Bayesian and Attentive Aggregation for Cooperative Multi-Agent Deep Reinforcemen Learning

This manuscript (permalink) was automatically built from phiresky/masters-thesis@4754703 on July 22, 2021.

Authors

- **Robin Ruede**
  · phiresky
  Autonomous Learning Robots Lab, Karlsruhe Institute of Technology

Abstract

Multi-agent reinforcement learning (MARL) is an emerging field in reinforcement learning with real world applications such as unmanned aerial vehicles, search-and-rescue, and warehouse organization. There are many different approaches for applying the methods used for single-agent reinforcement learning to MARL. In this work, we survey different learning methods and environment properties and then focus on a problem that persists through most variants: How should one agent use the information gathered from a large and varying number of observations of the world in order to make decisions? We focus on three different methods for aggregating observations and compare them regarding their training performance and sample efficiency. We introduce a policy architecture for aggregation based on Bayesian conditioning and compare it to mean aggregation and attentive aggregation used in related work. We show the performance of the different methods on a set of cooperative tasks that can scale to a large number of agents, including tasks that have other objects in the world that need to be observed by the agents in order to solve the task.

We optimize the hyperparameters to be able to show which parameters lead to the best results for each of the methods. In addition, we compare different variants of Bayesian aggregation and compare the recently introduced Trust Region Layers learning method to the commonly used Proximal Policy Optimization.

1. Introduction

Ant colonies, bee swarms, fish colonies, and migrating birds all exhibit swarming behavior to achieve a common goal or to gain an advantage over what's possible alone. Each individual within a swarm usually only has limited perception, and often there is no central control that enforces a common goal or gives instructions. Swarms of animals can self-organize and exhibit complex emergent behavior in spite of the limited intelligence, limited perception, and limited strength of every individual.

In recent years and with the advent of deep reinforcement learning, it has become possible to create artificial agents that solve fairly complex tasks in simulated environments as well as the real world, even when the path to the goal is difficult to identify and the reward is sparse. Most of the research, however, is focused on a single agent interacting with a world, and giving the single agent as much power and flexibility as it needs to solve the task. Akin to animal swarms, which often consist of fairly simple and limited individuals, we focus on simple artificial agents that need to cooperate to solve a common task, where the task is either difficult or impossible for an individual agent. The agents thus need to be able to learn to work together and to act even with the limited information they are able to perceive.

Swarms of artificial agents can have multiple applications, some inspired by swarms in the animal kingdom, some going beyond. Robots can be used in logistics to organize warehouses. Swarms of robots can be used to map out dangerous areas, solve mazes, find trapped or injured people in search-and-rescue missions quicker and more safely than humans [1]. Swarms of walking robots could be used for animal shepherding [2]. Drone swarms can be used for entertainment in light shows [3], for advertising, for autonomous surveillance [4], or they could be weaponized and used as military robots [5]. Further in the future, swarms of nanobots could be used in medicine to find and target specific cells in the body [6]. Swarms of autonomous spacecraft can be used to explore space, to harvest resources, and to terraform planets into paperclips [7].

Swarms of homogenous agents can be very resilient. Since they can be designed such that no one individual is a critical component of the swarm as a whole, individual failures do not necessarily result in a critical collapse of the whole swarm.

Robots can be controlled by explicit algorithmic behaviour, but that can be complicated, inflexible and fragile, as is shown by the success in recent applications of reinforcmeent learning to control tasks that have not been solved by pre-programmed algorithms [8]. Controlling robot swarms using policies learned with deep reinforcement learning has promising results in recent literature such as [9].

To apply deep reinforcement learning to multi-agent systems, we need to make a few adjustments to the existing learning algorithms and figure out how best to design the policy network. Specifically, we need a way to feed a large and varying amount of observations from the neighboring agents into the fixed size input of a dense neural network. In this work, we consider three aggregation methods on a set of different multi-agent tasks: Mean aggregation, Bayesian aggregation, and attentive aggregation. Our main goal is to compare these methods with regards to their training performance and sample efficiency. We limit ourselves to a specific subset of MARL tasks that are fully cooperative with a team reward, with homogenous agents in a centralized-learning/decentralized-execution setup.

We first give an overview over all the preliminaries we need for our work in Section 2, including reinforcement learning in general, the multi-agent extensions for reinforcement learning, and the background for the aggregation methods we use. Next, we describe some related work in Section 3. Then, we describe our contribution in Section 4 with details about the policy architecture and the specifics for the different aggregation methods. Our experimental setup, including the specific environments we use to carry out our experiments are described in Section 5. Finally, we show and interpret the results of our experiments in Section 6 and talk about the conclusions we can draw from the experiments as well as the potential for future work in Section 7.

2. Preliminaries

In this chapter, we introduce the preliminaries we need for our work. We describe reinforcement learning and two specific policy gradient training methods used for deep reinforcement learning in general and then the specifics of RL for multi-agent systems. While introducing the different variants and properties of multi-agent reinforcement learning we also describe the related work.

2.1. Reinforcement Learning

Reinforcement learning is a method of training an agent to solve a task in an environment. As opposed to supervised learning, there is no explicit label / path to a solution. Instead, the agent iteratively takes actions that affect its environment, and receives a reward when specific conditions are met. The reward can be dense (positive or negative signal at every step) or very sparse (only a binary reward at the end of an episode).

Reinforcement learning problems are usually defined as Markov Decision Processes (MDPs). An MDP is an extension of a Markov chain, which is a set of states with a transition probability between each pair of states. The probability only depends on the current state, not the previous states.

MDPs extend Markov chains, adding a set of actions (that allow an agent to influence the transition to the next state) and rewards (that motivate the agent). An MDP thus consists of:

- A set of states S (the *state space*)
- A set of actions A (the *action space*)
- The transition probability that a specific action in a specific state leads to specific second state $T(s, a, s') = P(s'|s, a)$

- The reward function that specifies the immediate reward an agent receives depending on the previous state, the action, and the new state $R : S \times A \times S \to \mathbb{R}$

The MDP can either run for an infinite period or finish after a number of transitions (the transition function then always returns the same state).

The method by which an agent chooses its actions based on the current state is called a *policy*. The policy defines a probability for taking each action based on a specific state.

Based on a policy we can also define the *value function*, which is the sum of all future rewards that an agent gets based on an initial state and a specific policy. The value function can also include an exponential decay for weighing future rewards, which is especially useful if the horizon is infinite.

Often we need to extend MDPs to allow for an observation model: A *partially observable Markov decision process* (POMDP) is an extension to MDPs that introduces an indirection between the set of states and the input to the policy (called an *observation*). In the definition of a POMDP a set of observations is added with a probability of each state leading to a specific observation. The policy now depends on the observation and the action instead of the state and the action. An observation does not necessarily include all the information from the corresponding state.

To successfully solve a reinforcement learning task, we need to find a policy that has a high expected reward — we want to find the *optimal policy function* that has the highest value function on the initial states of our environment. Since finding the optimal policy directly is impossible for even slightly complicated tasks, we instead use optimization techniques.

2.2. Actor-Critic Training Methods

Policy gradient training methods are a reinforcement learning technique that optimize the parameters of a policy using gradient descent. Actor-critic methods optimize both the policy and an estimation of the value function at the same time.

There are multiple commonly used actor critic training methods such as Trust Region Policy Optimization (TRPO) [10], Proximal Policy Optimization (PPO) [11] and Soft Actor Critic [12]. We run most of our experiments with one of the most commonly used methods (PPO) and also explore a new method that promises stabler training (PG-TRL).

2.2.1. Proximal Policy Optimization (PPO)

Proximal Policy Optimization [11] is an actor-critic policy gradient method for reinforcement learning. In PPO, each training step consists of collecting a set of trajectory rollouts, then optimizing a "surrogate" objective function using stochastic gradient descent. The surrogate objective function approximates the policy gradient while enforcing a trust region by clipping the update steps. PPO is a successor to Trust Region Policy Optimization (TRPO) with a simpler implementation and empirically better performance.

PPO optimizes the policy using

$$\theta_{k+1} = \text{argmax}_\theta E_{s,a\sim\pi_{\theta_k}}[L(s, a, \theta_k, \theta)]$$

Where π_{θ_k} is a policy with parameters θ in training step k, s is the state, $a \sim \pi_{\theta_k}$ is the action distribution according to the policy at step k. L is given by

$$L(s,a,\theta_k,\theta) = \min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}A^{\pi_{\theta_k}}(s,a), \operatorname{clip}(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}, 1-\varepsilon, 1+\varepsilon)A^{\pi_{\theta_k}}(s,a)\right).$$

A is the advantage of taking a specific action in a specific state as opposed to the other actions as weighted by the current policy, estimated using Generalized Advantage Estimation [13] based on the estimated value function.

Since the performance of PPO depends on a number of implementation details and quirks, we use the stable-baselines3 [14] implementation of PPO-Clip, which has been shown to perform as well as the original OpenAI implementation [15].

We use PPO as the default for most of our experiments since it is widely used in other deep reinforcement learning work.

2.2.2. Trust Region Layers (PG-TRL)

Differentiable trust region layers are an alternative method to enforce a trust region during policy updates introduced by [16]. PPO uses a fixed clipping ratio to enforce the trust region, which can result in unstable training. Instead of using a fixed clipping boundary, in PG-TRL a surrogate policy is created with a new layer at the end that projects the unclipped policy back into the trust region. The trust region and the projection is based on either the KL-divergence [17] or the Wasserstein W_2 distance [18].

After each training step, the projected new policy depends on the previous policy. To prevent an infinite stacking of old policies, the explicitly projected policy is only used as a surrogate, while the real policy as based on a learned approximation. To prevent the real policy and the projected policy from diverging, the divergence between them is computed again in the form of the KL divergence or the Wasserstein W_2. The computed divergence (*trust region regression loss*) is then added to the policy gradient loss function with a high factor.

We explore PG-TRL as an alternative training method to PPO.

2.3. Multi-Agent Reinforcement Learning (MARL)

Usually, reinforcement learning algorithms operate on POMDPs. POMDPs only have a single agent interacting with the environment, so for multi-agent settings, we need a different system that can model multiple agents interacting with the world. There are different ways of extending POMDPs to work for multi-agent tasks. Decentralized POMDPs (Dec-POMDP) [19] are a generalization of POMDPs that split the action and observation spaces

- A set of n agents $D = \{1, \dots, n\}$
- A set of states S
- A set of joint actions A
- A set of transition probabilities between states $T(s,a,s') = P(s'|s,a)$
- A set of joint observations O
- An immediate reward function $R: S \times A \to \mathbb{R}$

Note that this definition is very similar to a normal POMDP. The difference is that the set of joint observations and joint actions consists of a tuple of the observations/actions of each individual agent (i.e. $a \in A = \langle a_1, \dots, a_n\rangle$, where a_i is the action of agent i). In Dec-POMDPs every agent can have differing observations and actions.

2.4. Environment Model and Learning Process

In our experiments, we impose a set of restrictions on the environments and learning process. The restrictions we impose are mostly based on [20]. Here, we describe the major differing factors of both the learning process and the environments, as well as the variants we choose to consider.

In general, the agents in a multi-agent environment can differ in their intrinsic properties. For example, they can have different control dynamics, maximum speeds, different observation systems, or different possible actions. We only consider environments with homogenous agents: All agents have the same physical properties, observation space, and action space. They only differ in their extrinsic properties, e.g., their current position, rotation, and speed. This also causes them to have a different perspective, different observations and thus different actions, resulting in differing behavior even when they are acting according to the same policy.

We thus only consider a subset of Dec-POMDPs, namely those where the agents are homogenous — each agent has the same possible observations and actions. This has also been described as a SwarMDP by [21].

We only consider cooperative environments, and we use the same reward function for all agents. Real-world multi-agent tasks are usually cooperative since in adversarial environments, where different agents have different goals, one entity would not have control over multiple adversarial parties.

We focus on global visibility since the additional noise introduced by local observability would be detrimental to the quality of our results.

For training, we use the centralized-learning/decentralized-execution (CLDE) approach — a shared common policy is learned for all agents, but the policy is executed by each agent separately.

PPO and other policy gradient methods are designed for single-agent environments. We adapt PPO to the multi-agent setting without making any major changes: The policy parameters are shared for all agents, and each agent gets the same global reward. The value function for advantage estimation is based on the same architecture as the policy. Since the stable-baselines3 implementation of PPO is already written for vectorized environments (collecting trajectories from many environments running in parallel), we create a new VecEnv implementation that flattens multiple agents in multiple environments.

Similarly to the setup used for TRPO by [20], we collect the data of each agent as if that agent was the only agent in the world. For example, a batch of a single step of 10 agents each in 20 different environment becomes 200 separate training samples. Each agent still only has access to its own local observations, not the global system state. This means that during inference time, each agent has to act independently, based on the observations it makes locally.

2.5. Aggregation Methods

The observations of each agent in an MARL task contains a varying number of observables. The observables can be clustered into groups where each observable is of the same kind and shape. For example, one observable group would contain all the neighboring agents, while another would contain, e.g., a specific type of other observed objects in the world.

We need a method to aggregate the observations in one observable group into a format that can be used as the input of a neural network. Specifically, this means that the output of the aggregation method needs to have a fixed dimensionality. In the following, we present different aggregation methods and their properties.

2.5.1. Concatenation

The simplest aggregation method is concatenation, where each observable is concatenated along the feature dimension into a single observation vector. This method has a few issues however: We can have a varying number of observables, but since the input size of the neural network is fixed, we need to set a maximum number of observables, and set the input dimensionality of the policy architecture proportional to that maximum. Concatenation also ignores the other useful properties of the observables, namely the uniformity (the feature at index i of one observable has the same meaning as the feature at index i of every other observable in a group) and the exchangeability (the order of the observables is either meaningless or variable).

Concatenation scales poorly with a large number of observables since the input size of the neural network has to scale proportionally to the maximum number of observables. It is used for example by [22] to aggregate the neighboring agents' observations and actions.

2.5.2. Mean Aggregation

Instead of concatenating each element o_i in an observable group O, we can also interpret each element as a sample of a distribution that describes the current system state. We use the empirical mean of the samples to retrieve a representation of the system state ψ_O based on all observed observables, as shown in Equation 1.

$$\psi_O = \mu_O = \frac{1}{|O|} \sum_{o_i \in O} o_i \qquad (1)$$

The encoder is an arbitrary function that maps the observation into a latent space, and can be represented by a neural network with shared weights across the observables in an observable group. [20] used mean aggregation for deep multi-agent reinforcement learning and compared it to other aggregation methods. [23] applied mean aggregation to more complex tasks in more realistic simulated environments.

Mean aggregation is strongly related to mean field theory. Mean field theory is a general principle of modeling the effect that many particles have by averaging them into a single field, ignoring the individual variances of each particle. The application of mean field theory for multi-agent systems were formally defined by [24] as *Mean Field Games*. In MARL, mean field Q-learning and mean field actor-critic was defined and evaluated by [25]. [26] use mean fields productively for control of numerous unmanned aerial vehicles.

