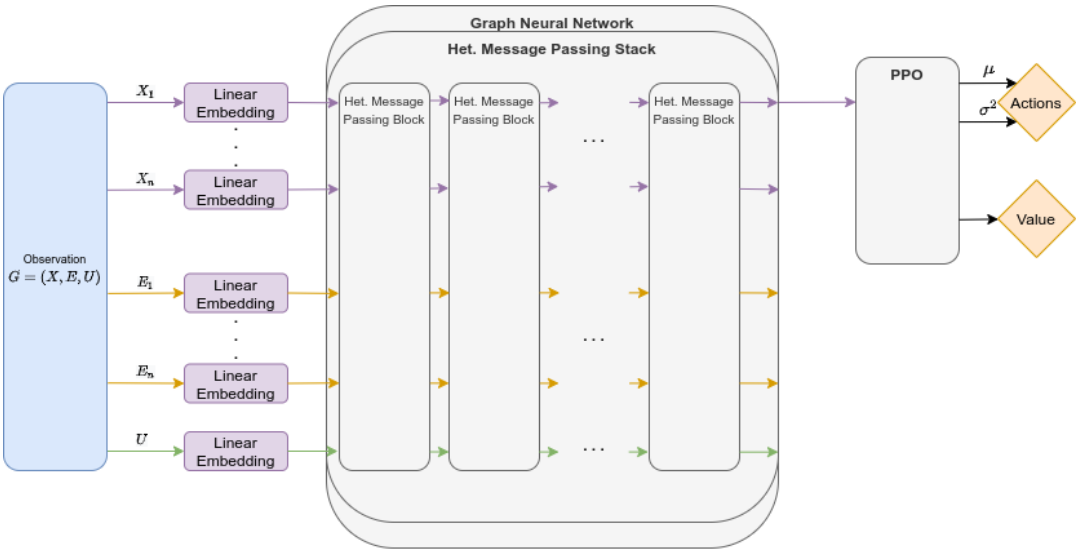


Figure 4.4.: An overview of the heterogeneous version of the architecture. The overall structure stays the same, but the input observation is composed of mulitple node-types and edge-types. Each of the input embeddings per graph is unique. The policy architecture only uses one of the note types as input.

Each of the node-types and edge-types are linearly embedded. Each type has it's own linear transformation. As different types define completely aggregation groups, weight sharing is not used. Global features are also embedded independently. Structurely the heterogeneous GNN stacks and blocks are the same, they only have more inputs. The underlying heterogeneous edge-, node- and global-modules have the following functions:

$$\text{Edge-Module: } x'_{e_i} = f_{e_i}(x_v, x_u, x_{e_i}, x_g), e_i = (u, v)$$
$$\text{Node-Module: } x'_{v'_i} = f_{v_i}(x_{v_i}, [\otimes_j \oplus_{\{e_j=(v_i,u)\}} x'_{e_j}], x_g)$$
$$\text{Global-Module: } x'_g = f_g([\otimes_j \oplus_{V_j} x'_{v\in V_j}], [\otimes_k \oplus_{E_k} x'_{e\in E_k}], x_g)$$

In addition to the options outlined in the homogeneous GNN section, we are able to define different functions for $\otimes$ and $\oplus$. Aggregator $\oplus$ follows the global aggregation function setting shared by the network, while $\otimes$ on the other hand can be defined as a concatenation or as the global aggregation function. Ruede et al. (2021) only uses a concatenation. The concatenation is more expensive as the input dimension of the MLP of the node-, or global-module is larger. Though it is more expressive than the alternative of using aggregation. In tasks like multi evader pursuit, if the different node-types are aggregated, it is harder for agents to distinguish stimuli from other agents or from the evaders.

