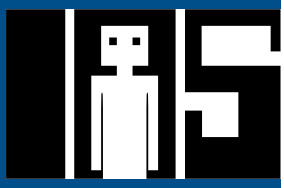
Reinforcement Learning in Decentralized Multi-Goal Multi-Agent Settings

Reinforcement learning in dezentralisierten Szenarien mit mehreren Zielen und Akteuren
Master thesis by David Rother
Date of submission: May 7, 2021

1. Review: M.Sc. Fabio Muratore
2. Review: Prof. Dr. Jan Peters
Darmstadt

TECHNISCHE
UNIVERSITÄT
DARMSTADT



Erklärung zur Abschlussarbeit gemäß §22 Abs. 7 und §23 Abs. 7 APB der TU Darmstadt

Hiermit versichere ich, David Rother, die vorliegende Masterarbeit ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass im Fall eines Plagiats (§38 Abs. 2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei der abgegebenen Thesis stimmen die schriftliche und die zur Archivierung eingereichte elektronische Fassung gemäß §23 Abs. 7 APB überein.

Bei einer Thesis des Fachbereichs Architektur entspricht die eingereichte elektronische Fassung dem vorgestellten Modell und den vorgelegten Plänen.

Darmstadt, 7. Mai 2021

David Rother

Abstract

Multi-Agent Reinforcement Learning is a trending research topic in many fields such as robotics, distributed control and economics. Many real-world Multi-Agent tasks are to complex to be solved by simple rule based systems. Instead, multi-agent reinforcement learning aims to find solutions through interaction with the environment it is placed in. However, with AI being increasingly interwoven in our lives, research needs to not only find good solutions for the one using it, but do so in a socially acceptable manner. The contribution of this thesis to support efforts in this direction are threefold. First, we create a novel scenario, where a test of social skill is singled out to measure how well an algorithm does in that regard. Second, we extend the policy gradient method PPO to incorporate a social component in its gradient by using the theory of mind concept from psychology, developing a new algorithm DM3 that takes ideas from the related algorithm CM3. To compute influence on other agents, an agent takes their perspective and imagines the impact of its actions by modeling their thought process as its own, leading to a social understanding of the situation. The last contribution is the incorporation of additional metrics beyond the sum of rewards, the utilitarian measure. Using different metrics, invented in the field of social economics, allows one to gauge different properties of the resulting behavior when evaluating cohabitating artificial agents. We can show that the newly created scenario distinguishes between social and non-social behavior through examination of the expected social behavior. Our algorithm DM3 manages to learn in a social manner, whereas PPO does expectedly not. However, the gained capabilities come at the cost of local performance power because the algorithm gets stuck in worse local optima. The additional metrics highlight differences in social capabilities and help identify where an algorithm performs poorly because of worse local optima and where the algorithm does not learn social behavior.