2.5.3. Aggregation With Other Pooling Functions

Instead of using the empirical mean, other aggregation methods can also be used:

Max pooling:

$$\psi_O = \max_{o_i \in O} o_i$$

Softmax pooling:

$$\psi_O = \sum_{o_i \in O} o_i \frac{e^{o_i}}{\sum_{o_j \in O} e^{o_j}}$$

Max-pooling is widely used in convolutional neural networks to reduce the image dimensionality where it consistently outperforms mean (average) pooling. Softmax aggregation was used by [27] for MARL.

2.5.4. Bayesian Aggregation

Aggregation with Gaussian conditioning works by starting from a Gaussian prior distribution and updating it using a probabilistic observation model for every seen observation. Gaussian conditioning is widely used in applications such as Gaussian process regression [28].

We consider Bayesian aggregation as defined by [29, sec 7.1]. We introduce z as a random variable with a Gaussian distribution:

$$z \sim \mathcal{N}(\mu_z, \sigma_z^2) \quad (p(z) \equiv \mathcal{N}(\mu_z, \sigma_z^2))$$

Initially, this random variable is estimated using a diagonal Gaussian prior as an a-priori estimate:

$$p_0(z) \equiv \mathcal{N}(\mu_{z_0}, diag(\sigma_{z_0}^2))$$

This prior is then updated with Bayesian conditioning using each of the observed elements r_i. We interpret each observation as a new sample from the distribution $p(z)$, each with a mean r_i and a standard deviation σ_{r_i}. We use the probabilistic observation model and consider the conditional probability

$$p(r_i|z) \equiv \mathcal{N}(r_n|z, \sigma_{r_i}^2).$$

With standard Gaussian conditioning [30] this leads to the closed form factorized posterior description of z:

$$\sigma_z^2 = \frac{1}{\frac{1}{\sigma_{z_0}^2} + \sum_{i=1}^n \frac{1}{\sigma_{r_i}^2}}$$

$$\mu_z = \mu_{z_0} + \sigma_z^2 \cdot \sum_{i=1}^{n} \frac{(r_i - \mu_{z_0})}{\sigma_{r_i}^2}$$

Bayesian aggregation was used by [29] for context aggregation in conditional latent variable (CLV) models. There is no related work using Bayesian aggregation for MARL.

2.5.5. Attention Mechanisms

Attention mechanisms are commonly used in other areas of machine learning.

An attention function computes some compatibility between a *query* Q and a *key* K, and uses this compatibility as the weight to compute a weighted sum of the *values* V. The query, key, and value are all vectors.

The multi-head attention mechanism was introduced by [31] and is the core of the state-of-the-art model for many natural language processing tasks.

We only consider the specific multi-head residual masked self attention variant of attention mechanisms for observation aggregation used by [9].

There, the attention function in is a scaled dot-product attention as described by [31]:

$$\mathrm{Attention}(Q, K, V) = \mathrm{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

d_k is the dimensionality of the key K.

Instead of using only a single attention function, [31] uses multiple independent attention heads. The inputs (Q, K, V) of each of the heads $1, \ldots, n$ as well as the concatenated output is transformed with a separately learned dense layers (described as weight matrices W_i^Q, W_i^K, W_i^V, W^O). The full multi-head attention module $\mathrm{MHA}()$ thus looks like this:

$$\mathrm{head}_i = \mathrm{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

$$\mathrm{MHA}(Q, K, V) = \mathrm{concat}(\mathrm{head}_1, \ldots, \mathrm{head}_n)W^O$$

For self-attention, the query Q, the key K and the value V are all set to the same input value. In [9], the authors combine the multi-head attention with a mean aggregation. Note that they only use the attention mechanism to individually transform the information from each separate observable into a new feature set instead of directly using it as a weighing function for the aggregation.

[32] use a different approach with a decentralized policy and a centralized critic. An attention mechanism is used to weigh the features from the other agents, then aggregates them to calculate the value function.

[33] first encode all aggregatable observations together with the proprioceptive observations with an “extrinsic encoder”, then compute attention weights using another encoder from that and aggregate the features retrieved from a separate “intrinsic encoder” using those weights.

2.5.6. Other Aggregation Methods

[20] also did experiments with aggregation into histograms and aggregation with radial basis functions, though the results indicated they were outperformed by a neural network encoder with mean aggregation.

3. Related Work

There are many variants of applying reinforcement learning to multi-agent systems.

An overview over recent MARL work and some of the differing properties can be found in [34] and [35].

3.1. Centralized vs Decentralized Learning

During training, the agent policies car either be learned in a centralized or a decentralized fashion.

In decentralized learning, each agent learns their own policy, while in centralized learning, the policy is shared between all agents. Decentralized learning has the advantage that the agents do not need to be homogenous, since the learned policy can be completely different. It also means that it's harder for the agents to learn to collaborate though, since they are not intrinsically similar to each other. Decentralized learning is used in [36–38].

Centralized learning means the training process happens centralized, with a single policy. It has the advantage of only needing to train one policy network and the possibility of being more sample-efficient. Centralized learning requires homogenous agents since the policy network parameters are shared across all agents. It is used for example in [39,40].

3.2. Centralized vs Decentralized Execution

When using centralized learning, it is possible to use a single policy to output actions for all agents at the same time. This is called “centralized execution”. The policy chooses the actions based on the global system state in a “birds-eye” view of the world. The disadvantage is that this means agents can't act independently if the environment has restrictions such as partial observability, and it is less robust to communication failures and to agent “deaths”.

The alternative is decentralized execution — the observation inputs and action outputs of the policy networks are local to a single agent. This can formally be described as a *Decentralized Partially Observable Markov Decision Process* (Dec-POMDP) [41]. When combining centralized learning with decentralized execution, there is only one policy network that is shared between all agents but the inputs and outputs of the policy are separate.

Centralized-learning with decentralized-execution is thus a compromise between performance, robustness, and sample efficiency. CLDE is used e.g. by [20,22,39,40,42].

3.3. Cooperative, Adversarial, Team-Based

Multi-agent environments can be cooperative, adversarial, or team-based. Cooperative environments are those where all the agents have one common goal. In adversarial tasks each agent has their own independent goal that conflicts with the goal of the other agents. An example for a cooperative environment is the rendezvous task: All agents need to meet up at a single point, where the agents have to decide independently on the location. The reward here is the negative average pairwise distance of the agents, see Section 5.2.1. Note that a cooperative environments can also include other adversarial agents, as long as those agents are controlled by an explicit algorithm and not a learned policy. From the perspective of the policy, these agents are considered part of the environment.

Cooperative learning can be done with separate agent rewards or a single common reward. Cooperative learning with a single reward means the reward must be somewhat sparse, since each agent cannot easily judge what the impact of its own actions were on the reward in any given time step. This also makes it impossible for an agent to gain an egoistic advantage over the other agents, enforcing the learned policy to become Pareto optimal. Another approach was introduced by [43], who create a compromise between the sample-efficiency of individual rewards and the Paretooptimality of a common reward for cooperative MARL settings with their *Hysteretic Q-Learning* algorithm that jointly optimizes an individual as well as a common reward.

Adversarial environments are usually zero-sum, that is the average reward over all agents is zero. An example for an adversarial environment is the *Gather* task described by [44]: In a world with limited food, agents need to gather food to survive. They can also kill each other to reduce the scarcity of the food. Another example of an adversarial task is Go [45] as well as many other board games, though these usually have a fairly small number of agents.

Team-based tasks have multiple teams that each have a conflicting goal, but from the perspective of each team the task is cooperative (with a single reward function). The team reward can either be defined directly for the team or by averaging a reward function of each team member.

An example of a team-based environment is the OpenAI Hide-and-Seek Task [9]. In this task, there are two teams of agents in a simulated physical world with obstacles, and members of one team try to find the members of the other team. The Hide-team is rewarded +1 if none of the team members is seen by any seeker, and -1 otherwise. The Seek-team is given the opposite reward.

3.4. Partial Visibility

The observations that each agent receives in our experiments are local. For example, if one agent sees another, that agent's properties are observed relative to the current agent — the distance, relative bearing, and relative speed.

In addition, each agent may only have local visibility, for example it can only observe the positions of agents and objects in the world within some radius or the visibility can be hindered by obstacles. In [20] both the local and global visibility variants of the same tasks were considered.

3.5. Simultaneous vs Turn-Based

In general, multi-agent environments can be turn-based or simultaneous. In turn-based environments each agent acts in sequence, with the world state changing after each turn. In simultaneous environments, all agents act "at the same time", i.e. the environment only changes after all agents have acted in one time step.

Simultaneous environments can be considered a subset of turn-based environments, since a simultaneous environment can be converted to a turn-based one by fixing the environment during turns and only applying the actions of each agent after the time step is finished. For this reason the API of PettingZoo [46] is based around turn-based environments. Examples of turn-based environments include most board games and many video games. Most real world scenarios are simultaneous though, so we only consider environments with simultaneous actions.

3.6. MARL Tasks in Related Work

[9] create a team-based multi-agent environment with one team of agents trying to find and "catch" the agents of the other team.

[33] test their attention-based architecture on three environments: (1) A catching game in a discrete 2D world, where multiple paddles moving in one dimension try to catch multiple balls that fall down the top of the screen. (2) A spreading game, where N agents try to cover N landmarks. The world is

continuous, but the action space is discrete. This game is similar to the spreading game defined in [22]. (3) StarCraft micromanagement: Two groups of units in the StarCraft game battle each other, each unit controlled by an agent.

[46] create an infrastructure for multi-task environments similar to the OpenAI Gym and add standardized wrappers for Atari multiplayer games, as well as a reimplementation of the MAgent environments [44], the Multi-Particle-Environments [22] and the SISL environments [40].

[20] test mean aggregation on two environments: (1) A task where all agents need to meet up in one point (rendezvous task). (2) A task where the agents try to catch one or more evaders (pursuit task). We compare our method on both of these tasks.

[23] define a set of environments based on Kilobots, simple small round robots in a virtual environment together with boxes. The Kilobots are tasked with moving the boxes around or arranging them based on their color. We compare our method on the box assembly as well as the box clustering task.

4. Scalable Information Aggregation for Deep MARL Policies

We introduce a policy architecture for deep multi-agent reinforcement learning that projects observations from one or more different kinds of observables into samples of latent spaces, then aggregates them into a single latent value. This makes the architecture scale to any number as well as a varying number of observables.

4.1. Policy Architecture

In general, our policy architecture is based on [20] and [9].

While the policy architecture and weights are shared for all agents, the inputs are dependent on the current agent a_k, thus leading to different outputs for the action and value function for each agent.

Depending on the task, the inputs differ. In general, we always have the proprioceptive observations (how the agent a_k sees itself), and the observations from the n neighboring agents $a_1, \dots, a_n$. The observations from each neighbor have the same shape, but differ from the self-observations since they can include other information like the distance between a_k and a_i, and they may be missing some information that may not be known to a_k. In addition, there can be additional sets of observations, for example for objects or scripted agents in the environment. The architecture can handle any amount of observation sets, but the observations in each set must have the same shape.

We call each of the sets of observations “aggregation groups”.

First, we collect the observations for each instance $1 < i < n$ of each aggregation group G. These observations can be relative to the current agent. Each observation in the aggregation group G is thus a function of the agent a_k and the instance g_i:

$$o_{k \to g_i} = \mathrm{observe}(a_k, g_i)$$

After collecting the observations for aggregation group g, the observations are each encoded separately into a latent embedding space:

$$e_{k\to g_i} = \mathrm{enc}(o_{k\to g_i})$$

The Encoder $\mathrm{enc}()$ is a dense neural network with zero or more hidden layers.

After being encoded, we interpret each instance of an aggregation group as one sample of a latent space that represents some form of the information of the aggregation group that is relevant to the agent. These samples are then aggregated using one of the aggregation methods described below to get a single latent space value for each aggregation group:

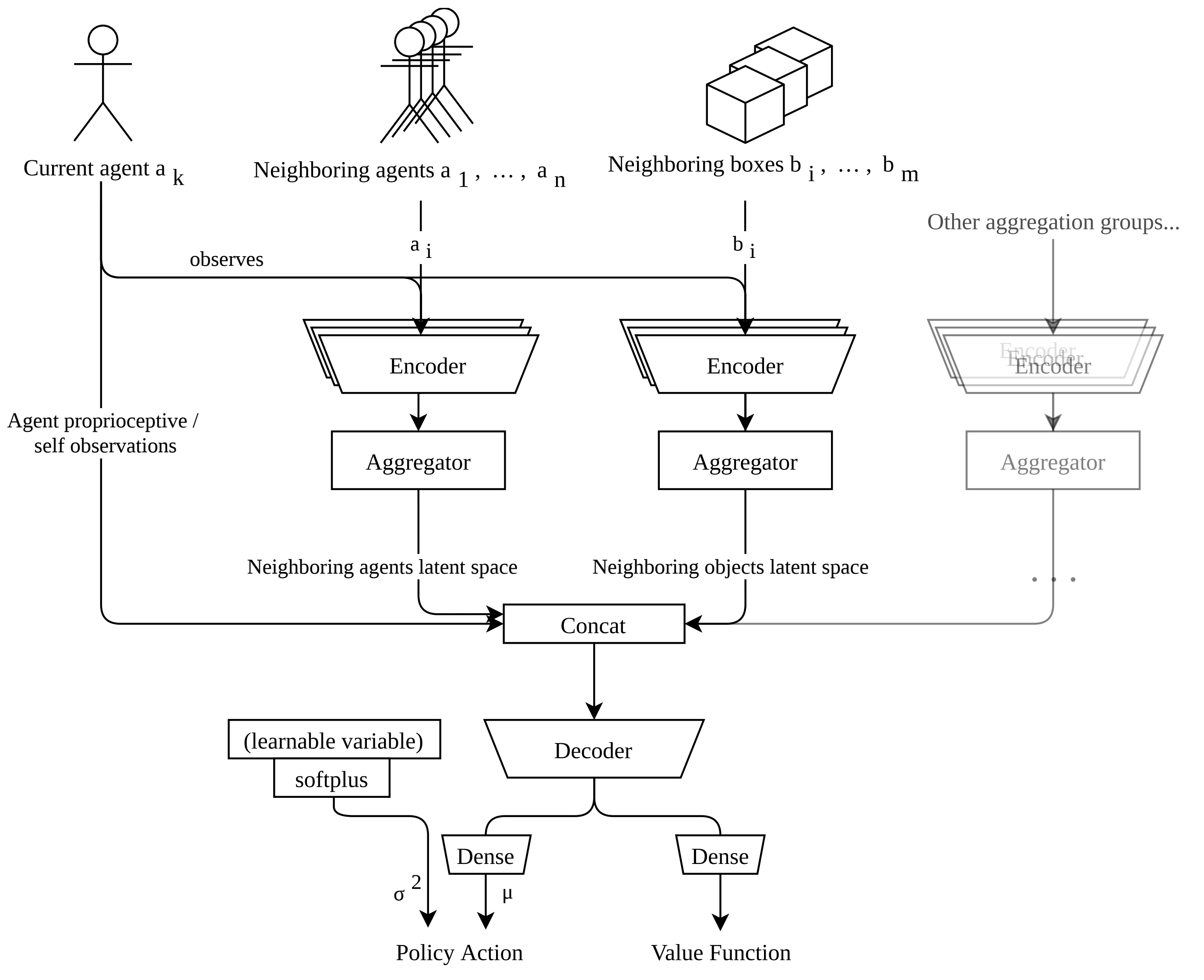
$$e_{k\to G} = \mathrm{agg}_{i=0}^{n}(e_{k\to g_i})$$

We then concatenate all the latent spaces as well as the proprioceptive observations p to get a single encoded value e_k:

$$e_k = (p, G_1, G_2, \dots)$$

This value is then passed through a decoder that consists of one or more dense layers. Finally, the decoded value is transformed to the dimensionality of the action space or to 1 to get the output of the value function. While we share the architecture for the policy and value function, we use a separate copy of the compute graph for the value function, so the weights and training are completely independent.

