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. Hierfür müssen die Agenten etwaige große Menge an Beobachtungen über ihre Umwelt verarbeiten und lernen um in der Lage zu sein das Problem zu lösen. Somit ist eine effiziente Datenstruktur für die Observationen unumgänglich um auch große menge 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 inter-Agentischen Kommunikationen zu repräsentieren. In dieser Arbeit möchten wir die Auswirkung und den Nutzen von GNNs auf das lernen von MARL Problemen untersuchen. Der Fokus liegt dabei 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 einen GNN Teil, welcher problemlos auf beliebig viele Durchläufe skalieren kann. Dieser Ansatz basiert auf der bereits veröffentlichen Arbeit für eine multi-agent deep reinforcement Architektur, welche man in Ruede et al. (2021) findet. Zudem wurde der Ansatz 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 that 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 architecture found in Ruede et al. (2021). Furthermore we expanded it 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

Applications or real-world problems that require a solution.

- Multi-agent: Any cooperative task. Network Systems, Biology: Ants, Bees. GNN: Deep Learning

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!).
- GNN: Graph Convolutional reinforcement learning.

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)
- 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?

-

talking about the structure of the thesis

-

Stuff to talk about in introduction.

- Applications or real-world problems that require a solution.
- Based on Robin Paper => expand for this bachelor thesis with GNN.
- MARL: Good for certain applications like
- GNN more and more popular
- Using GNN for MARL
- What has research focused on?
- Some examples from research what can be done.
- What we set out to do what our basic goal was, that should be a natural conclusion of what we talked about above.

Chapter 2.

Preliminaries

2.1. Reinforcement Learning

- RL
- MDP (?)
- MARL

2.2. Multi-Agent Reinforcement Learning

- MARL
- PoMDP (?)
- Environments, Agents etc. (MAS)

2.3. Message Passing GNN

- vanilla message-passing GNN
- heterogeneous Graphs and Multi-Color Graphs.

2.4. Neural Networks

- NN

Chapter 3.

Related Work

20 referenced papers. 2-3 sections

- RL
  - Swarm RL (max, Robin)
  - PPO
  - TRL
  - homogeneous vs heterogeneous agents.
- GNN
  - GNNs
  - GATs
  - MeshGraphNets

Deisenroth et al. (2013)

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, but it retains the overarching structure. This makes both GNNs interchangeable. Most of the changes needed for heterogeneous graphs is located in the edge-, node- and global-modules.

4.1. PPO - Policy Architecture

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

- Observation as Graph. Graph Generation.
  - Node features is data an agent knows about himself. Edges are observations about neighbors. Notation!
  - Comparison to Robin. self-observe => Node features, observe of neighbors => Edge features.
- Linear Embedding (Action Dimension => Latent Dimension).
  - Can abstract into a more generalized representation of state observation. That allows for learning relevant relations.
- Node features as Input

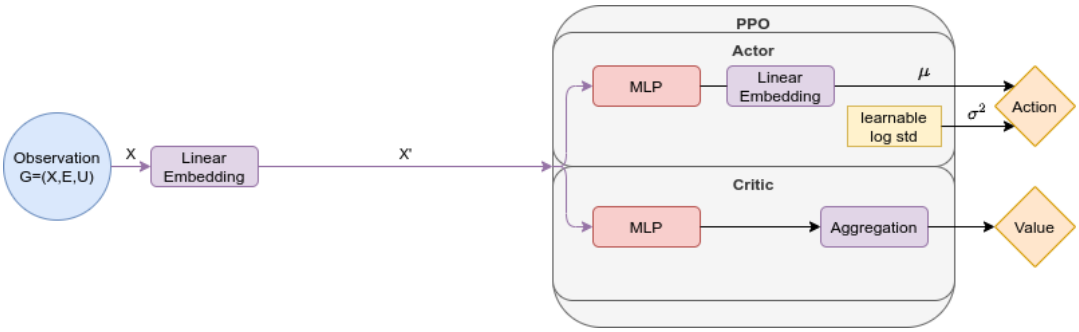


Figure 4.1.: Policy architecture used for Swarm Reinforcement Learning with Graph Neural Networks. The node features of the observation graph is the only data used. It is put into a latent dimension and then used by the PPO Network to calculate the action as a distribution and the value.