Zusammenfassung

Reinforcement Learning mit mehreren Akteuren ist ein aktuelles Forschungsthema in Robotik, verteilten Steuerungssystemen, Wirtschaft und anderen Forschungsfeldern. Viele Aufgaben, die mehrere Akteure involvieren, sind zu komplex um sie mit einem Regel basierten System zu lösen. Stattdessen findet Reinforcement Learning Lösungen durch Lernen. Da Künstliche Intelligenz (KI) immer weiter in unser Leben eindringt ist es notwendig, dass die KI der Zukunft nicht nur ihre Aufgabe löst, sondern dies auch mit einem sozialen Bewusstsein macht. In dieser Thesis unterstützen wir Forschungsanstrengungen in dieser Richtung auf drei Art und Weisen. Wir stellen eine neue Lernumgebung für mehrere Akteure vor, in der wir explizit auf soziales Verhalten testen und dabei andere Einflussfaktoren möglichst eliminieren. Darüber hinaus erweiteren wir die policy gradient Methode PPO und inkorporieren soziale Aufmerksamkeit in das Lernverfahren. Dabei greifen wir auf die Theory of Mind aus der Psychologie zurück, um unser Vorgehen zu motivieren, und entwickeln einen neuen Algorithmus DM3, der Ideen von dem Algorithmus CM3 übernimmt. Um den Einfluss auf andere Akteure zu berechnen, nimmt ein Akteur die Perspektive eines anderen ein und modelliert den Gedankenprozess seines Gegenübers als würde er ihn selbst erleben. Dies erlaubt ihm einzuschätzen, welche Auswirkungen die eigenen Aktionen auf andere haben werden und führt dadurch zu einem sozialem Verständnis der Situation. Als letztes erweitern wir die Evaluation um bekannte Metriken aus dem Bereich der Wohlfahrtsökonomik. Indem wir andere Metriken außer der Summe der Einkommen verwenden, das utilitaristische Prinzip, können wir unterschiedliche Aspekte des gelernten Verhaltens beurteilen. Wir zeigen, dass unser neu geschaffenes Szenario eine Unterscheidung von sozialem Verhalten und nicht-sozialem Verhalten erlaubt. Unser Algorithmus DM3 schafft es, soziales Verhalten zu lernen in diesem Miniaturbeispiel, während PPO dies nicht kann. Jedoch verschlechtert sich von DM3 die Performanz die eigene Aufgabe zu lösen, da der Algorithmus zu einem lokal schlechteren Ergebnis konvergiert. Die zusätzlich angewendeten Metriken unterstreichen die Unterschiede in den erworbenen sozialen Fähigkeiten und helfen zu identifizieren

an welcher Stelle die Algorithmen aufgrund mangelndem sozialen Verhalten oder einem suboptimalem Lösungsverhalten bei der eigenen Aufgabe schlecht abgeschnitten haben.

Contents

1. Introduction

Artificial Intelligence (AI) and machine learning are increasingly adopted not only in different business domains but also in all different areas of research, such as physics, chemistry, biology and many more. Those systems are often used to either predict, visualize our explain some form of data or learn control of robots. Usually, problems are solved using supervised or unsupervised learning algorithms. However, for challenges with a system or an agent that can interact with its environment reinforcement learning solutions may be deployed, as for example in control tasks. Since the advent of Deep Reinforcement Learning (DRL), by combining RL with deep learning, RL has gained popularity equally among researchers and media. In 1997 TD Backgammon was the first successfull application of RL in the game Backgammon [52, 51]. The first milestone that gained wider attraction was the super-human performance of a DRL agent in Atari games [31]. Thereafter, multiple other impressive challenges like mastering the board game of GO have been tackled [47]. Other hard tasks were approached soon after with Dota 2 playing bots by openAI [33]and Starcraft II bots by DeepMind [55]. An ever emerging topic in RL research are systems with multiple agents. These systems are relevant for many fields, such as behavioral sciences, control and optimization, economics, robotics and many more. Expanding single agent RL to multiple agents entails new problems to be explored [1].
Multi Agent Reinforcement Learning (MARL) is garnering more attention in recent times. In MARL agents can either be cooperative, competetive or employ mixed behavior without a clear social status of agents to one another with both friends and foes explicitly defined. Most studied scenarios are cooperative scenarios, e.g. common ressource allocation [35] or traffic control [4], and competetive games. In competetive games agents do directly compete against each other and the most common setting are multiplayer games [52, 47]. Mixed games are even more complex, where an agent has both friends and foes as for example in team games [33, 55]. But agents still have to find a globally satisfying solution, which is the same for all agents on a team. However, in those scenarios agents are explicitly interested in maximizing a team reward and/or rewards of foes are directly