Figure 1 shows a schematic of the general model architecture described above.



Current agent a k
Neighboring agents a 1 , … , a n
Neighboring boxes b i , … , b m
Other aggregation groups...
observes
a i
b i
Encoder
Encoder
Encoder
Agent proprioceptive / self observations
Aggregator
Aggregator
Aggregator
Neighboring agents latent space
Neighboring objects latent space
Concat
(learnable variable)
softplus
Decoder
Dense
Dense
σ 2
μ
Policy Action
Value Function

Figure 1: A schematic of our general model architecture for deep MARL policies with scalable information aggregation. The observation inputs consist of the agent itself and multiple aggregatable observation groups. The observations from the aggregation groups are each passed though an encoder and aggregated with an aggregator. Afterwards all the aggregated observations are concatenated and decoded to get the policy and value function.

The tasks we consider have a continuous action space. We use a diagonalized Gaussian distribution where the mean μ of each action is output by the neural network while the variance of each action is a free-standing learnable variable only passed through $\exp$ or $\mathrm{softplus}$ to ensure positivity.

4.1.1. Mean/Max/Softmax Aggregation

Each sample in the latent space is weighted by a function $\mathrm{weigh}()$ and aggregated using an aggregation operator $\bigoplus$:

$$e_{k\to G} = \bigoplus_{i=1}^{n} \mathrm{weigh}(e_{k\to g_i})$$

For mean aggregation, the weighing function multiplies by $\frac{1}{n}$ and the aggregation operator is the sum:

$$e_{k \to G} = \sum_{i=1}^{n} \frac{1}{n} \cdot (e_{k \to g_i})$$

For max aggregation, the weight is 1 and the aggregation operator takes the largest value:

$$e_{k \to G} = \max_{i=1}^{n} e_{k \to g_i}$$

For softmax aggregation, the weight is based on the softmax function and the aggregation operator is the sum:

$$e_{k \to G} = \sum_{i=1}^{n} \left(\frac{\exp(e_{k \to g_i})}{\sum_{j=1}^{n} \exp(e_{k \to g_j})} \right) e_{k \to g_i}$$

4.1.2. Bayesian Aggregation

We use a separate latent space and thus a separate observation model $p(z)$ for each aggregation group.

To make the Bayesian aggregation as described in Section 2.5.4 work in our policy network, we need an estimation describing the Gaussian prior (μ_{z_0} and $\sigma_{z_0}^2$) as well as the observed means r_i and variances $\sigma_{r_i}^2$.

The mean and variance of the Gaussian prior are learned as free-standing variables using the backpropagation during training. The prior variance as well as the variance of the observations are rectified to enforce positivity using $\mathrm{softplus}$.

To get the observed elements r_i, we use the two encoder networks enc_r and enc_σ that consist of a set of dense layers:

$$r_i = \mathrm{enc}_r(o_{k \to g_i}), \quad \sigma_{r_i} = \mathrm{enc}_\sigma(o_{k \to g_i})$$

enc_r and enc_σ are either separate dense neural networks or a single dense neural network with the output having two scalars per feature (one for the mean, one for the variance). Finally, we retrieve the value of $e_{k \to G}$ from the aggregated latent variable, using either just the mean of z:

$$e_{k \to G} = \mu_z$$

or by concatenating the mean and the variance:

$$e_{k \to G} = (\mu_z, \sigma_z^2).$$

The mean and variance are calculated from the conditioned Gaussian based on the encoded observations as described in Section 2.5.4:

$$\sigma_z^2 = \frac{1}{\frac{1}{\sigma_{z_0}^2} + \sum_{i=1}^{n} \frac{1}{\mathrm{enc}_\sigma(o_{k \to g_i})^2}}$$

$$\mu_z = \mu_{z_0} + \sigma_z^2 \cdot \sum_{i=1}^{n} \frac{(\mathrm{enc}_r(o_{k\to g_i}) - \mu_{z_0})}{\mathrm{enc}_\sigma(o_{k\to g_i})^2}$$

A graphical overview of this method is shown in Figure 2.

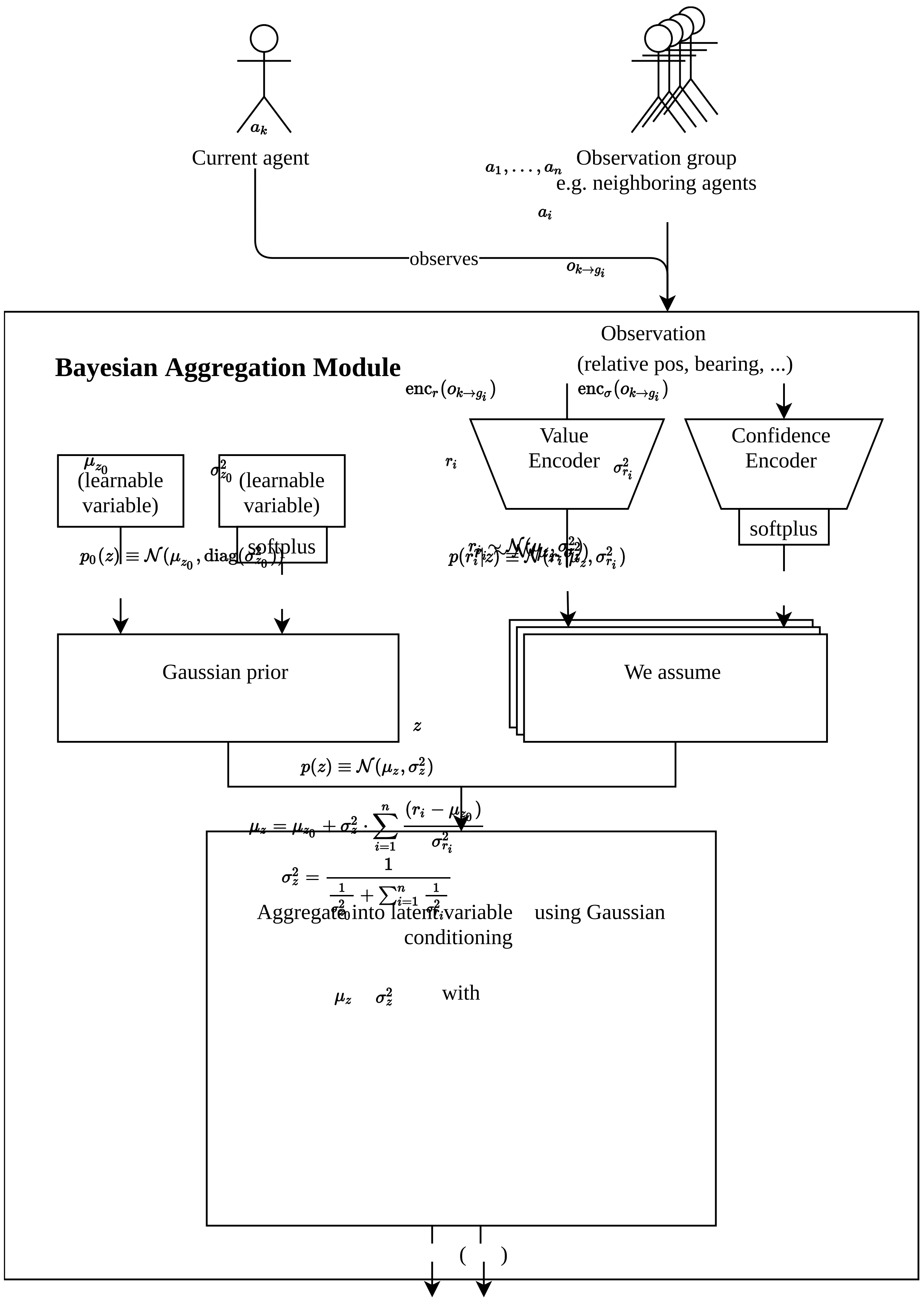

a_k
Current agent
Observation group
$a_1, \ldots, a_n$
e.g. neighboring agents
a_i
observes
$o_{k \to g_i}$
Bayesian Aggregation Module
Observation
(relative pos, bearing, ...)
$\text{enc}_r(o_{k \to g_i})$
$\text{enc}_\sigma(o_{k \to g_i})$
Value Encoder
Confidence Encoder
r_i
$\sigma^2_{r_i}$
softplus
μ_{z_0} (learnable variable)
$\sigma^2_{z_0}$ (learnable variable)
softplus
$p_0(z) \equiv \mathcal{N}(\mu_{z_0}, \text{diag}(\sigma^2_{z_0}))$
$r_i \sim \mathcal{N}(\mu_z, \sigma^2_{r_i})$
$p(r_i|z) \equiv \mathcal{N}(r_i|\mu_z, \sigma^2_{r_i})$
Gaussian prior
We assume
z
$p(z) \equiv \mathcal{N}(\mu_z, \sigma^2_z)$
$\mu_z = \mu_{z_0} + \sigma^2_z \cdot \sum_{i=1}^{n} \frac{(r_i - \mu_{z_0})}{\sigma^2_{r_i}}$
$\sigma^2_z = \frac{1}{\frac{1}{\sigma^2_{z_0}} + \sum_{i=1}^{n} \frac{1}{\sigma^2_{r_i}}}$
Aggregate r_i into latent variable using Gaussian conditioning
μ_z σ^2_z with
( )

Figure 2: Bayesian Aggregation in graphical form. The observations from the neighboring agents are encoded with the value encoder and the confidence encoder. They are then used to condition the Gaussian prior estimate of the latent variable z to get the final mean and variance of the a-posteriori estimate. The mean and optionally the variance estimate of z are concatenated with the latent spaces from the other aggregatables and passed to the decoder as shown in Figure 1.

4.1.3. Attentive Aggregation

When using residual self-attentive aggregation, we define the aggregated feature as

$$e_{k \to g} = \frac{1}{n} \sum_{i=1}^{n} \mathrm{resatt}(\mathrm{enc}(o_i))$$

with enc being a dense neural network with zero or more hidden layers, and

$$\mathrm{resatt}(o_i) = o_i + \mathrm{dense}(\mathrm{MHA}(o_i, o_i, o_i)).$$

Residual means that the input is added to the output of the attention mechanism, so the attention mechanism only has to learn a *residual* value that modifies the input features.

The MHA module is the multi-head attention module from [31] as described in Section 2.5.5.

This method of attentive aggregation is similar to the method successfully used by [9]. An overview over the model architecture used in [9] can be seen in Figure 3.

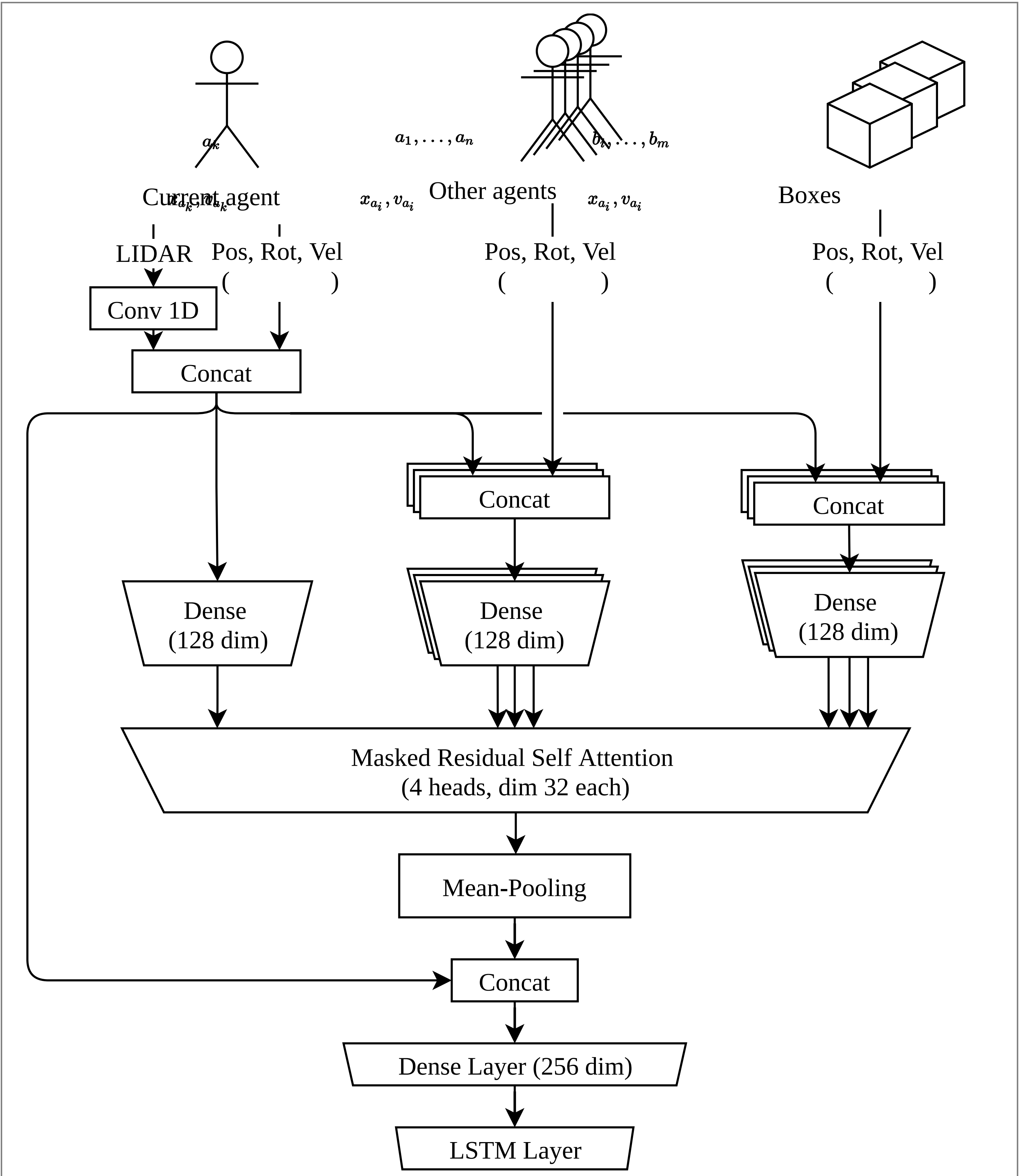


Figure 3: A schematic of the model architecture used by OpenAI [9] using masked residual self-attention. It is similar to our architecture (Figure 1) except for the LIDAR in the self-observations as well as the LSTM layer at the end.