- PPO Architecture Similar to Robin.
- Actor: MLP, Linear Embedding (to Action Space), learnable std (Diag Gaussian).
- Critic: MLP, Aggregate (Single Value: min, mean, max), no Embedding (compared to Robin.)

4.2. Homogeneous Message-passing GNN Architecture

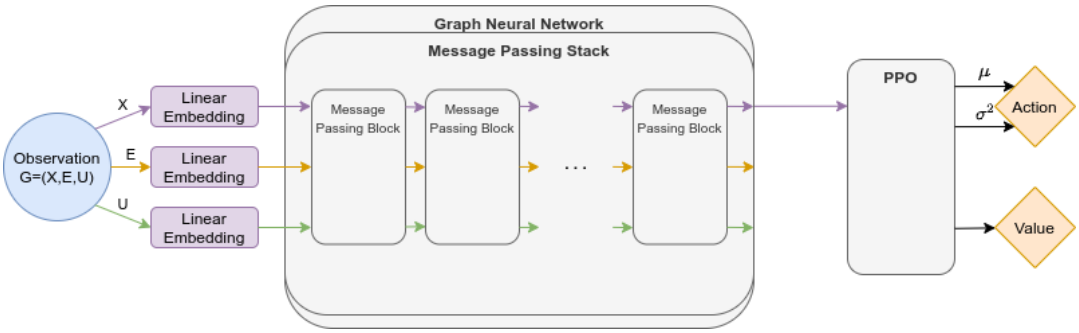


Figure 4.2.: Here a GNN is added as a step before the policy architecture. The entire observation is put into a latent space and then passed through multiple message passing passes. The outputs of one pass is used as the input of the next pass. This is coordinated through the message passing stack.

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

- Observation Embedding. Nodes, Edges and Globals.
  - No shared weights.

- Robin: Basically a Message Passing like structure.
  - For a single observation group:
  - $f(a_i) = decoder(selfObserve(a_i), \underset{j \in \mathcal{N}_i}{aggr}\ encoder(a_i, observe(a_i, a_j)))$
  - $f(x_i) = \phi(x_i, \underset{j \in \mathcal{N}_i}{aggr}\ \psi(x_i, x_j))$
- GNN composed of Stack. Stack are multiple Message Passing Blocks (a single message passing pass) in sequential order.
  - Easy to create multiple hops.
- Block: 3 Modules: Edge-, Node-, Global. How they are wired. Compare that to normal message passing?
- Edge-Module:
  - $x_{e'} = f_e(x_v, x_u, x_e, x_g)$
- Node-Module:
  - $x_{v'} = f_v(x_v, \oplus_{\{e'=(v,u)||_v\}} x_{e'}, x_g)$
- Global-Module:
  - $x_{g'} = f_g(\oplus_V x_{v'}, \oplus_E x_{e'}, x_g)$
- Variations/Parameter/Features:
- share-base:
- aggregation-function used in modules.
- use-residual-connections.

An overview of the homogeneous message passing block can be seen in Figure 4.3.

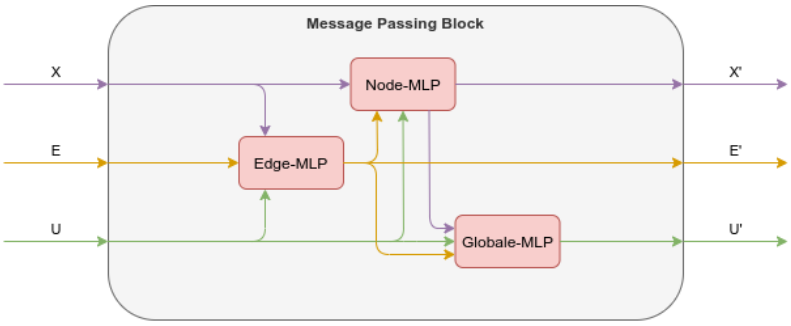


Figure 4.3.: This is a detailed view of one message passing block. It uses a edge-, node- and global-module to compute a single message passing pass.