coupled to form a zero-sum game. Zero-sum games feature opposing players, where the sum of values of each player is zero. Moreover, the advantage of one player is in total the same quantitative disadvantage of the other player(s).
There are many different kinds of roblems in MARL and often simplifying assumptions are required. For example from a learners local perspective the problem becomes non stationary when other agents learn concurrently and taking the joint action space into account results in issues regarding scalability [21]. Another issue in MARL is the need for definition of the learning goal, which could be finding a Nash Equilibrium or the stability of the policy or a suitable adaptive strategy given a fixed class of other agents [61].
In this work, we look into cooperation of multi agent systems with the possible extension of human interaction. In the case of multi agent systems, we do not use precomputed strategies or explicit communication. Currently, most settings concentrate on a single cooperative goal, where agents need to learn to overcome certain hurdles to achieve some globally optimal reward. A popular branch of these problems are sequential social dilemmas, where cooperation is hard to achieve because agents need to learn to not exploit others as this leads to a local optimum in the reward space [24, 42, 2]. To go even further, we want to shift the focus away from cooperation and competition, where an action is beneficial to both or bad for the competitor but beneficial for the own agent.
A rather unexplored area of MARL are coexistence scenarios. In those scenarios multiple agents act in the same environment but the reward of the agents is not linked to other agent's rewards. Each agent has one of many possible goals during an episode and tries to achieve it, obtaining only local observations and rewards. Coexistence scenarios expand on the current landscape of mainline MARL work, embedding potentially cooperative or competetive work into broader real world scenarios, where there might exist other unrelated agents. Coexistence scenarios in the real world are particularly relevant when unplanned interaction between multiple agents happens. These situations can happen in shared workspace environments, such as factories, shared open spaces like roads with multiple vehicles navigating ,or crowds, where people avoid collisions as good as possible. The goal of each agent is to maximize its own reward but also maximize the other agent's reward if there is no negative impact on its own reward. This introduces an interesting trade-off because the agents may cooperate but also have to compete potentially for limited ressources. Since each agent is considered an individual and independent entity we aim to develop a decentralized learning system.
Current MARL tasks require mostly cooperation and only few even have multiple goals for different agents, where no explicit cooperation is needed. Additionally, most environments can be solved, when maximizing for the sum of rewards for all agents such that all agents arrive at satisfying solutions locally. In contrast, we introduce new multi goal multi agent scenarios in the multi agent particle environment [29], in which agents share space and

have to avoid collisions to reach their individual goals.
Logically following, the evaluation of MARL systems with multiple selfish agents in coexistence scenarios needs additional metrics to evaluate what constitutes a good result. In coexistence scenarios the evaluator might not only be interested in how well the average of all agents performed but is concerned about other things such as fairness as well. Social welfare functions from the field of economics and sociology are promising candidates in that regard [6, 19]. In these fields measures of wealth of a group of humans or agents in our case exist that take into account the wealth (in reward) for single agents in contrast to the overall wealth. This allows us to do a richer evaluation and comparison of algorithms beyond standard measurements.
In this thesis we contribute a novel way to learn in decentralised scenarios, using a credit mechanism in reinforcement learning and introduce new environment scenarios alongside a deeper evaluation, by using additional metrics.
The remaining part of the thesis is structured as follows. The next chapter introduces related work and begins with the foundations of reinforcement learning in single agent scenarios. Afterwards, various approaches multi-agent reinforcement learning are presented and ideas from psychology are described. The third chapter elaborates our approach to learning, as well as ideas for better evaluation and why new tasks are needed in social reinforcment learning settings. The fourth chapter contains all experiments. Alongside the experiments all results are presented and discussed. In the last chapter we give an overarching discussion and summary of all results and conclude our work in an outlook.

2. Related Work

In this Chapter we introduce the foundations of reinforcement learning in single and multi agent settings as well as evaluation methods for RL systems. We start with definitions of reinforcement learning and present algorithms in deep reinforcement learning as well as various concepts of deep learning. We explore methods of single-agent RL and continue by explaining different multi-agent RL settings. The concluding part will explain other MARL methods and how the existing methods relate to our work.