5. Experiments

5.1. Experimental Setup

All of our experiments were run using stable-baselines3 (sb3). *OpenAI Baselines* is a deep reinforcement learning framework that provides implementations of various reinforcement learning algorithms for TensorFlow. Stable-baselines is an extension of the original code base with better documentation and more flexibility for custom architectures and environments. sb3 is the continuation of the stable-baselines project, rewritten using PyTorch.

We extended sb3 for the multi-agent use case by adapting the vectorized environment implementation to support multiple agents in a single environment, as well as adapting the training and evaluation functionality to correctly handle the centralized-learning decentralized-execution method. We also implement Trust Region Layers [16] as a new training algorithm for sb3 in order to be able to directly compare PPO and TRL.

We optimized the hyper parameters using Optuna [47].

{describe batch size, other meta settings}

5.2. Considered Tasks

To evaluate the different aggregation methods we need simulated environments where multiple agents can cooperatively solve a task. Since most of the commonly used tasks used to benchmark reinforcement learning algorithms are designed for a single agent, we use custom built environments.

The following shows which specific tasks we consider.

5.2.1. Rendezvous Task

In the rendezvous task, a set of n agents try to meet up in a single point. An example is shown in Figure 4. Our implementation of the rendezvous task is modeled after [20].

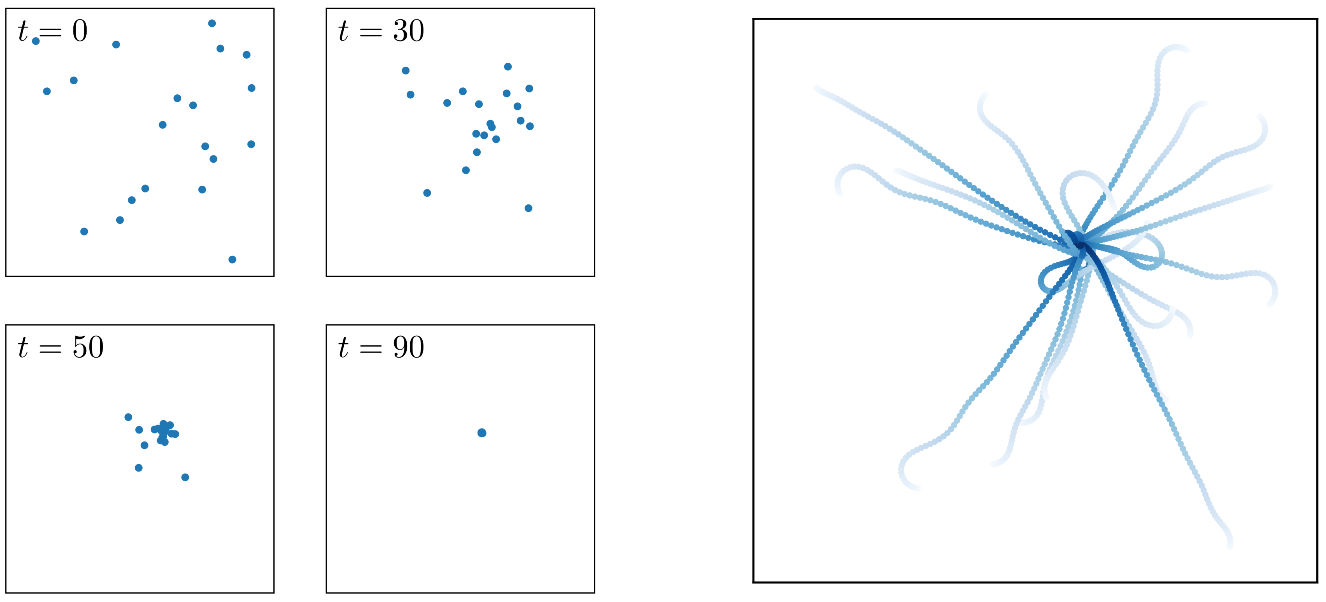


*Figure 4: Visualization of one successful episode of the rendezvous task (from **[20]**)*

The agents are modeled as infinitesimal dots without collisions. They use double-integrator unicycle dynamics [48], so the action outputs are the acceleration of the linear velocity ($\dot{v}$) and the angular velocity ($\dot{\omega}$). The agents move around in a square world that either has walls at the edge or is on a torus (the position is calculated modulo 100).

The observation space of agent a_i comprises the following information:

1. The own linear velocity v_{a_i}
2. The own angular velocity ω_{a_i}
3. The ratio of currently visible agents
4. If the observation space is not a torus: the distance and angle to the nearest wall
5. For each neighboring agent:
    1. The distance between the current agent and the neighbor
    2. The cos and sin of the relative bearing (the angle between the direction the agent is going and the position of the neighbor)
    3. The cos and sin of the neighbor's bearing to us
    4. The velocity of the neighbor relative to the agent's velocity
    5. The ratio of currently visible agents for the neighbor

The reward that every agent gets is the mean of all the pairwise agent distances:

$$r = \frac{1}{n} \sum_{i=0}^{n} \sum_{j=i+1}^{n} ||p_i - p_j||$$

The episode ends after 1024 time steps.

5.2.2. Single-Evader Pursuit Task

In the pursuit task, multiple pursuers try to catch an evader. The evader agent has a higher speed than the pursuers, so the pursuers need to cooperate to be able to catch it. The world is a two-dimensional torus (the position (x, y) are floating point numbers modulo 100). If the world was infinite, the evader could run away in one direction forever, and if it had walls, the pursuers could easily corner the evader. The pursuit tasks are modeled after [20]. An example episode of the pursuit task is in Figure 5.

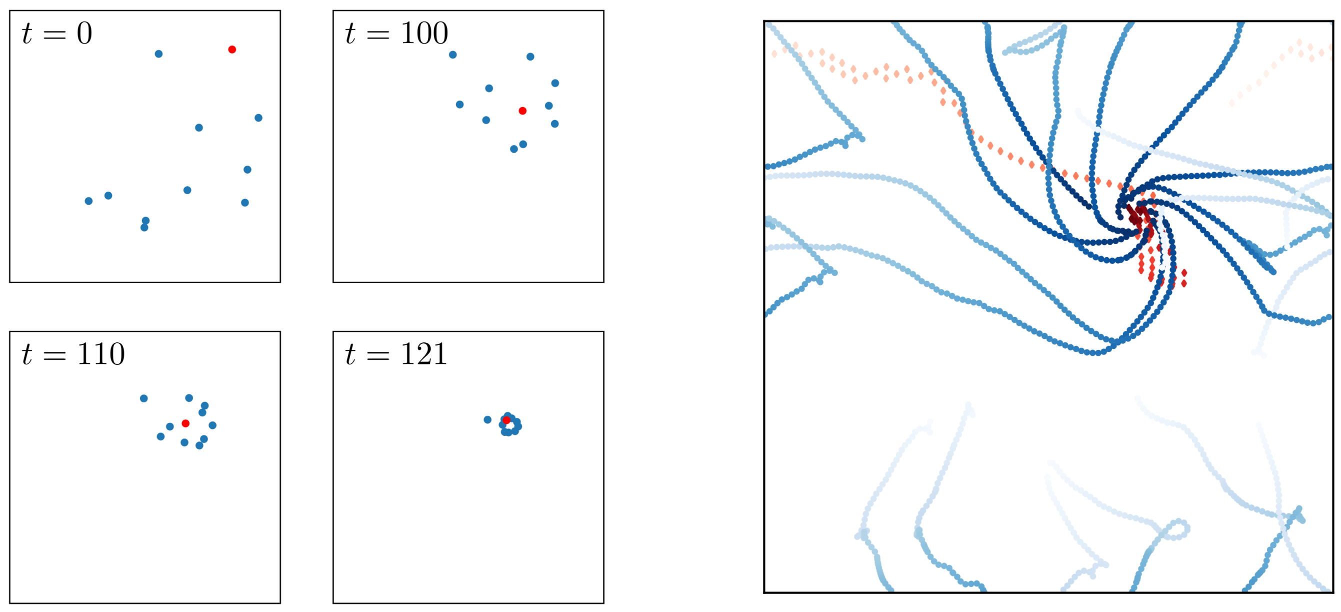


Figure 5: Visualization of one successful episode of the pursuit task (from [20]). The pursuers are in blue, the evader is in red.

The agents are modeled with single-integrator unicycle dynamics. The action outputs are the linear velocity (v) and the angular velocity (ω).

The evader is not part of the training and uses a hard-coded algorithm based on maximizing Voronoi-regions as described by [49]. It is thus part of the environment and not an “agent” as seen from the perspective of the training algorithm.

The observation space for the single-evader pursuit task comprises the following information:

1. The ratio of agents that are visible
2. For each neighboring agent:
    1. The distance between the current agent and the neighbor
    2. The cos and sin of the relative bearing (the angle between the direction the agent is going and the position of the neighbor)
    3. The cos and sin of the neighbor’s bearing to us
3. For each evader:
    1. The distance between the current agent and the evader
    2. The cos and sin of the relative bearing

The reward function of all the agents is the minimum distance of any agent to the evader:

$$r = min_{i=0}^{n} ||p_{a_i} - p_e||$$

The episode ends once the evader is caught or 1024 timesteps have passed. The evader is declared as caught if the distance is minimum distance between an agent and the evader is less than 1 of the world width.

5.2.2.1. Multi-Evader Pursuit

The multi-evader pursuit task is the same as the normal pursuit task, except there are multiple hard-coded evaders.

The reward here is different, since defining the reward as for the single-evader task is not obvious. The reward is +1 whenever an evader is caught, 0 otherwise. An evader is declared caught if the distance to the nearest pursuer is less than $\frac{2}{100}$ of the world width. Contrary to the single-evader task, the episode does not end when an evader is caught and instead always runs for 1024 timesteps.

5.2.2.2. Box Assembly Task

In the box assembly task, the agents are modeled similar to Kilobots, as described by [23]. The agents are two-dimensional circles with collision.

For simplicity, we again use single-integrator unicycle dynamics with the linear velocity v and the angular velocity ω as the action outputs instead of the specific dynamics of real Kilobots.

The world is a square and contains a few two-dimensional boxes (squares) as obstacles. For the box assembly task, the goal is to get all the boxes as close together as possible. Since they agents are much smaller than the boxes, moving a box is hard for a single agent. The reward is the negative sum of the pairwise distances between the boxes. We run this task with four boxes and 10 agents.

An example of a successful episode of the task is in Figure 6.

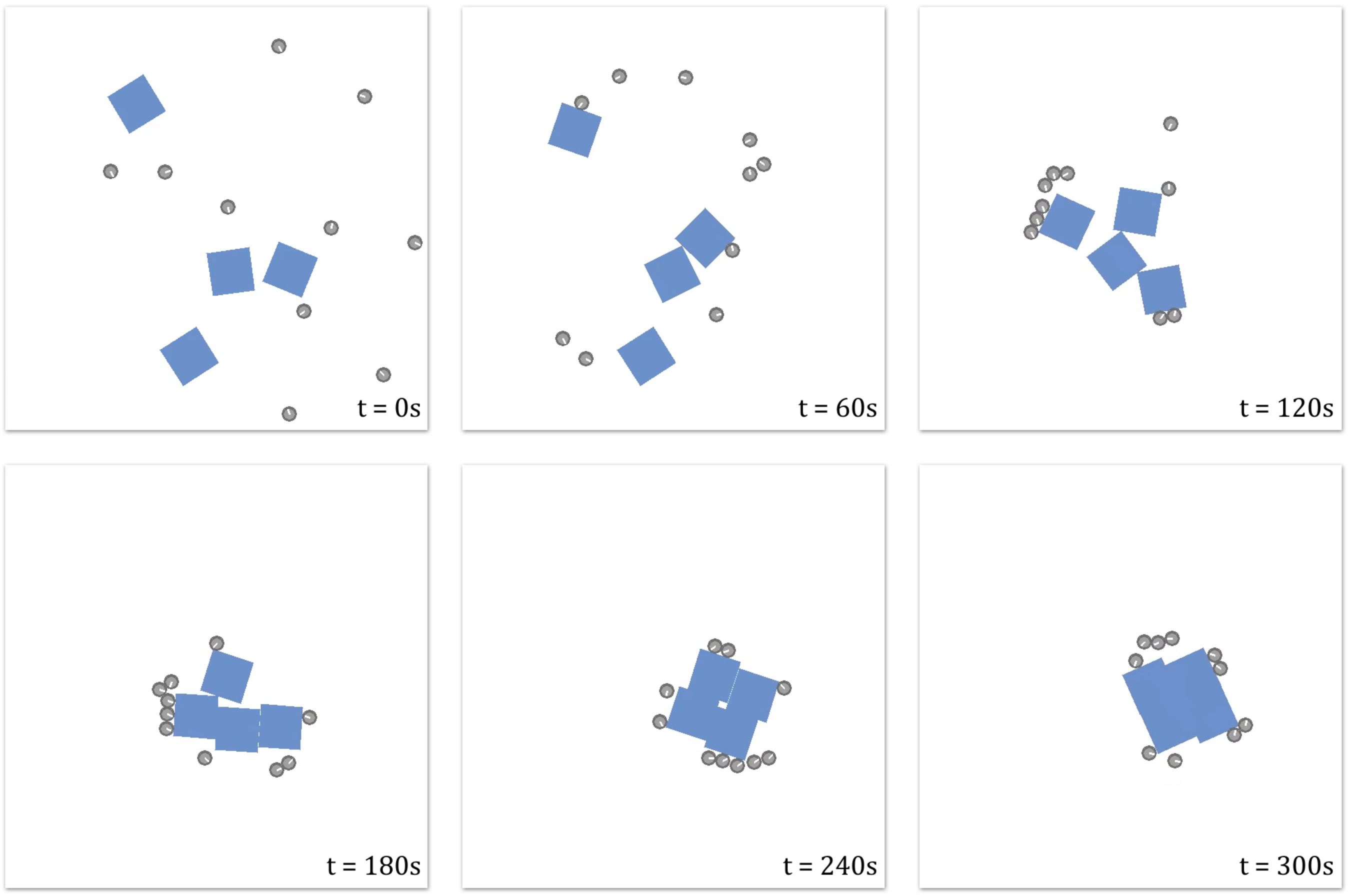


Figure 6: Example successful episode of the box assembly task (from [23])

The observation space for the box assembly task contains the following information:

1. The absolute position (x,y) of the current agent
2. The sin, cos of the absolute rotation of the current agent
3. For every neighboring agent:
    1. The distance between the current agent and the neighbor
    2. The sin and cos of the bearing angle to the neighbor
    3. The sin and cos of the neighbor's rotation relative to our rotation
4. For every neighboring box:
    1. The distance between the current agent and the box
    2. The sin and cos of the bearing angle to the box
    3. The sin and cos of the box's rotation relative to our rotation

5.2.2.3. Box Clustering Task

The task setup for box clustering is the same as for the box assembly task, except that each box is assigned a color. The goal is to move the boxes into an arrangement such that boxes of the same color are as close together as possible, while boxes of different colors are far away. The reward is the negative sum of pairwise distances between the boxes of each cluster plus the sum of pairwise distances of the center of mass of each cluster.