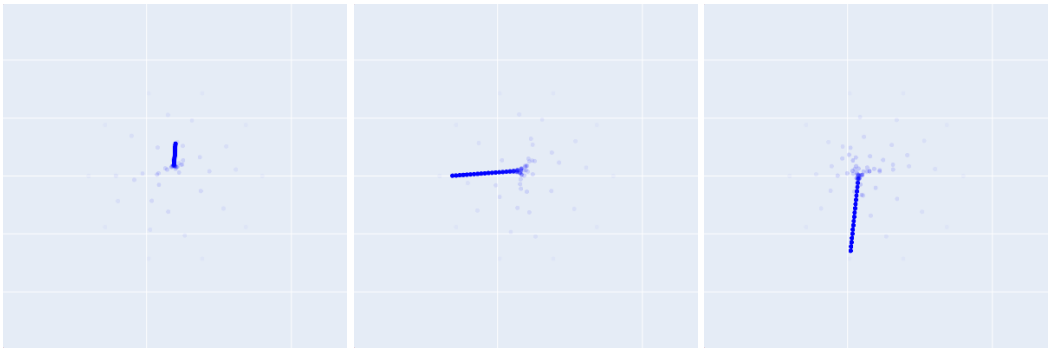
Chapter 5.

Experiments

5.1. General Setup

The following experiments were written to work with the DAVIS project Freymuth (2021). DAVIS already included the core structure for Multi-Agent Reinforcement Learning research with support for graph observations. Furthermore it simplified the recording of key metrics and training with a cluster. Training was done via the BWUniCluster2 by the state of Baden-Württemberg through bwHPC. We adapted the DAVIS project to work with heterogeneous observation graphs and implemented the tasks needed for our experiments.

All the experiments use, if not otherwise stated, Proximal Policy Optimization (PPO) Schulman et al. (2017) with additional code level optimizations from Engstrom et al. (2020). Additionaly we implemented the ability for the actor and critic to use different Graph Neural Networks or use the same. In the former case the actor and critic are able to learn different graph networks to suit their needs. If not stated otherwise, actor and critic use different GNNs.



(a) (b) (c)

Figure 5.1.: Three example episodes for Rendezvous. The circular start position used for evaluation can be seen. The agents are able to meed in a couple of steps. Then they move a little as a group.

5.2. Tasks

5.2.1. Rendezvous

The goal of the Rendezvous task, is for n agents to converge onto a single point. An example episode can be seen in Figure 5.1.

The environment can be configured as a torus (position is wrapped using modulo) or as an rectangular world with borders (position is clipped). Positions are using floating-point precision. It terminates after a given amount of timesteps. The agents are dots without collisions. They use a direct dynamic model, therefore the two actions they can perform represent movement in the x-Axis and y-Axis respectively. The reward function r consists of two terms. First we use the mean of the normalized pairwise distances between the agents as a distance penalty d_p. Secondly we use an velocity penalty v_p, that scales squared to the mean of the velocity v:

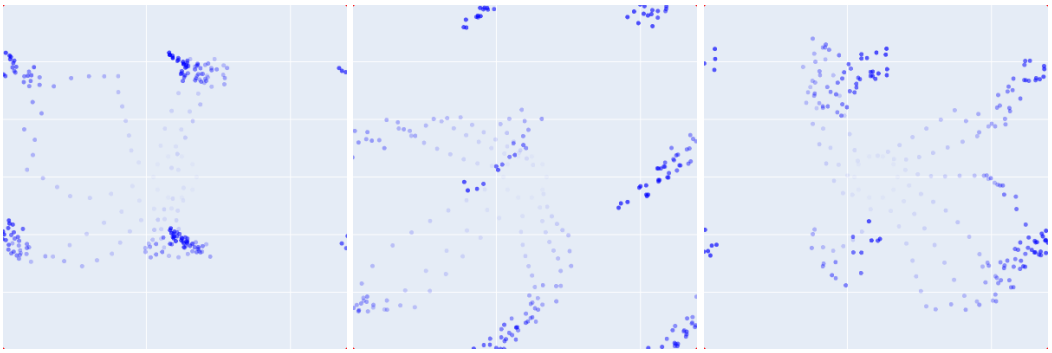
$$
\begin{aligned}
v_p &= \text{mean}(v^2) \\
d_p &= \text{mean}(\frac{\text{pairwise-distances}}{\text{worldsize}}) \\
r &= v_p + d_p
\end{aligned}
$$

A culling method is used, so that the agents have a finite sensor range, either kNN or euclidean distance. The observation graph is homogeneous and composed of the following aspects:

- node features:
  1. normalized agent positions (optional)

- edge features:
  1. normalized pairwise agent distances
- global features: None

5.2.2. Dispersion



(a) (b) (c)

Figure 5.2.: Three example episodes for Dispersion. The circular start position used for evaluation can be seen. The agents try to disperse and organize into four distinct groups. Each example used $f(x) = x$, so they just calculated the mean.