4.3. Heterogeneous Message-passing GNN Architecture

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

- heterogeneous Graphs, MultiGraph. How the data is layed out (math!). Here shown as multiple Graphs.

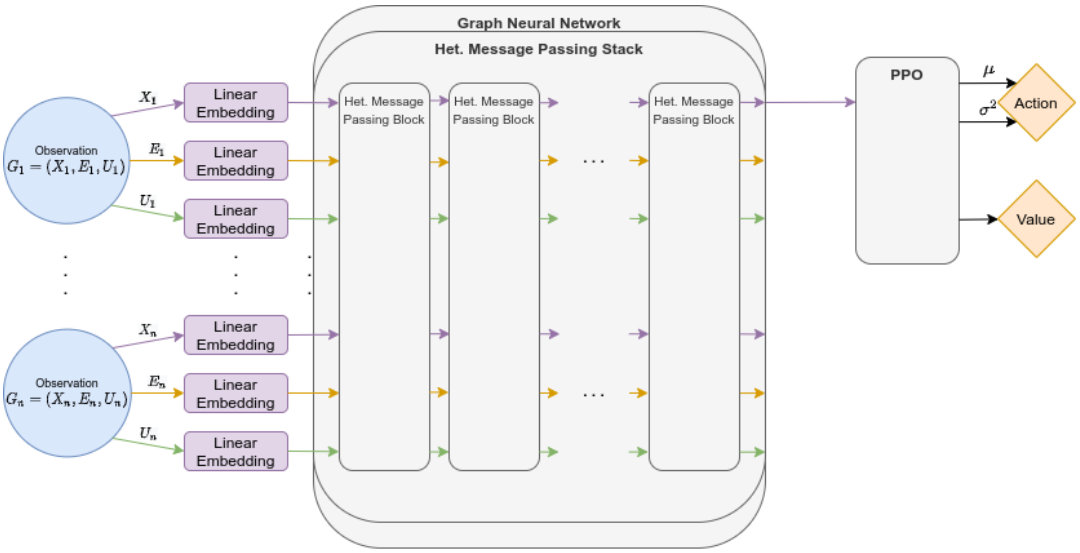


Figure 4.4.: An overview of the heterogeneous version of the architecture. The overall structure stays the same, but the input observation is now mulitple graphs. each of the input embeddings per graph is unique. The policy architecture only uses one of the note types as input.

  - Robin: Needed so that we have support as multiple observation groups. We use heterogeneous graphs.
  - edge-types, node-types (PyG) => We show it using multiple graphs. (Math!)
- Linear Embedder for each type!
  - No shared weights. (Robin: shared weights!) why?
- Stacks and Blocks are basically the same, it just now support more inputs.
- Edge-Module:
  - $x'_{e_i} = f_{e_i}(x_v, x_u, x_{e_i}, x_g), e_i = (u, v)$
  - iterate: edge-types -> node-types
- Node-Module:
  - $x'_{v'_i} = f_{v_i}(x_{v_i}, [\otimes_j \oplus_{\{e_j=(v_i,u)\}} x'_{e_j}], x_g)$
  - iterate: node-types -> edge-types
- Global-Module:
  - $x'_g = f_g([\otimes_j \oplus_{V_j} x'_{v \in V_j}], [\otimes_k \oplus_{E_k} x'_{e \in E_k}], x_g)$
- neighbor-aggregation-types concat(aggr()), aggr(aggr())
  - concat(aggr()) (Robin): More expressive, more expensive.
  - aggr(aggr()) (Robin): less expressive, less expensive. In multi-pursuit: hard to distinguish agents from evaders.