The observation space is the same as the observation space of the box assembly task, except that each cluster of objects is aggregated into a separate aggregation space.

{example clustering episode}

6. Results

In this section, we present the results of our experiments. We compare (1) the training performance of the different aggregation methods on various tasks, (2) aggregating different observation sets into a single vs multiple latent spaces (3) various variants of Bayesian aggregation (4) the training algorithms PPO and TRL.

In order to compare the performance of different variants, we consider the average reward per episode over the agent steps used during training. This way we can evaluate and compare the overall best results as well as the sample efficiency.

Each policy gradient training step consists of multiple gradient descent steps with a set of batches of trajectory rollouts generated using the previous policy. We evaluate the performance after each training step on a separate set of evaluation environments.

Unless mentioned otherwise, the following setup is used:

- Plotted is the median reward of the n runs at a specific training step, the error band is the 25th and 75th percentile of runs
- The activation function used after each layer is LeakyReLU
- For the Bayesian aggregation:
  - We only use the mean value μ_z as an output, not μ_z and σ_z^2
  - We use a single shared encoder for the value and confidence estimates
  - The a-priori estimate μ_{z_0} is learnable separately for each feature dimension
  - The variance of both the a-priori estimate as well as the encoded estimates are rectified using $\mathrm{softplus}$.
- Multiple aggregatable groups are aggregated into separate latent spaces
- The parameters of the policy and value function are not shared
- The training algorithm used is PPO

Due to time and resource constraints, we only use individually hyper-parameter optimized architectures on the multi-evader pursuit and rendezvous tasks, and use the same architecture for all aggregation methods on the other tasks.

6.1. Aggregation Method Results

We compare the performance of the different aggregation methods (Bayesian aggregation, mean aggregation, attentive aggregation) on multiple tasks.

We use the notation `64-agg-64` to describe the layer sizes of the neural network: The numbers before `agg` are the sequential layer sizes of the dense layers of the encoders of each aggregation group. The numbers after `agg` are the layer sizes in the decoder after the concatentation of the proprioceptive observations with the aggregated observations (compare Figure 1).

6.1.1. Multi-Evader Pursuit Task

Here, we consider the multi-evader pursuit task with 20 pursuers and 5 evaders on a torus. Figure 7 shows the results of the multi-evader pursuit task with different aggregation methdos with the same architecture used in [20] to be able to directly compare the results. The architecture is 64-agg-64 with the tanh activation function. With this architecture, the Bayesian aggregation performs best.

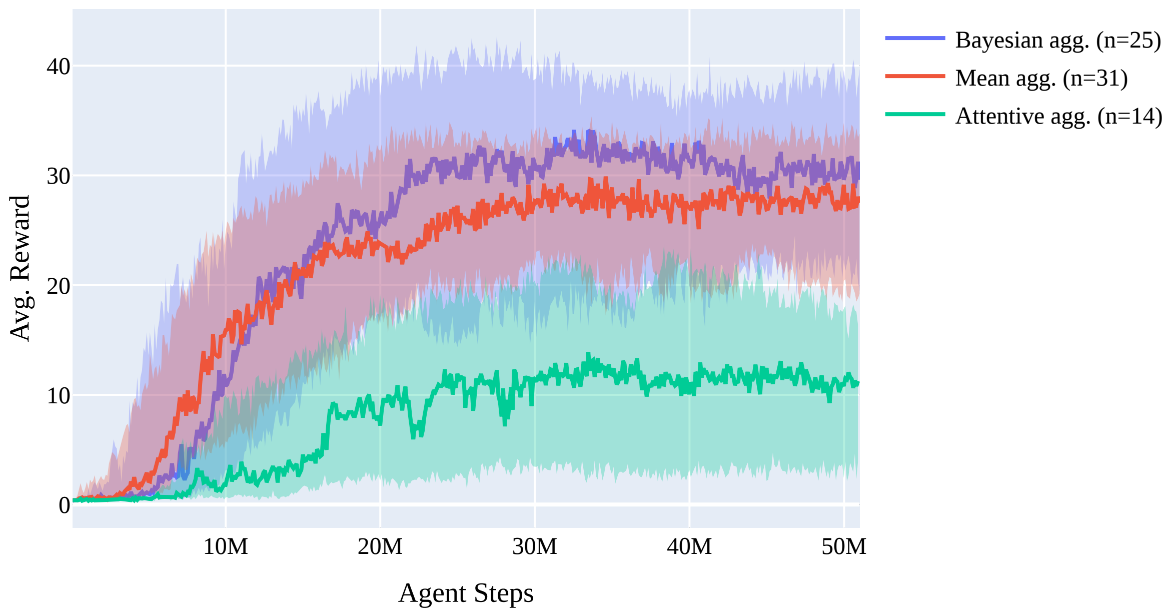


Figure 7: Results on the multi-evader pursuit task with the NN architecture adapted from [20] . The Bayesian aggregation performs best.

Figure 8 shows the results with neural network architectures that were separately hyper-parameter optimized for each aggregation method except max aggregation. The optimized architecture for the mean aggregation is `174-226-97-agg-96`, the same architecture is used for the max aggregation. The optimized architecture for the Bayesian aggregation is `120-60-agg-160`. The optimized architecture for the attentive aggregation is `72-agg-132-200`. All optimized architectures use the LeakyReLU activation functions. Note that the architecture optimized on the mean aggregation is the deepest with three hidden layers before the aggregation, while the optimized architecture on the attentive aggregation has multiple layers after the aggregation instead. With the hyper-parameter optimized architecture, the mean aggregation performs best. The results are still similar when using the same `120-60-agg-160` architecture for every aggregation method. These results of Figures 7, 8 indicate that the Bayesian aggregation outperforms the mean aggregation when the neural network is limited in size, but has no advantage when the neural network is sufficiently large and deep. The neural network seems to be able to implicitly learn to transform and weigh the information from the different observables, compensating the advantage of the additional structure of relevancy / certainty that is given in the Bayesian aggregation.

Figure 9 shows the results of the top third of runs. The performane of the mean and Bayesian aggregation is similar, indicating that the best runs are similar, but that the Bayesian aggregation has more runs that fail to achieve the peak performance.

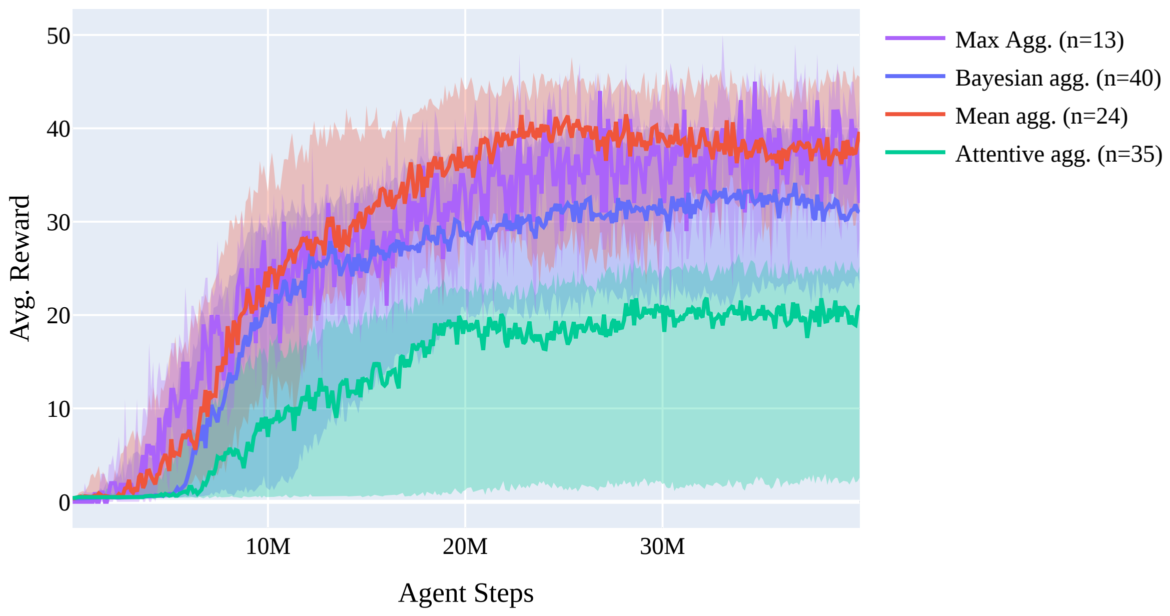


Figure 8: Results on the multi-evader pursuit task with the NN architecture hyper-parameter optimized for each aggregation method. The mean aggregation performs best.

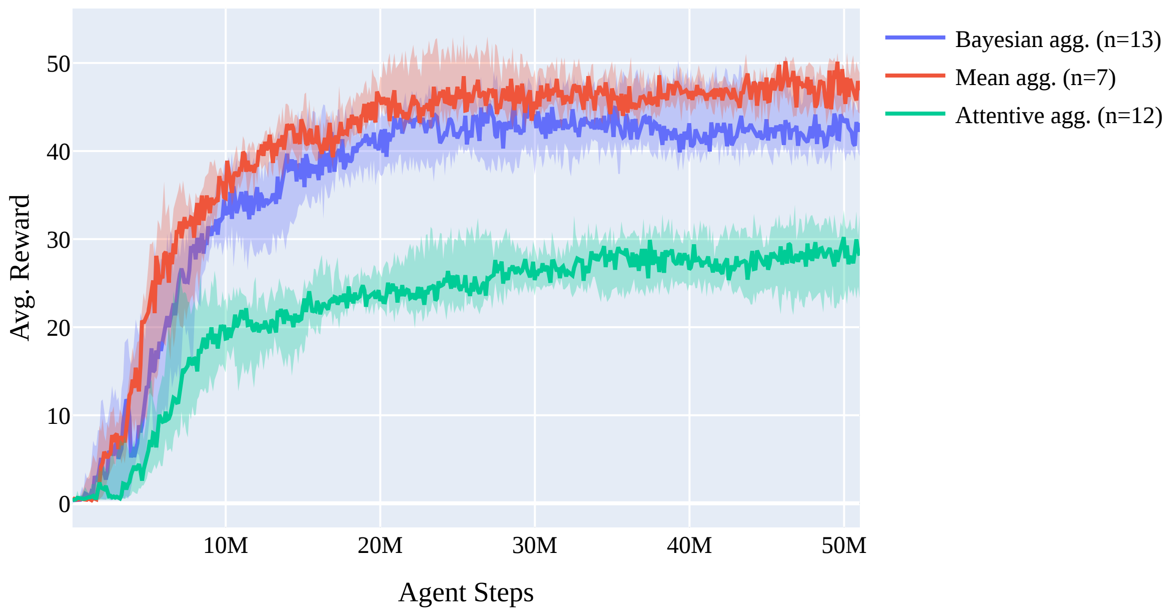


Figure 9: Like Figure 8 but only the top 1/3 of runs. This shows that the peak performance of the mean and the Bayesian aggregation is similar.

6.1.2. Single-Evader Pursuit Task

Figure 10 shows the results on the single-evader pursuit task with 10 pursuers and one evader. The neural network architecture is fixed at `120-60-agg-160` for all methods. All methods learn the task quickly, with the mean aggregation achieving the maximum performance slightly faster. This shows that the task is simpler than the multi-evader pursuit task, which is both due to the fact that there are fewer evaders and that the reward is more sparse (minimum-distance for single-evader vs count-catches for multi-evader).

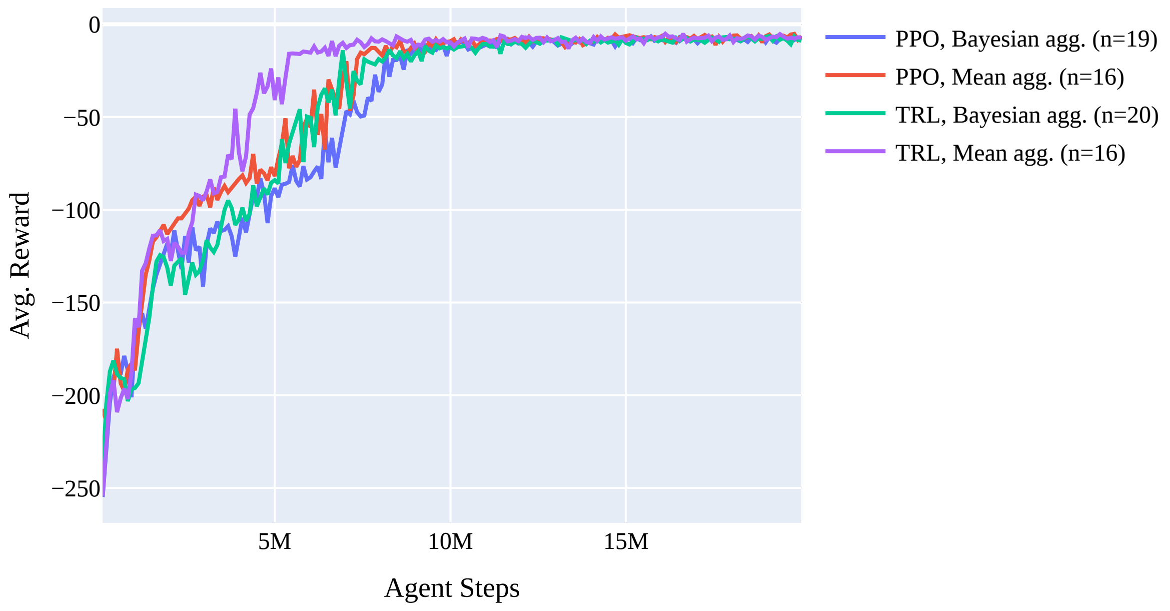


Figure 10: Results on Single Pursuit task. Architecture: 120-60-agg-160.