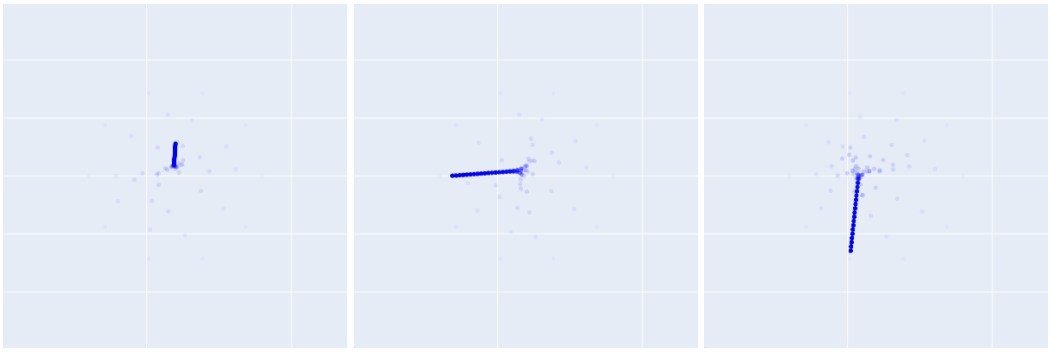
In a Dispersion task, the agents should try to maximize the distance between each other. An example episode can be seen in Figure 5.2.

This environment is a variation of Rendezvous and therefore shares most of it's properties. The main difference lies within the reward function r. Our implementation allows for different reward calculations using functions $f(x)$ on the reward, before calculating the mean. Supported functions are x, x^2, $\sqrt{x}$, $\min(x)$, $\max(x)$

$$
\begin{aligned}
a_p &= \text{mean}(a^2) \\
d_p &= \text{mean}(\frac{f(\text{pairwise-distances})}{\text{worldsize}}) \\
r &= a_p + d_p
\end{aligned}
$$

5.2.3. Single Evader Pursuit

For Single Evader Pursuit the agents have to catch a single evader. Given that the evader has a higher velocity than the agents, they have to cooperate. An example episode can be seen in Figure 5.3.



(a) 10 Agents (b) 15 Agents (c) 20 Agents

Figure 5.3.: An example for a successfull Single Evader Pursuit episode

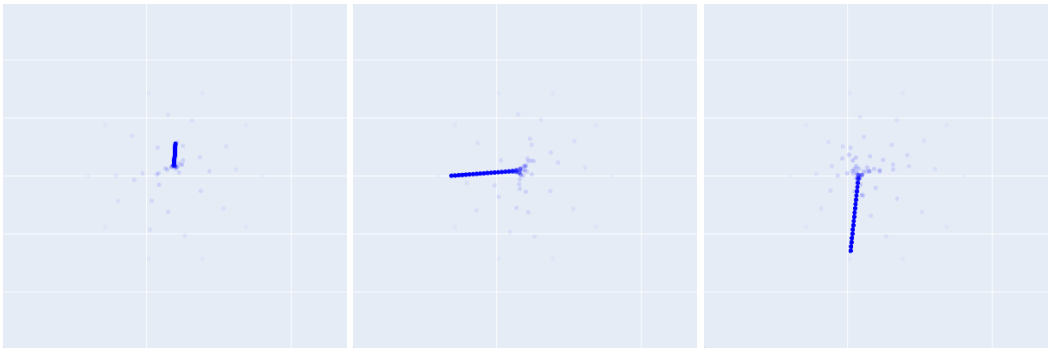
The environment can be configured as a torus (position is wrapped using modulo) or as an rectangular world with borders (position is clipped). Positions are using floating-point precision. It terminates after a given amount of timesteps or when the single evader has been caught. A catch is triggered when one agent is less than 1% of world size away from the evader. The agents are collisionless dots. Like Rendezvous a direct dynamic model is used. The evader is part of the environment and will not learn. It's dynamic is either a simple linear movement or uses Voronoi-regions which is based on Zhou et al. (2016). The minimum normalized distance of the agents to the evader is used for the distance penalty d_p and the velocity penalty v_p is the same as Rendezvous:

$$
\begin{aligned}
v_p &= \text{mean}(v^2) \\
d_p &= \text{mean}(\frac{\text{agent-evader-distances}}{\text{worldsize}}) \\
r &= v_p + d_p
\end{aligned}
$$