An overview of the heterogeneous message passing block can be seen in Figure 4.5.

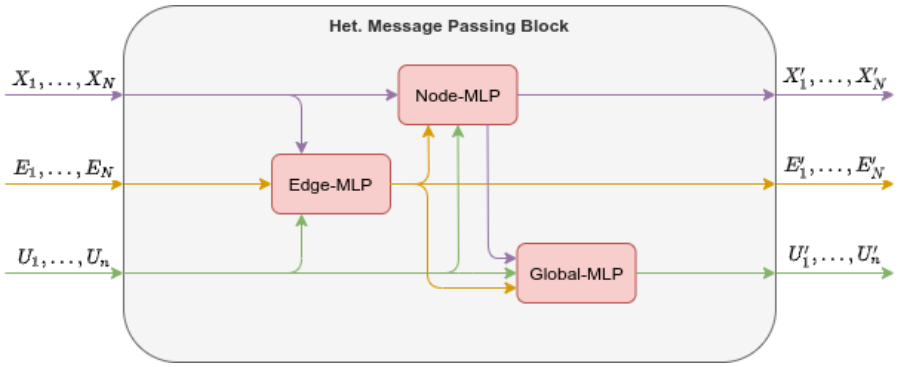


Figure 4.5.: Also the structure of the message passing block stays similar. Now the three blocks take all the nodes, edges and global-features of all graphs as input and as output.

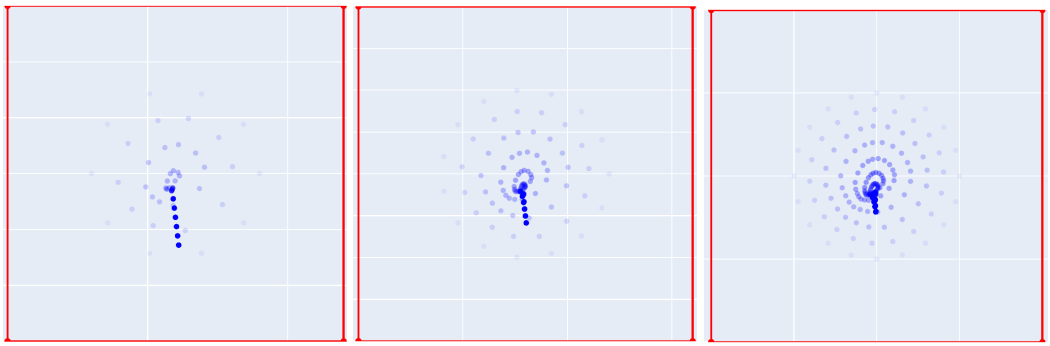
Chapter 5.

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 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 needed tasks.

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 need. If not stated otherwise, actor and critic use different GNNs.



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

Figure 5.1.: An example for a successfull rendezvous episode

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.

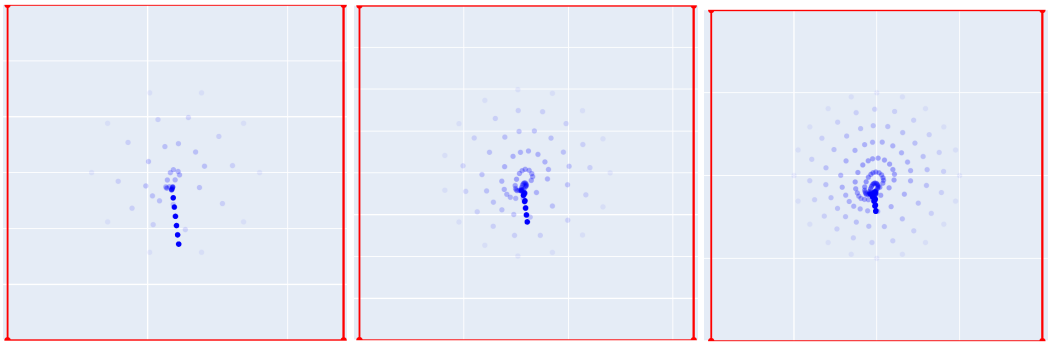
The environment can be configured as a torus (position is wrapped using modulo) or as an rectangular world with borders (position is clipped). Positions are using floating-point precision. It terminates after a given amount of timesteps. The agents are dots without collisions. They use a direct dynamic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