2.1. Single Agent Learning

The basic problem to encounter in reinforcement learning is to control the actions of a single agent in a stationary environment without other agents or agents with fixed policies.

2.1.1. Reinforcement Learning

Reinforcement learning is a method in machine learning with which an agent (AI) adapts its behavior through reinforcement, reward or punishment, when interacting with an environment [49]. The Agent tries to maximize its reward and adopts a reward driven behavior [3, 32, 61, 21]. RL can be formulated as a Markov Decision Process $(S, A, R(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}), P(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t), \gamma)$, where $0 \leq \gamma \leq 1$ is the discount factor, which reduces the value of the rewrd collected in future states, and t denotes the current time step. A is the set of actions an agent may take and S is the set of states an agent can be in, including the subset of inital states. R is a reward function that maps a state, action and the following state to a scalar reward and P is the State transition function that maps a state action pair to a state distribution.
The action choice of an agent follows a policy π. In general, the policy π is a mapping

from states to a probability distribution over actions: $\pi \colon s \mapsto p(A = \mathbf{a}|s)$. We can compute the return G of a single episode as

$$R = \sum_{t=0}^{T-1} \gamma^t r_{t+1},$$

where r_{t+1} is the reward collected at time step $t+1$. The goal is to maximize the expectation of the reward for all episodes from each state given a policy, yielding the optimal policy π^*

$$\pi^* = \underset{\pi}{\operatorname{argmax}}\, \mathbb{E}[R|\pi]. \tag{2.1}$$

Given the basic formulation of MDPs and the key optimization problem we next introduce the most common approaches for solving tasks with RL.

Value Functions

A value function is a mapping between any state our agent can be in and a real valued scalar. It describes the expected reward when starting in a state $\mathbf{s}$ and taking actions according to the policy π. Therefore, the value function is defined as

$$V^\pi(\mathbf{s}) = \mathbb{E}[R|\mathbf{s}, \pi] = \mathbb{E}^\pi \left[\sum_{t=0}^{\infty} \gamma^t r_{t+1} | \mathbf{s}_t = \mathbf{s} \right].$$

Similarly to the optimal policy $\pi*$ we can formulate an optimal value function

$$V^*(\mathbf{s}) = \max_\pi V^\pi(\mathbf{s}), \quad \forall \mathbf{s} \in S.$$

Having either the optimal value function or the optimal policy automatically yields the other one, since we could always pick the action that maximizes the expected return in any situation. Since we do not have either but rather try to learn the optimal policy or the optimal value function we introduce another function. The state-action-value function allows us to compute the expected rewards for each action we can take for each state action pair $(\mathbf{a}, \mathbf{s})$. In other terms the function describes the quality of taking an action $\mathbf{a}$ in state $\mathbf{s}$ and is accordingly named the quality function or short Q-function. The Q-function is defined as

$$Q^{\pi}(\mathbf{s}, \mathbf{a}) = \mathbb{E}\left[R|\mathbf{s}, \mathbf{a}, \pi\right].$$

Both the value and Q-function depend on the states and available actions because some states may be intrinsically better than others, such as a winning position in a board game. But they also depend on the policy since even in a favorable state a bad policy may yield bad returns.

The last notion we want to introduce is the one of advantage [23, 15]. Advantage aims to compute the difference of taking any action versus taking an action according to the current policy. Thus, we get a notion of better or worse with respect to our current policy. The advantage function is defined as

$$A^{\pi}(\mathbf{s}, \mathbf{a}) = Q^{\pi}(\mathbf{s}, \mathbf{a}) - V^{\pi}(\mathbf{s}).$$

We can deduce that an advantage is positive if Q^{π} is bigger than V^{π} and a negative vice versa. Intuitively, we can improve our policy by making actions more likely that have an advantage over the actions chosen by our current policy.