6.1.3. Rendezvous Task

[Figure 11](#) shows a comparison between mean and Bayesian aggregation on the rendezvous task with twenty agents in a two dimensional square world with walls. The medium architecture is the one optimized on the pursuit task (`120-60-agg-160`). The small architecture is the one used in [20] (`64-agg-64`). The optimized architecture was optimized on the rendezvous task directly: `146-120-agg-19-177-162` . All architectures and aggregation methods successfully solve the task. The aggregation method does not make any difference in the performance, but both the small and the medium architecture have a “drop” in training speed at around 2 million steps, while the optimized architecture smoothly learns the problem. The logarithmic scale graph to the right shows that while the small and optimized architecture both reach the same final score, the medium architecture never reaches the same score. This might be because both the small and the optimized architecture have a bottleneck layer after the aggregation, forcing the neural network to simplify the strategy.

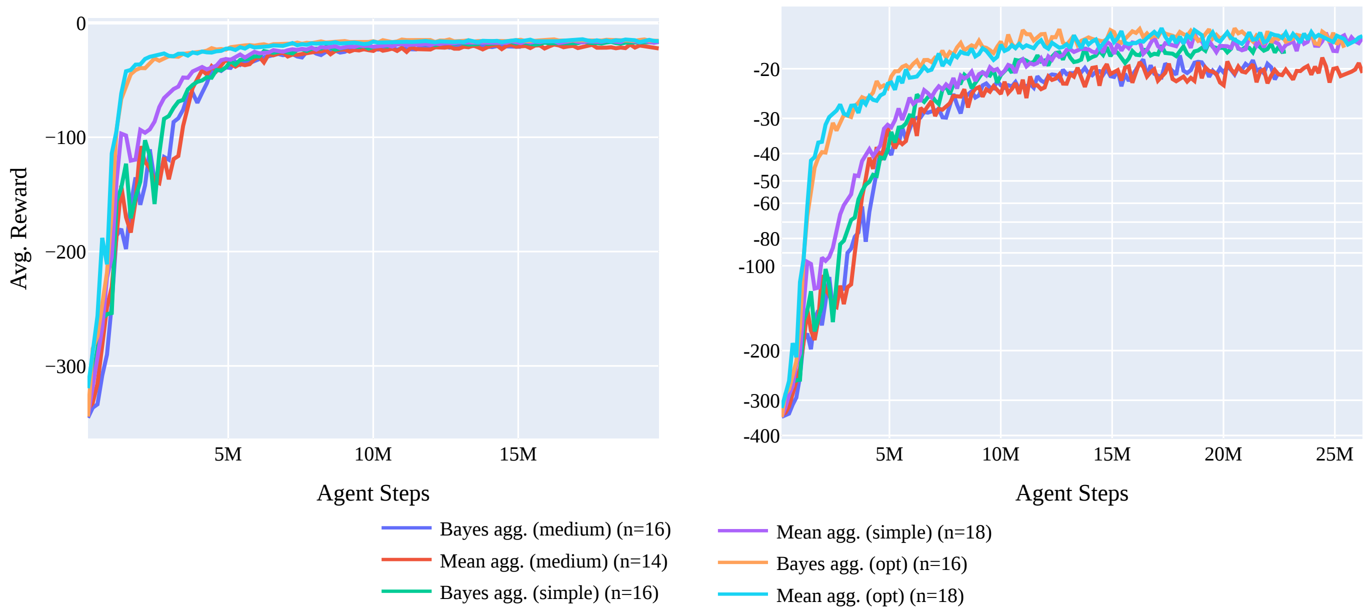


Figure 11: Results on the rendezvous task. The results barely differ between the mean and Bayesian aggregation, but the size of the policy architecture makes a difference. In the logarithmic view on the right it can be seen that the medium architecture does not reach the same final performance as the other two architectures.

6.1.4. Assembly Task

Figure 12 shows the results on the assembly task with ten agents and four boxes. The three aggregation methods perform very similar, with the attentive aggregation learning the task slightly quicker.

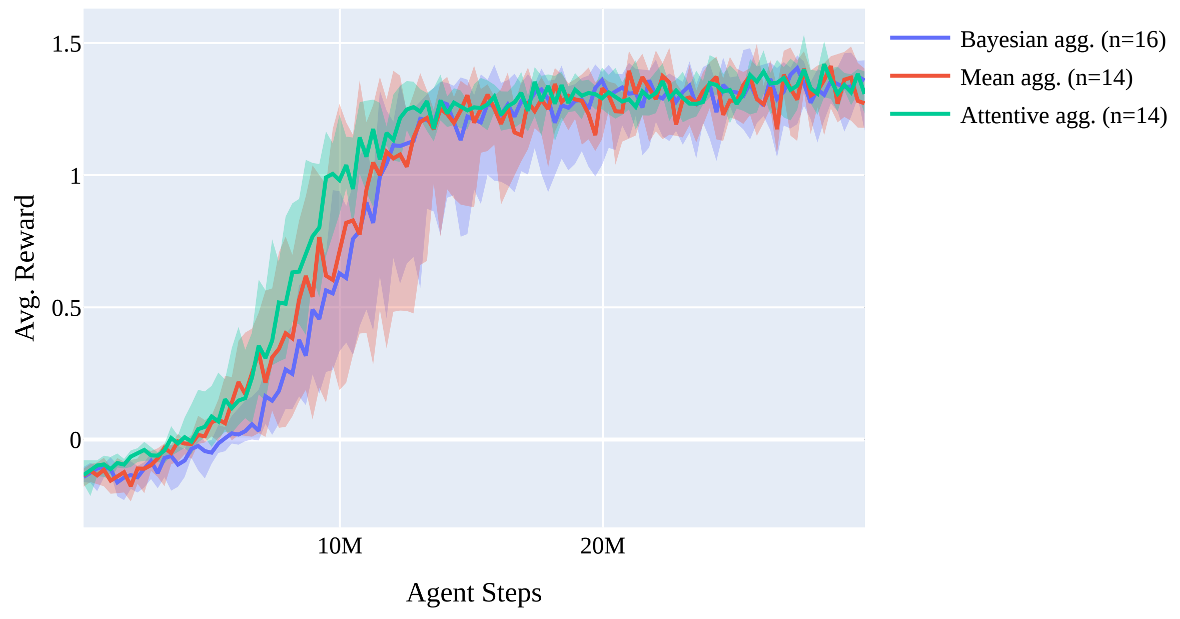


Figure 12: Results on the assembly task.

6.1.5. Clustering Task With Two Clusters

Figure 13 shows the results on the clustering task with four boxes split into two clusters.

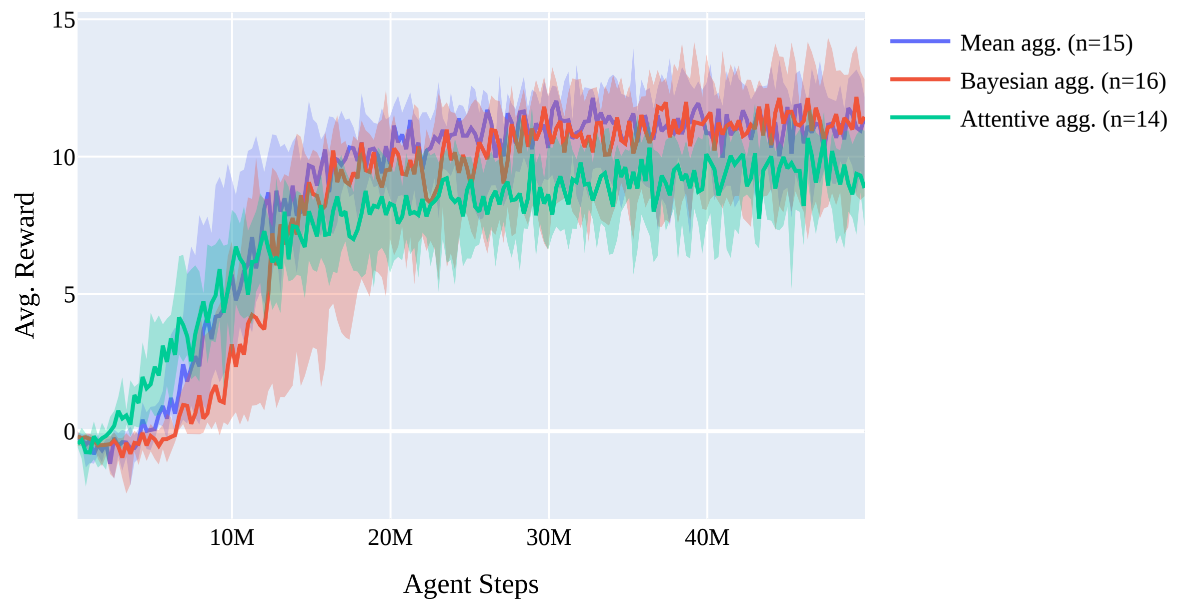


Figure 13: Clustering2 results

6.1.6. Clustering Task With Three Clusters

Doesn’t work :(

6.2. Learning Algorithm Comparison (PPO vs PG-TRL)

In the following, we show some results of the trust region layers policy gradient (TRL) training method (see Section 2.2.2) compared to PPO.

Figure 14 shows the learning algorithm comparison on the multi-evader pursuit task. The architecture are the ones hyper-parameter optimized on PPO on each of the aggregation methods. TRL seems to show significantly improved training performance for the Bayesian aggregation and similar performance for the mean aggregation.

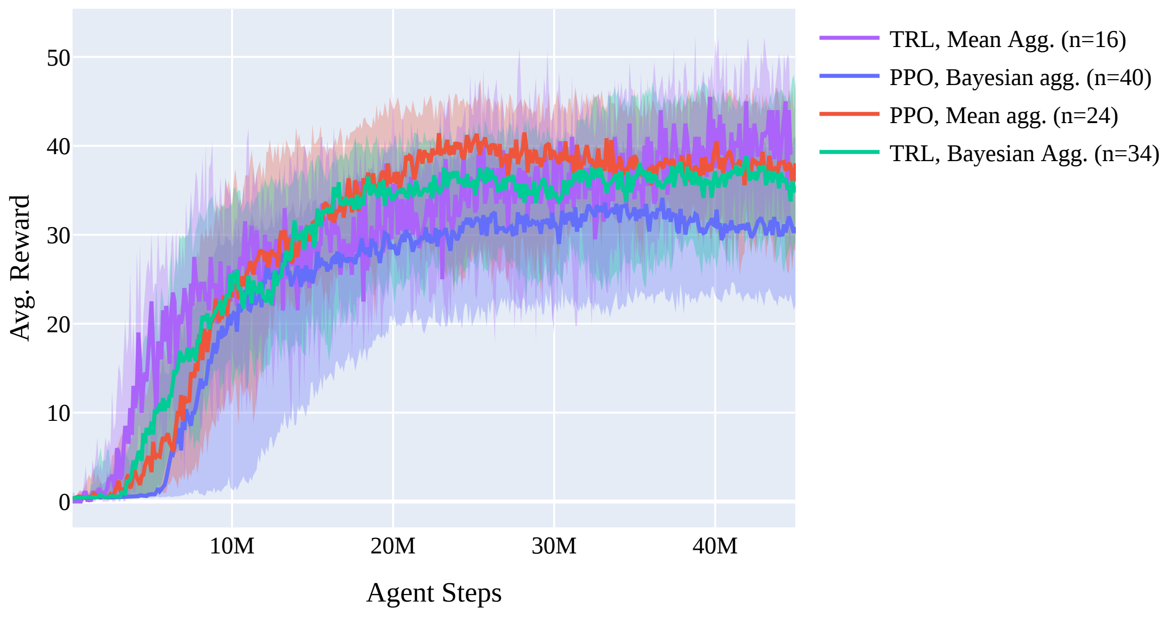


Figure 14: TRL vs PPO learning algorithms on the multi-evader pursuit task. For Bayesian aggregation, the training is faster and the end result better. For mean aggregation, the results are fairly similar.

Figure 15 shows the comparison on the single-evader pursuit task. Here, the performance of TRL is better than PPO on both the mean and the Bayesian aggregation.

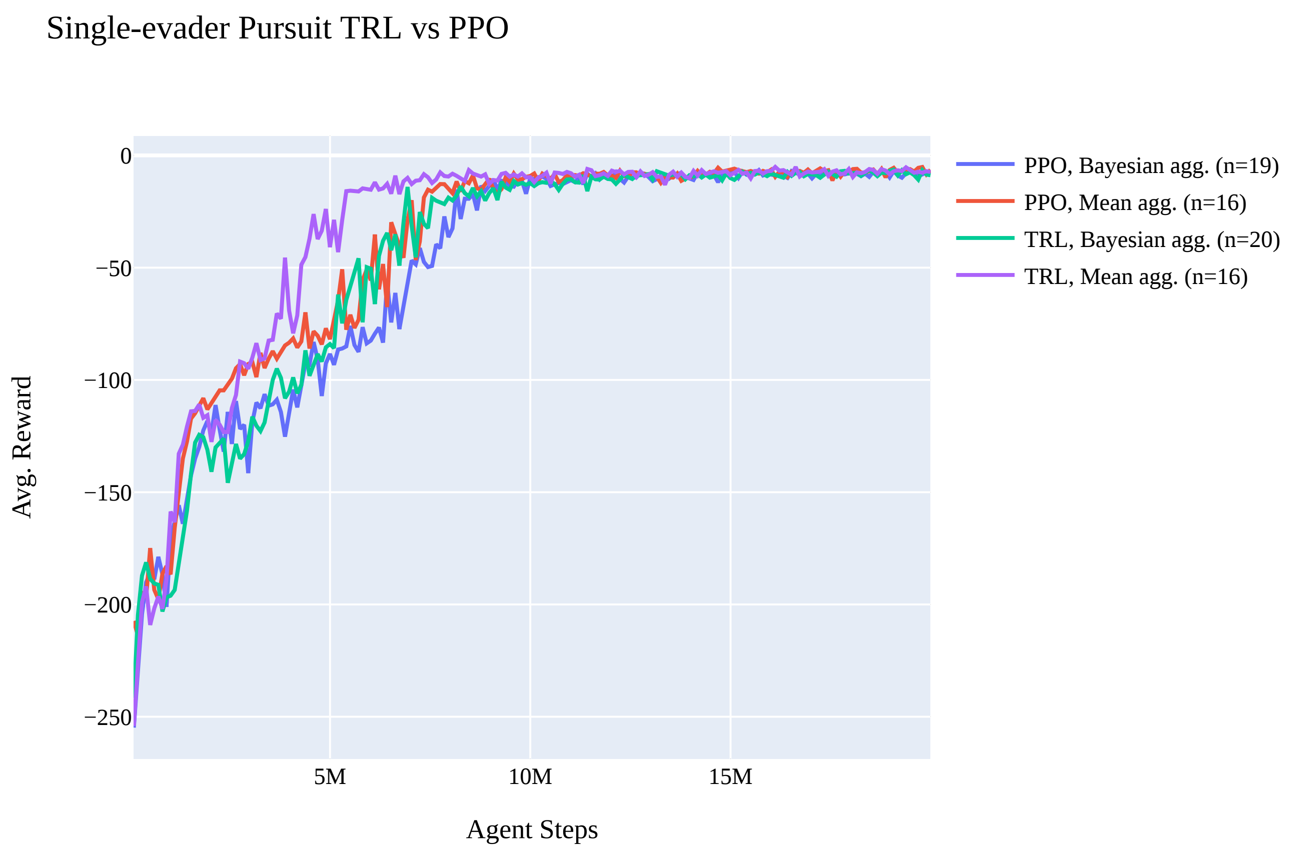


Figure 15: TRL vs PPO learning algorithms on single-evader pursuit task. The performance of TRL is better for both the mean and the Bayesian aggregation.