$$a_p = mean(a^2),\ d_p = mean(\frac{\text{pairwise-distances}}{\text{worldsize}}),\ r = a_p + d_p$$

A culling method is used, so that the agents have a finite sensor range, either kNN or euclidean distance can be used. 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) 10 Agents (b) 15 Agents (c) 20 Agents

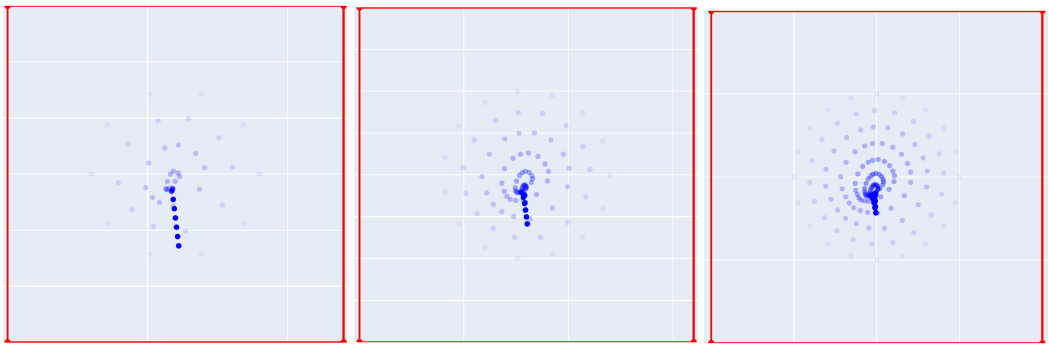
Figure 5.2.: An example for a successfull dispersion episode

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 $f(x) = x$, $f(x) = x^2$, $f(x) = \sqrt{x}$, $f(x) = \min(x)$, $f(x) = \max(x)$

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

5.2.3. Single Evader Pursuit



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

Figure 5.3.: An example for a successfull single evader pursuit episode

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 work cooperate. An example episode can be seen in Figure 5.3.

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 on 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 action penalty a_p is the same as rendezvous:

$$a_p = mean(a^2),\ d_p = mean(\frac{\text{agent-evader-distances}}{\text{worldsize}}),\ r = a_p + d_p$$

Single evader pursuit also supports the same culling methods as rendezvous. The observation graph is heterogeneous and composed of the following aspects:

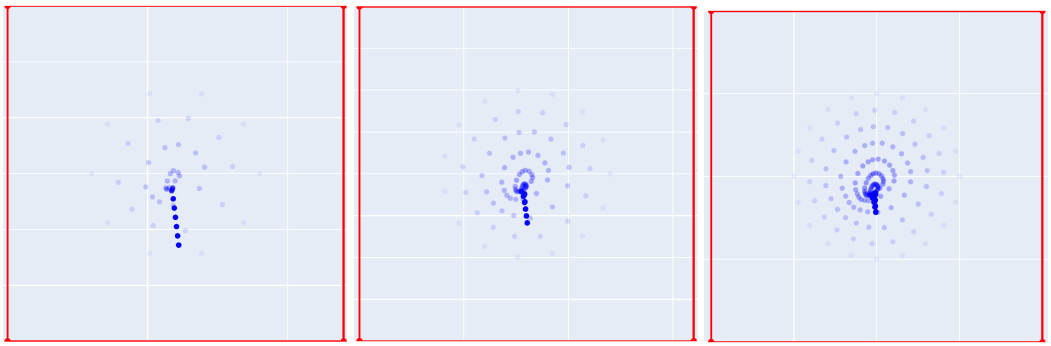
- agent node features:
    1. normalized agent positions (optional)
- evader node features:
    1. normalized evader positions (optional)
- agent-to-agent edge features:
    1. normalized pairwise agent distances
- agent-to-evader edge features:
    1. normalized agent to evader distances
- global features: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