Given the mathmatical foundation of value, quality and advantage the goal is to solve for the optimal solution. To this end, we reformulate our problem in a recursive setting, where the value of our function depends on the recursive application of itself. This paradigm is known as dynamic programming [5] and it allows us to rewrite the value and Q-function as their corresponding Bellman equations

$$V^{\pi}(\mathbf{s}) = \sum_{\mathbf{a} \in A} \pi(\mathbf{s}, \mathbf{a}) \sum_{\mathbf{s} \in S} p(\mathbf{s}'|\mathbf{s}, \mathbf{a}) \left[r(\mathbf{s}, \mathbf{a}, \mathbf{s}') + \gamma\, V^{\pi}(\mathbf{s}')\right],$$

$$Q^{\pi}(\mathbf{s}, \mathbf{a}) = \sum_{\mathbf{s}' \in S} p(\mathbf{s}'|\mathbf{s}, \mathbf{a}) \left[r(\mathbf{s}, \mathbf{a}, \mathbf{s}') + \gamma \sum_{\mathbf{a}' \in A} \pi(\mathbf{s}', \mathbf{a}') Q^{\pi}(\mathbf{s}', \mathbf{a}')\right].$$

Since we assume the Markov property, the Bellman equations can be proven to converge to the best policy with Q-learning [57]. Q-learning tries to learn the Q-function for a given MDP, that is it estimates the value of state-action pairs. The optimization process then usually consists of two alternating steps, policy evaluation and policy improvement. But the major drawback of dynamic programming is that it can only be used if the dynamics of our MDP are fully known. To add to that, the computational complexity rises exponentially, also known as the curse of dimensionality, if the number of states

and state-action pairs gets higher, rendering the algorithm useless for most real world applications. If the environment is a priori unknown to the agent, one needs to explore it to get estimates of the V and Q functions. Since bootstrapping value functions using dynamic programming methods is infeasible, we instead use Monte Carlo methods where we estimate the expected return of being in a state. This is done by averaging the return from many different rollouts of our current policy. Thus, we can estimate the value given a state $\mathbf{s}$ and a policy π as

$$V^{\pi}(\mathbf{s}) \approx \frac{1}{M} \sum_{i=1}^{M} R^{i},$$

where M is the number of rollouts from state $\mathbf{s}$ and R^{i} is the Return of the i'th rollout. The main drawbacks of this formulation are the need of episodic MDPs and a sufficient level of exploration. A lack of exploration may lead to missing the optimal solution and the agent may be stuck not solving the task or simply not getting out of a local optimum. This problem is also known as the eploration exploitation dilemma. If the agent explores too much the evaluated policy simply becomes close to a random policy and does not yield useful behavior or converge to any coherent behavior. Likewise, if we only select actions using our current policy the agent might never explore areas of our environment or action sequences that would improve the final return, thus, never actually reaching the global optimum. Two possible solutions to the exploration problem are ϵ-soft policies and the Softmax action selection method. ϵ-soft in the simplest case is ϵ-greedy, where we select the action according to our policy with a probability of $1 - \epsilon$ and select otherwise randomly one of the other actions. In practice one starts with a high ϵ value so that many random actions are taken at the beginning of learning, basically aquiring knowledge of the world. Later on with lower values the policy starts to exploit this knowledge to find the optimal solution. In the case of Softmax action selection each action has a probability of being assigned based on its relative Q-value to all other actions. When using a softmax distribution one can adjust the "temperature" over time, which adjusts the entropy of the action distribution. This allows to introduce simulated annealing, which for a slow enough annealing process guarantees convergence. However, epsilon greedy methods can be applied universaly to all policy methods to establish sufficient exploration.

In general, to deal with exploration there are two different types of methods available. The first are on-policy algorithms, where SARSA [40] is a common example using temporal difference learning. With on-policy methods we create some episodes with our current policy and use some form of ϵ-soft algorithm to ensure exploration. Second are off-policy

algorithms, which use a behavioral policy to generate episodes. The behavioral policy is a different policy from the learned policy. The advantage of off-policy learning is that we can target the search to more promising areas of the state-action space for example by using expert generated policies. The disadvantage in that case is that we will not explore states that the behavioral policy never visits, thus missing potentially better solutions.