Figure 16 shows the comparison on the assembly task. All aggregation methods perform the same, except the Bayesian aggregation with TRL, which is worse.

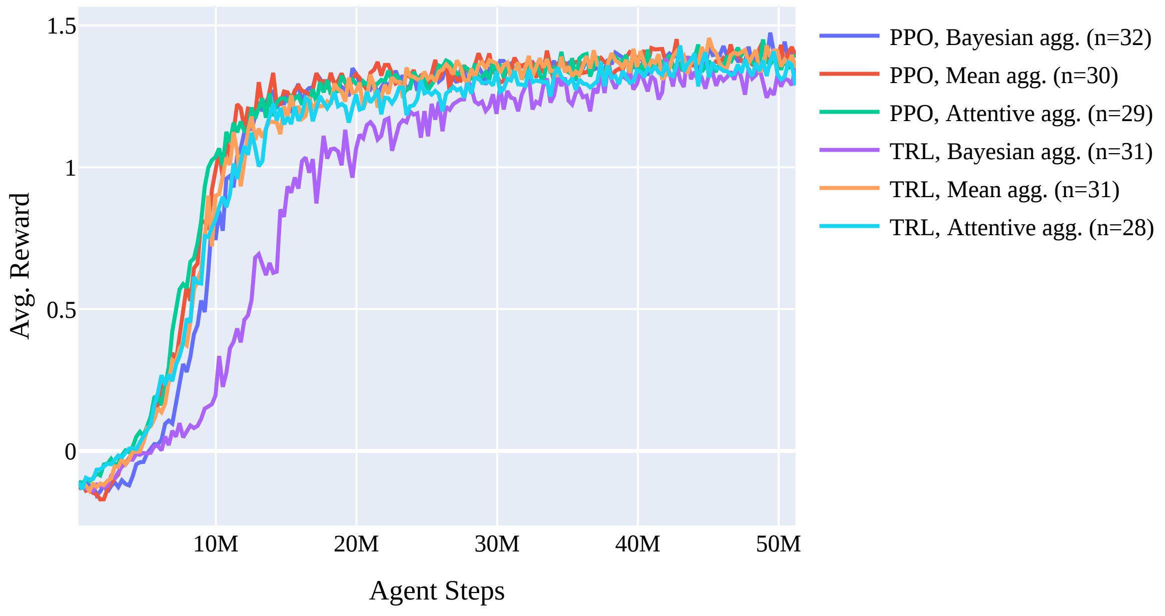


Figure 16: TRL vs PPO learning algorithms on the assembly task. The training performance is the same for all aggregation methods, except that the Bayesian aggregation performs worse with TRL.

In summary, TRL seems to perform the same or better in most cases, with the Bayesian aggregation on the assembly task being an outlier.

6.3. Same Space vs Separate Space Aggregation

For tasks where we have multiple aggregation groups, we can also aggregate all observables into the same latent space instead of separate ones. This means that instead of each aggregation space containing the information of those aggregatables, the single aggregation space must contain the information of all observables as it pertains to the current agent. The potential advantage is that the policy can share more parameters and thus be more sample efficient. It can also scale to a larger number of categories of objects (aggregation groups). Figure 17 shows a schematic comparison between the two methods.

In the other experiments we alwas use separate spaces. Figure 18 shows the results of aggregating into a single space vs. into separate spaces for the multi-evader pursuit and assembly tasks. In the case of the multi-evader pursuit task same-space aggregation means that both the neighboring pursuers as well as all the evaders are aggregated into one space. For the assembly task, both the neighboring agents as well as the boxes are aggregated into one space. The separate-space aggregation performs better in both tasks, with a higher margin in the multi-evader task.

In summary, while the same-space aggregation is promising by in theory being able to scale to more complex environment, it performs worse for our tasks. This might be due to the fact that learning different encoder networks to output information into the same latent space is hard. An alternative explanation might be that our experiments only having few aggregation groups so the value of parameter sharing is low, or due to the lack of experiments with more different hyper-parameters.

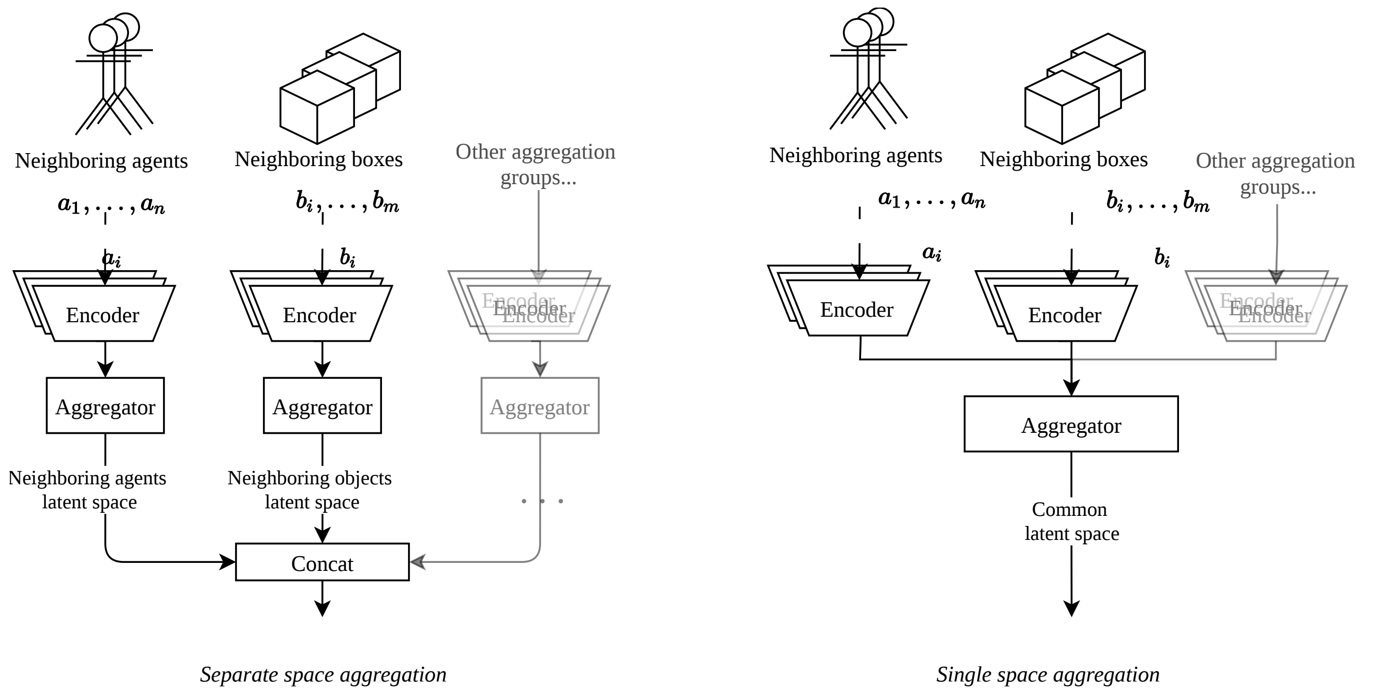


Figure 17: Schematic comparison of separate space vs. same-space aggregation. In same-space aggregation the encoders are still separate, but the latent space is shared. The same-space aggregation shares more parameters and can scale better to more aggregation groups.

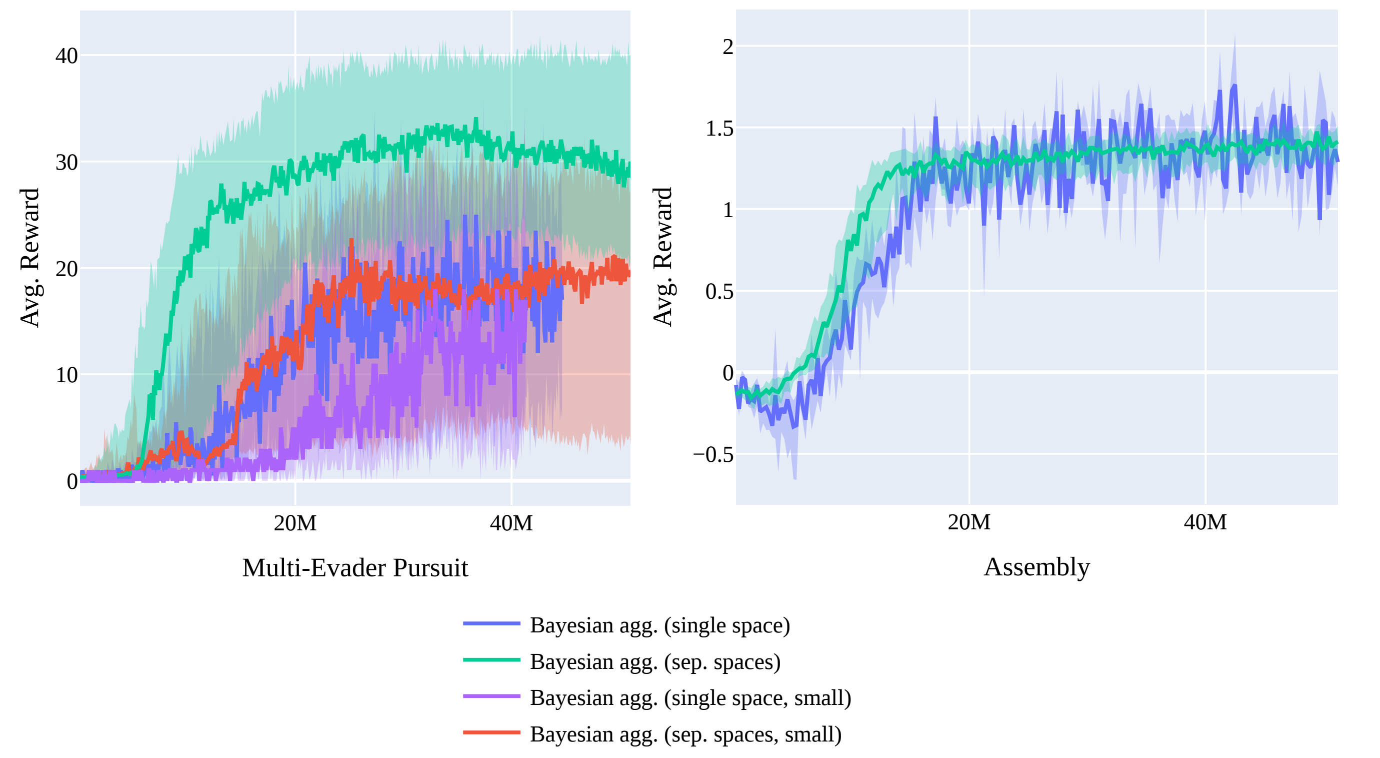


Figure 18: Separate vs same-space aggregation on the multi-evader pursuit and assembly tasks. The same-space aggregation performs worse in both tasks.

6.4. Bayesian Aggregation Variants

The following shows results for some variants of the Bayesian aggregation.

6.4.1. Separate vs Common Encoder

As described in Section 4.1.2, we can either have a shared encoder to predict the mean and variance of each sample in each aggregation space by making the last layer of the encoder have two outputs for each feature, or have two fully separate networks (enc_r and enc_σ). In our experiments, using one common encoder with two outputs generally performs better.

6.4.2. Using the Aggregated Variance or Only the Mean

In the other experiments with Bayesian aggregation, we only use the predicted mean of the Gaussian distribution as an input to the decoder:

$$e_{k\to G} = \mu_z$$

Since the Bayesian aggregation also gives us a full a-posteriori Gaussian distribution, we also have an estimate of the variance for each feature in the latent space that is computed from the apriori variance conditioned on each seen latent space sample. We can feed this variance to the decoder by concatenating it with the mean predictions in the hope that the neural network is able to use this additional information:

$$e_{k\to G} = (\mu_z, \sigma_z^2)$$

The results of applying this method to the multi-evader pursuit are seen in Figure 19. The neural network architecture is the same as for the other experiments with Bayesian aggregation on multi-evader pursuit (`120-60-agg-160`). Including the variance in the decoder inputs decreases the performance.

The decreasing performance could be a result of the increased dimension of the decoder inputs. Adding the variance inputs doubles the number of values the decoder has to process and learn from. Since the structure of the encoded values and variances can not known beforehand to the decoder, it has to learn to interpret more information than when receiving just the mean values. The added variance inputs should give the decoder the ability to understand the confidence of each of the value predictions and weigh them accordingly, but the added complexity seems to make it not worth it.

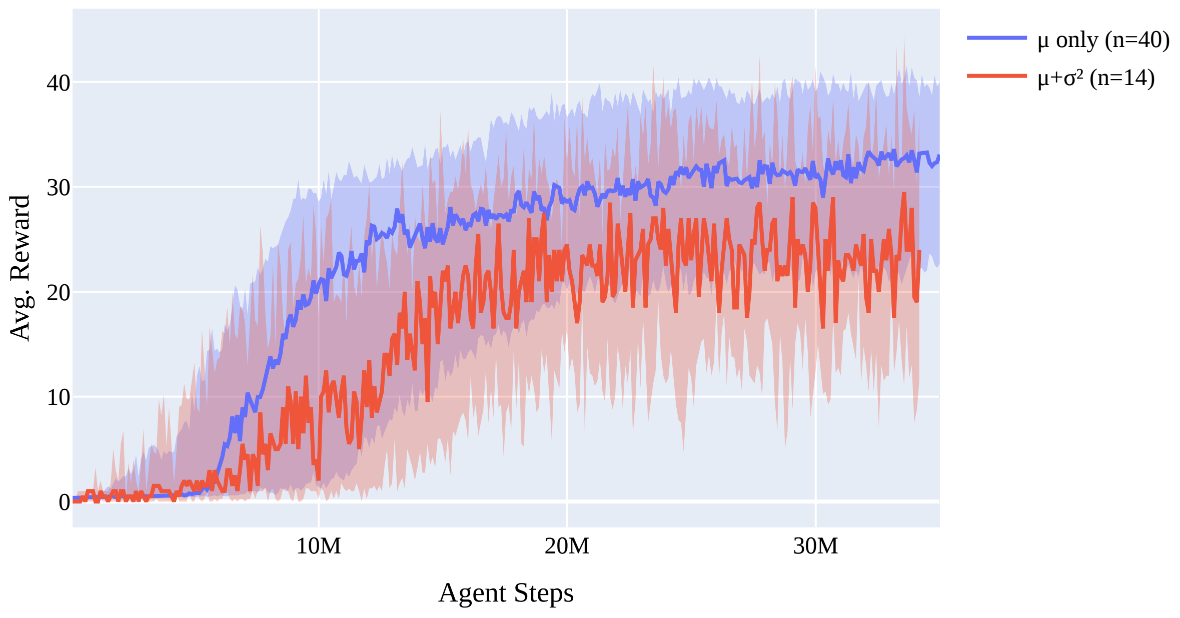


Figure 19: Results of Bayesian aggregation on the multi-evader pursuit task, depending on whether the variance is also fed into the decoder or only the mean.