Single Evader Pursuit also supports the same culling methods as Rendezvous. The observation graph is heterogeneous consists of the following aspects:

- agent node features:
  1. normalized agent positions (optional)
- evader node features:
  1. normalized evader positions (optional)
- agent-to-agent edge features:
  1. normalized pairwise agent distances
- agent-to-evader edge features:
  1. normalized agent to evader distances
- global features: None

5.2.4. Multi Evader Pursuit



(a) 10 Agents (b) 15 Agents (c) 20 Agents

Figure 5.4.: An example for a successfull Multi Evader Pursuit episode

In Multi Evader Pursuit the agents have to catch multiple evaders. An example episode can be seen in Figure 5.4.

This task is largely the same as single evader pursuit. The catch threshhold is changed to less than 2% of world size. The reward uses the established velocity penalty v_p and a catch count c. It will reward the agents with +1 every time they catch an evader:

$$v_p = \text{mean}(v^2)$$
$$r = v_p + c$$

Chapter 6.

Evaluation

This section elaborates on the results of our experiments. As a proof of concept we aimed to show, that our architecture is able to learn rendezvous tasks with a more or less GNN Blocks (1). Then we compared training on a set of agents, while evaluating on a different amount of agents (2). Afterwards we showed the effect of different aggregation functions on training (3), and how multiple GNN hops can effect the result if the observation radius for the agent varies (4).

Evalution of the results used different environments compared to training. While the agents trained on randomized starting positions, they were tested with fixed starting positions to allow comparable results. A similar setting was used for evader starting position in Pursuit environments. We considered the average reward per training step and the last reward per training step as performance metrics. The mean gave us a meaningfull metric for the progress of the RL algorithm, while the last reward had a more geometric importance for solving the tasks like Rendezvous.

Unless mentioned otherwise, the following experimental settings were used: Experiment settings that are generally true (overview, add Appendix for more details) go through more of the settings for the environments used that are important, like agents have same starting position etc.

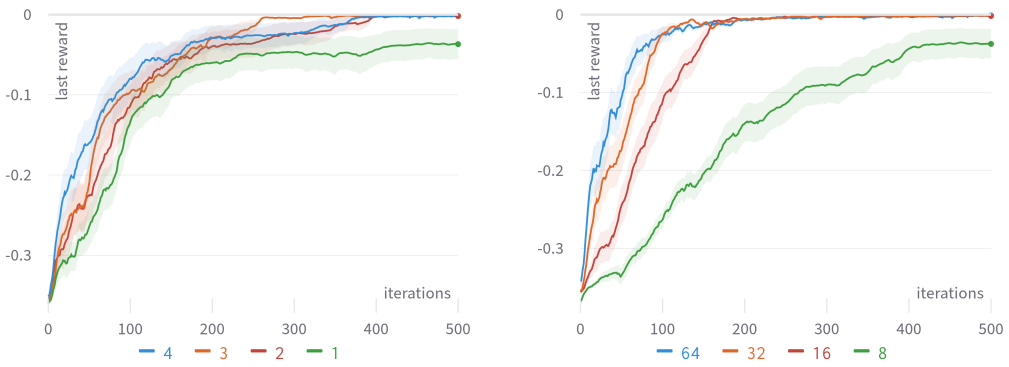
- Most plots show the standard error from the input data.
- The environment always used a torus setup (position is wrapped using modulo, no borders).
- Observations did not directly include the positions of agents or evaders which would trivialize the tasks.

- No observation culling was used: Each agent could see every other agent and evader.
- The agents used direct dynamics: Their actions directly influenced movement on the x- and y-Axis.
- Rewards and Observation were normalized using a running mean and std to improve learning.
- Used mean aggregation.
- Independet GNN-stack for the actor and critic, which used residual connections.
- In heterogeneous graphs we used concatenation of the edge-types not aggregation

During the evaluation of the experiments, we experienced unexpected hardware issues that caused a subset of our runs to crash. Due to time constraints we were not able to fully replicate the respective experiments. Note that we do not include these crashed runs in our evaluation, and that the resulting grid searches are thus partially incomplete. The general findings of this thesis remain unchanged.