Policy Gradients

Instead of learning the value of states there is another learning paradigm in reinforcement learning called policy search. We learn the policy and maximize (2.1) directly. We do that by maximizing the expected returns over all trajectories generated by our current policy. Therefore the objective function becomes

$$J(\theta) = \mathbb{E}_{\tau\sim\rho_\theta}[R(\tau)] = \mathbb{E}_{\tau\sim\rho_\theta}\left[\sum_{t=0}^{T}\gamma^t\, r(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1})\right],$$

where $\tau = (\mathbf{s}_0, \mathbf{a}_0, \mathbf{s}_1, \mathbf{a}_1, ..., \mathbf{s}_T, \mathbf{a}_T)$ is a trajectory generated by our policy and sampled from the distribution ρ_θ with θ being the parameters of our policy. In policy gradient approaches the main idea is to take the gradient of the objective function with respect to the policy parameters [36]. Updates are then straightforward and using gradient ascent we obtain the update rule

$$\theta_{\text{new}} = \theta_{\text{old}} + \eta\, \nabla_{\theta_{\text{old}}} J(\theta_{\text{old}}).$$

The REINFORCE [58] algorithm estimates the gradient as

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\rho_\theta}[\nabla_\theta \log \rho_\theta(\tau)\, R(\tau)]. \tag{2.2}$$

The estimate is Monte Carlo sampling as we take the average of the reward-weighted log-likelihood of a trajectory. Because we use the empirical returns of each trajectory in the gradient computation the variance of the gradients is usually very high. We can lower the variance by introducing a baseline and subtract it from the empirical return. To do so different algorithms have been proposed that also extend the system to actor-critic methods, which will be discussed in the deep reinforcement learning section. Policy gradient methods have a wide array of advantages. Since the formulation does not

rely on knowing the dynamics of the MDP they are inherently model-free and we do not need to know about task or reward models. Furthermore, policy gradient methods are easily applicable to continuous action spaces in contrast to value based methods. Other advantages are that policy gradient methods empirically have better convergence properties than Q-learning methods and are generally more stable [10], they are effective in high dimensional action spaces and they learn stochastic policies. Disadvantages that one has to face are the inefficient learning of policies through poor sample efficiency, the convergence to local optima instead of the global one and that the evaluation of the policy gradient has a high variance.

2.1.2. Deep Reinfocement Learning

Deep Reinfocement Learning (DRL) and especially the use of Convolutional Neural Networks (CNN) [26] have made RL methods viable for high dimensional state-action space problems. Additionally, Recurrent Neural Networks (RNN) [39] and Long-Short Term Memory (LSTM) [22] architectures make it possible to apply DRL algorithms to problems with temporal dependencies between states. In this section we present the advances in Deep Reinforcement Learning (DRL) and mention how the algorithms address different challenges encountered in the base formulation of RL.

Value Based

The first succesfull application of DRL with a value based approach was the DQN algorithm [30]. DQN solves two prominent problems, the correlation of inputs and outputs that arises because succeeding states are not independent, violating the assumption that states are independent and identically distributed and the non-stationarity of the target Q-values. These issues are solved with two seperate mechanisms in DQN. The first is an Experience Replay Memory (ERM) that stores transitions of many runs. Then, training is done on a random batch sampled from the ERM. The other proposal was the use of a target network that computes the target Q-values and is frozen over many episodes. It is only updated very infrequently, allowing the Q-network to learn stationary target values and thus converging.

While ERM and target networks have an overall positive effect, their use slows down learning by a significant margin and increases sample complexity. To aleviate those problems several modifications have been proposed, such as double Q-learning [54], prioritized experience replay [43], duelling networks [56] working with an advantage

function or a recurrent version DRQN [18] that is able to tackle POMDPs thanks to an LSTM [22] layer.