7. Conclusion and Future Work

7.1. Conclusion

We have made a comprehensive comparison between the performance of mean aggregation, Bayesian aggregation, and attentive aggregation to collect a varying number of observations on a set of different deep reinforcement learning tasks. We have observed that there is no clear advantage of one of the methods over the others, with the results differing strongly between the different tasks.

We have also shown the results of a few variants of the Bayesian aggregation and concluded that encoding the variance with the same encoder as the estimate, aggregating into separate latent spaces and not using the aggregated variance as an input to the decoder achieves the best results.

Finally, we have applied a new training method (trust region layers) to multi-agent reinforcement learning and compared it to the commonly used PPO. The results indicate that TRL is usually at least as good as PPO and in some cases outperforms it.

7.2. Future Work

There are many avenues for future work in this area.

All of our experiments used global visibility due to the noisy nature of limited local visibility making it harder to make any strong conclusions. Since the local visibility case also increases the uncertainty of each observation though, it might be a case where Bayesian aggregation performs better. Future work should include experiments that have a larger environment with observability limited by range or by obstacles.

Even though we show in our experiments that Bayesian aggregation into the same latent space is worse than aggregating into separate spaces, there might be potential there to be able to scale the number of aggregation groups with more hyperparameter tuning or by introducing a two-stage encoder where the first encoder is separate by aggregation group and the second encoder is shared, then aggregating the output of the second encoder into the same space.

We also only considered tasks with implicit communication - the agents had to infer the intent of the other agents purely by their actions. There is related work that adds explicit communication between agents that is learned together with the policy. This is usually implemented as another action output that is written directly into the observation of the other agents instead of affecting the world. Explicit communication architectures may be able to handle some tasks better than those with implicit communication, but they are often only applicable to environments with exactly two agents. For more agents, the performance of explicit communication architectures may be affected by the aggregation methods used and thus might be a use case for Bayesian aggregation.

Bibliography

1. **Multi-Agent Systems for Search and Rescue Applications**
Daniel S Drew
Current Robotics Reports (2021-06) https://doi.org/10.1007/s43154-021-00048-3

2. **Occlusion-Based Coordination Protocol Design for Autonomous Robotic Shepherding Tasks**
Junyan Hu, Ali Emre Turgut, Tomáš Krajník, Barry Lennox, Farshad Arvin
IEEE Transactions on Cognitive and Developmental Systems (2020)
https://ieeexplore.ieee.org/abstract/document/9173524

3. **Drone shows: Creative potential and best practices**
Markus Waibel, Bill Keays, Federico Augugliaro
ETH Zurich (2017) https://www.research-collection.ethz.ch/handle/20.500.11850/125498

4. **Collaborating miniature drones for surveillance and reconnaissance**
Axel Bürkle
Unmanned/Unattended Sensors and Sensor Networks VI (2009-09)
https://www.spiedigitallibrary.org/conference-proceedings-of-spie/7480/74800H/Collaborating-miniature-drones-for-surveillance-and-reconnaissance/10.1117/12.830408.short

5. **Drone Swarms**
Andrew W Sanders
US Army School for Advanced Military Studies Fort Leavenworth United States (2017-05)
https://apps.dtic.mil/sti/citations/AD1039921

6. **Controlling swarms of medical nanorobots using CPPSO on a GPU**
Davide Ceraso, Giandomenico Spezzano
2016 International Conference on High Performance Computing Simulation (HPCS) (2016-07)
https://ieeexplore.ieee.org/document/7568316

7. **Classification of global catastrophic risks connected with artificial intelligence**
Alexey Turchin, David Denkenberger
AI & SOCIETY (2020-03) https://doi.org/10.1007/s00146-018-0845-5

8. **Learning Dexterous In-Hand Manipulation**
OpenAI, Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, … Wojciech Zaremba
arXiv:1808.00177 [cs, stat] (2019-01) http://arxiv.org/abs/1808.00177

9. **Emergent Tool Use From Multi-Agent Autocurricula**
Bowen Baker, Ingmar Kanitscheider, Todor Markov, Yi Wu, Glenn Powell, Bob McGrew, Igor Mordatch
arXiv:1909.07528 [cs, stat] (2020-02) http://arxiv.org/abs/1909.07528

10. **Trust Region Policy Optimization**
John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, Philipp Moritz
International Conference on Machine Learning (2015-06)
http://proceedings.mlr.press/v37/schulman15.html

11. **Proximal Policy Optimization Algorithms**
John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov

arXiv:1707.06347 [cs] (2017-08) http://arxiv.org/abs/1707.06347

12. **Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor**
Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, Sergey Levine
arXiv:1801.01290 [cs, stat] (2018-08) http://arxiv.org/abs/1801.01290

13. **High-Dimensional Continuous Control Using Generalized Advantage Estimation**
John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, Pieter Abbeel
arXiv:1506.02438 [cs] (2018-10) http://arxiv.org/abs/1506.02438

14. **Stable Baselines3**
Antonin Raffin, Ashley Hill, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, Noah Dormann
GitHub repository (2019)

15. **PPO — Stable Baselines3 1.1.0a11 documentation**
Antonin Raffin, Ashley Hill, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, Noah Dormann
stable-baselines3.readthedocs.io (2020) https://stable-baselines3.readthedocs.io/en/master/modules/ppo.html#results

16. **Differentiable Trust Region Layers for Deep Reinforcement Learning**
Fabian Otto, Philipp Becker, Vien Anh Ngo, Hanna Carolin Maria Ziesche, Gerhard Neumann
(2020-09) https://openreview.net/forum?id=qYZD-AO1Vn

17. **On Information and Sufficiency**
S Kullback, RA Leibler
The Annals of Mathematical Statistics (1951-03) http://projecteuclid.org/euclid.aoms/1177729694

18. **Optimal Transport: Old and New**
Cédric Villani
Springer-Verlag (2009) https://www.springer.com/gp/book/9783540710493

19. **A Concise Introduction to Decentralized POMDPs**
Frans A Oliehoek, Christopher Amato
Springer International Publishing (2016) https://www.springer.com/gp/book/9783319289274

20. **Deep Reinforcement Learning for Swarm Systems**
Maximilian Hüttenrauch, Adrian Šošić, Gerhard Neumann
Journal of Machine Learning Research (2019) http://jmlr.org/papers/v20/18-476.html

21. **Inverse Reinforcement Learning in Swarm Systems**
Adrian Šošić, Wasiur R KhudaBukhsh, Abdelhak M Zoubir, Heinz Koeppl
Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems (2017-05) https://dl.acm.org/doi/10.5555/3091125.3091320

22. **Multi-agent actor-critic for mixed cooperative-competitive environments**
Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, Igor Mordatch
Proceedings of the 31st International Conference on Neural Information Processing Systems (2017-12) https://dl.acm.org/doi/10.5555/3295222.3295385

23. **Using M-Embeddings to Learn Control Strategies for Robot Swarms**
Gregor HW Gebhardt, Maximilian Hüttenrauch, G Neumann
www.semanticscholar.org (2019) https://www.semanticscholar.org/paper/Using-M-Embeddings-to-Learn-Control-Strategies-for-Gebhardt-H%C3%BCttenrauch/9f550815f8858e7c4c8aef23665fa5817884f1b3

24. **Mean field games**
Jean-Michel Lasry, Pierre-Louis Lions
Japanese Journal of Mathematics (2007-03) https://doi.org/10.1007/s11537-007-0657-8

25. **Mean Field Multi-Agent Reinforcement Learning**
Yaodong Yang, Rui Luo, Minne Li, Ming Zhou, Weinan Zhang, Jun Wang
International Conference on Machine Learning (2018-07)
http://proceedings.mlr.press/v80/yang18d.html

26. **Mean Field Deep Reinforcement Learning for Fair and Efficient UAV Control**
Dezhi Chen, Qi Qi, Zirui Zhuang, Jingyu Wang, Jianxin Liao, Zhu Han
IEEE Internet of Things Journal (2021-01)
https://ieeexplore.ieee.org/abstract/document/9137257

27. **Emergence of Grounded Compositional Language in Multi-Agent Populations**
Igor Mordatch, Pieter Abbeel
arXiv:1703.04908 [cs] (2018-07) http://arxiv.org/abs/1703.04908

28. **Gaussian Processes in Machine Learning**
Carl Edward Rasmussen
Advanced Lectures on Machine Learning: ML Summer Schools 2003, Canberra, Australia, February 2 - 14, 2003, Tübingen, Germany, August 4 - 16, 2003, Revised Lectures (2004)
https://doi.org/10.1007/978-3-540-28650-9_4

29. **Bayesian Context Aggregation for Neural Processes**
Michael Volpp, Fabian Flürenbrock, Lukas Grossberger, Christian Daniel, Gerhard Neumann
(2020-09) https://openreview.net/forum?id=ufZN2-aehFa

30. **Pattern Recognition and Machine Learning**
Christopher Bishop
Springer-Verlag (2006) https://www.springer.com/gp/book/9780387310732

31. **Attention is All you Need**
Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, Illia Polosukhin
Advances in Neural Information Processing Systems (2017)
https://papers.nips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html

32. **Actor-Attention-Critic for Multi-Agent Reinforcement Learning**
Shariq Iqbal, Fei Sha
International Conference on Machine Learning (2019-05)
http://proceedings.mlr.press/v97/iqbal19a.html

33. **Attentive Relational State Representation in Decentralized Multiagent Reinforcement Learning**
Xiangyu Liu, Ying Tan
IEEE Transactions on Cybernetics (2020) https://ieeexplore.ieee.org/document/9049415

34. **Multi-Agent Reinforcement Learning: A Selective Overview of Theories and Algorithms**
Kaiqing Zhang, Zhuoran Yang, Tamer Başar
arXiv:1911.10635 [cs, stat] (2021-04) http://arxiv.org/abs/1911.10635

35. **Multi-Agent Reinforcement Learning: A Review of Challenges and Applications**
Lorenzo Canese, Gian Carlo Cardarilli, Luca Di Nunzio, Rocco Fazzolari, Daniele Giardino, Marco Re, Sergio Spanò

Applied Sciences (2021-01) https://www.mdpi.com/2076-3417/11/11/4948

36. **Multi-Agent Reinforcement Learning via Double Averaging Primal-Dual Optimization**
Hoi-To Wai, Zhuoran Yang, Zhaoran Wang, Mingyi Hong
arXiv:1806.00877 [cs, math, stat] (2019-01) http://arxiv.org/abs/1806.00877

37. **Q D-learning: A collaborative distributed strategy for multi-agent reinforcement learning through {\rm Consensus} + {\rm Innovations}**
Soummya Kar, José MF Moura, HVincent Poor
IEEE Transactions on Signal Processing (2013)

38. **Fully Decentralized Multi-Agent Reinforcement Learning with Networked Agents**
Kaiqing Zhang, Zhuoran Yang, Han Liu, Tong Zhang, Tamer Basar
International Conference on Machine Learning (2018-07)
http://proceedings.mlr.press/v80/zhang18n.html

39. **Counterfactual Multi-Agent Policy Gradients**
Jakob Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, Shimon Whiteson
arXiv:1705.08926 [cs] (2017-12) http://arxiv.org/abs/1705.08926

40. **Cooperative Multi-agent Control Using Deep Reinforcement Learning**
Jayesh K Gupta, Maxim Egorov, Mykel Kochenderfer
Autonomous Agents and Multiagent Systems (2017)
https://link.springer.com/chapter/10.1007/978-3-319-71682-4_5

41. **Decentralized POMDPs**
Frans A Oliehoek
Reinforcement Learning: State-of-the-Art (2012) https://doi.org/10.1007/978-3-642-27645-3_15

42. **Learning to Communicate with Deep Multi-Agent Reinforcement Learning**
Jakob Foerster, Ioannis Alexandros Assael, Nando de Freitas, Shimon Whiteson
Advances in Neural Information Processing Systems (2016)
https://proceedings.neurips.cc/paper/2016/hash/c7635bfd99248a2cdef8249ef7bfbef4-Abstract.html

43. **Hysteretic Q-learning : An algorithm for Decentralized Reinforcement Learning in Cooperative Multi-Agent Teams**
Laetitia Matignon, Guillaume J Laurent, Nadine Le Fort-Piat
2007 IEEE/RSJ International Conference on Intelligent Robots and Systems (2007-10)
https://ieeexplore.ieee.org/document/4399095

44. **MAgent: A Many-Agent Reinforcement Learning Platform for Artificial Collective Intelligence**
Lianmin Zheng, Jiacheng Yang, Han Cai, Ming Zhou, Weinan Zhang, Jun Wang, Yong Yu
Proceedings of the AAAI Conference on Artificial Intelligence (2018-04)
https://ojs.aaai.org/index.php/AAAI/article/view/11371

45. **Mastering the game of Go without human knowledge**
David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, … Demis Hassabis
Nature (2017-10) https://www.nature.com/articles/nature24270

46. **PettingZoo: Gym for Multi-Agent Reinforcement Learning**
JK Terry, Benjamin Black, Nathaniel Grammel, Mario Jayakumar, Ananth Hari, Ryan Sullivan, Luis Santos, Rodrigo Perez, Caroline Horsch, Clemens Dieffendahl, … Praveen Ravi

arXiv:2009.14471 [cs, stat] (2021-06) http://arxiv.org/abs/2009.14471

47. **Optuna: A Next-generation Hyperparameter Optimization Framework**
Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, Masanori Koyama
Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (2019-07) https://doi.org/10.1145/3292500.3330701

48. **Formation constrained multi-agent control**
M Egerstedt, Xiaoming Hu
IEEE Transactions on Robotics and Automation (2001-12)
https://ieeexplore.ieee.org/document/976029

49. **Cooperative pursuit with Voronoi partitions**
Zhengyuan Zhou, Wei Zhang, Jerry Ding, Haomiao Huang, Dušan M Stipanović, Claire J Tomlin
Automatica (2016-10)
https://www.sciencedirect.com/science/article/pii/S0005109816301911