6.1. Proof of Concept

This proof of concept experiment used rendezvous. We wanted to show in a simpler environment that our architecture works in general. Furthermore we wanted to investigate if multiple number of hops can already have an impact on learning for rendezvous without any observation culling. We choose to use 10 agents that have 24 timesteps to meet each other in a torus-world of size 12. For these runs we used latent-dimension in the range of 8 to 64 and 1 to 4 number of hops. The last reward of a given iteration tells us if the agents were able to meet up in the alotted time and therefore we choose that as a comparison.

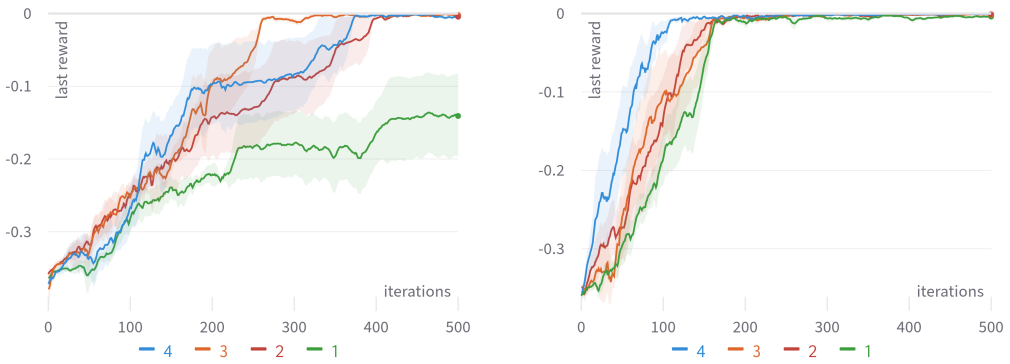


(a) number of hops

(b) latent dimension

Figure 6.1.: Results of the Rendezvous task for different latent dimensions. The left side shows that more hops help in rendezvous, though the effect drops of sharply after 2-3 hops.

Figure 6.1 clearly shows that most of the training environments were able to successfully solve the task. With at least 2 hops, the task was solved after 200 to 350 iterations. The difference in Rendezvous for more than 2 hops is pretty noisy. In early iterations, performance is directly tied to more hops. But later the difference between 2-4 gets smaller. 4 Even struggles to optimize beyond a ceratin threshhold for a while. It is clear that the architecture is able effect the observation input more strongly, and learn faster. Each extra hop added roughly 13% to the time needed to learn, so the benefit is rather small. This was expected though for a rather simple task. If we look at the right site, we can see the effect of different latent dimension to the effectiveness of learning. Only a latent-dimension of 8 was not enough for the agents to meet up in the alotted time. It is clear that the architecture was not able to correctly represent and process observation data for 10 agents.



(a) latent-dimension: 8

(b) latent-dimension: 16

Figure 6.2.: When we filter for latent-dimension of 8, we can clearly see a larger gap between 2-4 hops and just 1 hop on the left side. For latent-dimension larger than 8, this different becomes smaller and more noisy.

If we investigate the behavior for the different latent-dimensions, we can see a large difference in performance shown in Figure 6.2. Latent-dimensions 16 or more were able to solve the task easily. Here the gain of 4 hops is even stronger than comparing it to the general case. If we look at latent-dimension 8 it is clear that a single hop is a very strong outlier. The general performance is much worse, as it is not able to correctly represent the problem at hand. But the overall performance gain of 2 and more hops is much stronger. This shows in larger constraints the ability to process more of the observation data leads to a larger gain.

6.2. Ablation

Which environments used and special setting, (grid)

- Environments: Rendezvous
- Environment: min-num-agent, max-num-agent (also for evaluation environment).

- Network: latent-dimension, num-blocks

Questions to answer: (what we wanted to show)

- How good is the GNN structure able to abstract to different amounts of agents?
-

(Expected) Result:

-

6.3. Different Aggregation Functions

Which environments used and special setting, (grid)

- Environments: Dispersion
- Environment: reward-type
- Network: latent-dimension, num-blocks, aggregation-function

Questions to answer: (what we wanted to show)

- Does the architecture learn better/worse under different reward-functions in dispersion?
- How does number of hops help this?