Policy Gradient Based

In recent years policy gradient methods in conjunction with DNNs have garnered a lot of interest in the research community and much progress has been made on the algorithmic side such as DDPG [48], TRPO [46], PPO [45] and PPG [9]. As mentioned earlier we can estimate the policy gradient as in Eq. (2.2). According to the policy gradient theorem we can rewrite this equation in terms of single transitions instead of complete trajectories as

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} (\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(\mathbf{s}_t, \mathbf{a}_t) \sum_{t'=t}^{T} \gamma^{t'-t} r(\mathbf{s}_{t'}, \mathbf{a}_{t'}, \mathbf{s}_{t'+1})),$$

where additionally the concept of causality is introduced. Because the future reward does only depend on the reward of future states we simply take the sum of the rewards of future states, which is the definition of the Q-function. Following this notion Sutton et al. [49] showed that we can rewrite the gradient of the objective function as

$$\nabla_\theta J(\theta) = \mathbb{E}_{\mathbf{s}\sim\rho_\theta, \mathbf{a}\sim\pi_\theta}[\nabla_\theta \log \pi_\theta(\mathbf{s}, \mathbf{a})\, Q_\varphi(\mathbf{s}, \mathbf{a}))],$$

where a function approximator Q_φ is introduced to estimate the Q-function. The resulting architecture is actor-critic as we have a policy $\pi_\theta(\mathbf{s}, \mathbf{a})$ and a critic $Q_\varphi(\mathbf{s}, \mathbf{a})$.

So far the policy gradient is strictly on-policy as we are only allowed to use samples generated by the current policy. This restriction arises because the data distribution is dependent on the policy and becomes non stationary as the policy changes. An important concept to overcome this issue is importance sampling. We can rewrite our objective function in terms of a behavior policy b and the current policy θ and obtain

$$J(\theta) = \mathbb{E}_{\tau\sim\rho_\theta}[R(\tau)] = \mathbb{E}_{\tau\sim\rho_b}[\frac{\rho_\theta(\tau)}{\rho_b(\tau)} R(\tau)],$$

where the ratio $\rho_\theta(\tau)/\rho_b(\tau)$ is the importance sampling weight. It controls how much we learn from generated trajectories by decreasing importance of trajectories that have a much higher probability to be generated by the behavior policy than the current policy.

General Advantage Estimate

An important aspect to make Policy gradient based methods more stable is the Generalized Advantage Estimate (GAE) [44]. GAE adresses the bias/variance trade-off in machine learning. In reinforcement learning we get a small bias in the return term of the advantage formulation, if we rely on real rewards, that is the general return rollout of the trajectory. If we instead take estimates like the TD error

$$\delta_t = r_{t+1} + \gamma V^\pi(s_{t+1}) - V^\pi(s_t)$$

we get a high bias but small variance leading to a more stable gradient that in return may be more incorrect due to the bias. Schulman et al. takes the n-step advantage

$$A_n^t = \sum_{k=0}^{n-1} \gamma^k r_{t+k+1} + \gamma^n V^\pi(s_{t+n+1}) - V^\pi(s_t)$$

and instead of choosing a concrete n they reformulate the GAE as

$$A_t^{\text{GAE}(\gamma,\lambda)} = (1-\lambda) \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}, \tag{2.3}$$

where $\delta_t = r_{t+1} + \gamma V^\pi(s_{t+1}) - V^\pi(s_t)$. The parameter λ now allows to control the bias/variance trade-off. When $\lambda \geq 0$ is small the advantage estimate tends to the TD advantage, with a high bias and small variance. Whereas with a high value of $\lambda \leq 1$ the estimate becomes closer to the Monte Carlo advantage formulation leading to a low bias but high variance.