5.2.4. Multi Evader Pursuit

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 action penalty a_p and a catch count c. It will reward the agents with +1 every time they catch an evader:

$$a_p = mean(a^2),\ r = a_p + c$$



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

Figure 5.4.: An example for a successfull multi evader pursuit episode

Chapter 6.

Evaluation

6.1. Number of Hops

Questions to answer:

- Multi-Hops (multiple Layers) => Agents use Evader-Nodes to cache information/data.
- In which situations do you benefit from more layers? Show benefit of multiple layers.
- 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 Answers:

- 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.

- Environments: Rendezvous, Pursuit-Multi
- Environment: Culling Methods: more culling vs less culling, num-agents and dynamics?
- Network: num-blocks, latent-dimension?, aggregation-function?

6.2. Neighbor Aggregation Type

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 Answers:

- 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()).

aggr(aggr()) vs concat(aggr())

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

6.3. Randomized Number of Agents and Evaders

Questions to answer:

- How good is the architecture able to abstract task to random number of agents?
- How does this effect num-blocks?
- Extrapolate number of agents? Range (5-10), Evaluate on 10, 15, 20? and without range? (5?)
- How is this effected by culling?

Expected Answers:

- Random abstraction based on the range. More Randomness to train to => Better.
- Extrapolate: Better with range or without?
- In theory GNN are not effected by rearranging input data and latent-dimension stays the same!

- Culling: Should learn to work better with varying effective communication ranges (hops).

random number of agents

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

random number of agents + random number of evaders

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

6.4. PPO - Code Level optimizations

Questions to answer:

- How do these optimizations effect num-layers?

Expected Answers:

- a

29.11: list of new PPO features implemented!

- Environments: Rendezvous
- value-function-clipping (0.0 - 1.0), 1.0 = no clipping
- normalize rewards
- reward-clipping: graph-normalized constructor: reward-clip = 5, currently no parameter
- observation-normalization
- global gradient clipping: max-grad-norm
- tanh (insted of LeakyReLU)
- can those have an effect on num-layers, aggregation-function and neighbor-aggregation?

6.5. Dispersion

Questions to answer:

- How do different aggregation-functions and reward-type learn easier/harder?

- How is this effected by num-layers? with certain aggregation-function/reward-type combinations?

Expected Answers:

- a

- Environments: Dispersion
- Environment: Culling Methods: more culling vs less culling, reward-type
- Network: latent-dimension, aggregation-function, num-blocks

Chapter 7.

Conclusion and Future Work

7.1. Conclusion

- What comparison we made on different tasks.
- What have we observed
- What have we shown of results.
- Basically a summary of "evaluation":

Number of Hops
a

Neighbor Aggregation Type
a

Randomized Number of Agents and Evaders
a

PPO Hacks vs No Hacks
a

Dispersion
a

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 mult-iagent 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 agents 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 3 clusters. Here 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 structuraly 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.

TODO!!!!! Personalized rewards or responsibility assignment. Might be easier to learn, because it can better correlate which actions where how good. Need to change critic, as it currently only uses the aggregate of the values. Needs to use each value itself for the critic? Look at some papers and how they do that. This can also help in supporting heterogeneous

agents. Agents different strengths and weaknesses => can better understand how to use them if they are not aggregated. because actions are already used as non aggregated. (PAPER!)

Bibliography

M. P. Deisenroth, G. Neumann, and J. Peters. *A survey on policy search for robotics*. now publishers, 2013.

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, 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.

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.