(Expected) Result:

-

6.4. Under Observation Constraints

Which environments used and special setting, (grid)

- Environments: Rendezvous, Pursuit-Multi
- Environment: Culling Methods: more culling vs less culling
- Network: num-blocks, latent-dimension

Questions to answer: (what we wanted to show)

- Multi-Hops (multiple Layers) => Agents use Evader-Nodes to cache information/data.
- Effect of Multi-Hops on different levels of culling on more difficult tasks. Are they able to negate worse communication ranges? Compare policy on large communication, 1

layer vs. small communication, alot of layers. knn: num-layers: everyone can always communicate.
- What happens with really tight observation range?
- Pursuit-Multi: Strategy better then Robin? (are they able to create groups?).
- Describe the strategies in the different scenarios.

(Expected) Result:

- Harsher Culling => More Layers better. Can partially compensate. Still not the same information. Information about neighbors-neighbor still contains information about neighbor itself => it is influenced.
- policy on large communication, 1 layer > small communication, alot of layers. BUT may take longer?
- More Hops => Usually Better, but has limit. Complexer task => More Hops better (especially Pursuit-Multi).
- tight observation: at certain point it fails, even with alot of layers.

6.5. Neighbor Aggregation Types

Which environments used and special setting, (grid)

- Environments: Pursuit-Single, Pursuit-Multi
- Environment: Base-Pursuit-Multi with 3+ Hops?
- Network: latent-dimension, aggregation-function, neighbor-aggregation (aggr(aggr()) vs concat(aggr())), num-blocks

Questions to answer:

- Agents better/worse being able to distinguish themselves from evaders for concat(aggr())?
- How does this effect aggregation function for aggr(aggr()). Where do I get better performance?
- How is the effect of more/less hopps here?

(Expected) Result:

- concat(aggr()): worse iteration time, but easier to learn heterogeneous graphs.
- aggr(aggr()): probably mean aggregation, there are more features "preserved".
- num-hops: more hops should have more of an effect in concat(aggr()).

6.6. Directed vs Undirected GNN

Which environments used and special setting, (grid)

- Environments: Pursuit-Multi, Pursuit-Single
- Just add this to each of the other experiments (?)
- Environements: use-directed-graph.
- Network: num-blocks, latent-dimension, aggregation-function, neighbor-aggregation, value-function-scope

Questions to answer:

- what "learns" better or has better information propagation

(Expected) Result:

-

6.7. Node-wise vs Graph-wise Value-function

Which environments used and special setting, (grid)

- Environments: Rendezvous, Pursuit-Single
- Just add this to each of the other experiments (?)
- Environements:
- Network: num-blocks, latent-dimension, aggregation-function, neighbor-aggregation, value-function-scope

Questions to answer:

- graph-wise: more "true" to the original Single agent RL algorithms. Should correlate better? But need to figure out themselves which agent is responsible for a reward.
- node-wise: better able to to gauge which agent was responsible for a given value or reward.

(Expected) Result:

-

Chapter 7.

Conclusion and Future Work

7.1. Conclusion

- harsher culling => more layers better. partially compensate, but interpreted data about neighbor's neighbors.
- more hops, better but has limit (more complex, more hops).
- randomized: good abstractions.
- heterogeneous: concat(aggr())?
- GNN Structure: undirected vs directed.
- node-wise or graph-wise value function.

7.2. Future Work

More can be done to expand on the work already finished in this bachelor thesis.

All of the experiments in this paper only considered using Proximal Policy Optimization (PPO) Schulman et al. (2017), as it is a very common baseline training algorithm. However recent research shows that other methods might lead to better results for Multi-Agent Reinforcement Learning. Specifically Trust Region Layers (PG-TRL) Otto et al. (2021) is an alternative that is able to be atleast on par with PPO, while requiring less code-level optimizations. It is noted that in experiments using sparser rewards the fact that TRL has better exploration over PPO improves the results significantly. Though the base paper only explores single-agent problems. Ruede et al. (2021) explores Trust Region Layers (PG-TRL) Otto et al. (2021) for multi-agent tasks. The author explains that given an multi-agent cooperative task the agents are rewarded

as a group, which makes the reward more sparse. Therefore creating correlation of a single agent's action and the group reward is harder. The hyper parameters used for TRL were based on searches for PPO and no extensive testing for TRL was done. Even then TRL was able to perform similar to PPO.
Creating further experiments based on the architecture established in this thesis would very likely benefit from TRL.

Furthermore Ruede et al. (2021) also used more complex multi-agent task than we used in our experiments. In Box Clustering there are agents and multiple boxes. Each box is assigned to one cluster. The goal is to move the boxes, so that the distance between the boxes in a given cluster is minimal. Optimal solutions will require that the agents work together to move the boxes and that they split the work between them. In his thesis his approach worked well for two clusters of boxes, but fell appart with three clusters. Then the agents were only able to move one cluster correctly. Only after increasing the batch size and environment steps per training steps tenfold, the agents were able to consider more than 2 clusters. As explained above, our approach is structurally similar to Ruede et al. (2021) as both can be described with message-passing of GNNs. It was shown that more complex tasks, especially with tight communication ranges, benefitet hugely from multiple message passing hops. So one can assume that we would be able to solve Box Clustering better.

As this thesis is an extension of basic ideas found in Ruede et al. (2021), both thesis are designed to work for a single group of homogeneous agents. As stated in the architecture description, in heterogeneous graphs the policy training method can only use one node type. It is always trained on the agent node features. It would be possible to parameterize the node type it trains on. In team-based tasks you can have multiple competing groups of agents. Our architecture could support two policies that can be trained on different node types and therefore groups of agents.

7.3. Acknowledgements

The authors acknowledge support by the state of Baden-Württemberg through bwHPC.

Bibliography

M. Aurangzeb, F. L. Lewis, and M. Huber. Efficient, swarm-based path finding in unknown graphs using reinforcement learning. In *2013 10th IEEE International Conference on Control and Automation (ICCA)*, pages 870–877, 2013. doi: 10.1109/ICCA.2013.6564940.

T. Chu, J. Wang, L. Codecà, and Z. Li. Multi-agent deep reinforcement learning for large-scale traffic signal control. *IEEE Transactions on Intelligent Transportation Systems*, 21(3): 1086–1095, 2020. doi: 10.1109/TITS.2019.2901791.

L. Engstrom, A. Ilyas, S. Santurkar, D. Tsipras, F. Janoos, L. Rudolph, and A. Madry. Implementation matters in deep policy gradients: A case study on PPO and TRPO. *CoRR*, abs/2005.12729, 2020. URL `https://arxiv.org/abs/2005.12729`.

N. Freymuth. Davis project. Part of the research of the ALR research group at the Karlsruher Institute of Technology (KIT), 2021.

F. Otto, P. Becker, V. A. Ngo, H. C. M. Ziesche, and G. Neumann. Differentiable trust region layers for deep reinforcement learning. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=qYZD-AO1Vn`.

R. Ruede, G. Neumann, T. Asfour, and M. Huettenrauch. Bayesian and attentive aggregation for multi-agent deep reinforcement learning. *Autonomous Learning Robotics (ALR)*, 2021. URL `https://phiresky.github.io/masters-thesis/manuscript.pdf`.

J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust region policy optimization. In F. Bach and D. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1889–1897, Lille, France, 07–09 Jul 2015. PMLR. URL `https://proceedings.mlr.press/v37/schulman15.html`.

J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. URL http://arxiv.org/abs/1707.06347.

W. J. Zhou, B. Subagdja, A.-H. Tan, and D. W.-S. Ong. Hierarchical control of multi-agent reinforcement learning team in real-time strategy (rts) games. *Expert Systems with Applications*, 186:115707, 2021. ISSN 0957-4174. doi: https://doi.org/10.1016/j.eswa.2021.115707. URL https://www.sciencedirect.com/science/article/pii/S0957417421010897.

Z. Zhou, W. Zhang, J. Ding, H. Huang, D. M. Stipanović, and C. J. Tomlin. Cooperative pursuit with voronoi partitions. *Automatica*, 72:64–72, 2016. ISSN 0005-1098. doi: https://doi.org/10.1016/j.automatica.2016.05.007. URL https://www.sciencedirect.com/science/article/pii/S0005109816301911.

Appendix A.

Example Appendix

This is an example for an appendix.