Proximal Policy Optimization

PPO [45] is a policy gradient method. It constrains how much a policy may change in a single update by clipping the policy gradient. In the original work multiple loss functions are proposed and tested. The surrogate loss function that we adopt and optimize is

$$L^{\text{CLIP}}(\theta) = \mathbb{E}_t \left[\min(\rho_t(\theta) A_{\pi_{\text{old}}}, \text{clip}(\rho_t(\theta), 1-\epsilon, 1+\epsilon)) A_{\pi_{\text{old}}} \right]. \tag{2.4}$$

ρ is the importance sampling weight, ϵ tells us when to clip the importance sampling weight and $A_{\pi_{\text{old}}}$ is the advantage estimate. The first term in the min function is the surrogate loss as already formulated in TRPO [46]. The second part clips the importance sampling weight to restrict changes made to the policy. An alternative loss function uses the Kullback-Leibler divergence to restrict the change in distribution of the behavior. We decided for the clipped objective function because it is straightforward to interpret and works very well in practice. PPO allows to train multiple steps on the same batch of gathered experience, which greatly increases sample efficiency. The optimization process can be done using gradient descent, enabling us to use common optimizers.

2.2. Multi Agent Learning

In the real world many problems do not involve only a single actor but actually have multiple participants. Adressing this change, in RL the multi agent case gained a lot attention in the recent years. Single agent RL Q-learning solutions suffer from the non-stationarity of the environment as the transition dynamics of the MDP become dependent on other agents that change. Policy gradients on the other hand have to deal with an increasing variance as the number of agents grows making learning prohibitively inefficient [28]. Single agent solutions work well as long as only one agent learns even if others are present as they do not change and can be modeled as part of the environment. In Multi Agent RL (MARL) solutions multiple agents learn simultaneously, and may use different strategies. We give an introduction to MARL and introduce important algorithms that tackle the above issues.

2.2.1. Multi Agent Settings

The reward structure of an environment in MARL can be cooperative, competetive or mixed. In cooperative settings agents usually must learn to cooperate to achieve a common shared goal. In contrast in competetive settings agents compete against each other in a zero sum game. The gains of one agent are the losses of the others and the goal is trying to learn how to win. At last in the mixed setting agents may be cooperative and competetive at the same time as for example is the case in team sports. [21] A full classification of task types and agent awarness is given in Fig. 2.1.

Task type → ↓ Agent awareness	Cooperative	Competitive	Mixed
Independent	coordination-free	opponent-independent	agent-independent
Tracking	coordination-based	—	agent-tracking
Aware	indirect coordination	opponent-aware	agent-aware

Figure 2.1.: MARL algorithms categorized by their task type and the degree of awarness an agent has of other agents in the environment. [8]

2.2.2. Agent Independent Learners

The simplest approach to MARL problems is to have all agents learn seperately. This notion was introduced in independent Q-learning (IQL) [50] where agents access their local observations and have to maximize a joint reward. These types of independent learners are easily scalable and they are fast since they only need local observations. The original version of IQL uses tabular learning which suffers from the same problems of dimensionality as classical RL. Implementing DQN for IQL is not trivial since the other learning agents introduce non-stationarity in the sampled data distribution [34, 8, 21, 61]. This makes it infeasible to use replay memory as the underlying data distribution changes inbetween episodes. This could be circumvented by only training on the most recently generated samples but decreases sample efficiency greatly. In order to resolve this problem one example is to augment transitions with the probability of choosing the joint action [41]. Another approach to stabilise the experience replay is to use Multi Agent Importance Sampling (MAIS) to adress the non-stationarity in MARL systems. MAIS decreases the importance and sampling probability of irrelevant data [12].

2.2.3. Fully Observable Critic

Independent learners can fail to converge under the non-stationarity of multi agent environments. To resolve this issue one can use a fully observable critic. A fully observable critic has access to the complete state including the local perspectives of agents and all actions. For any observer that has all information and knows all actions there exists no non-stationarity because the transition dynamics do not depend any longer on the policy

of the agents [34, 28]. One of the earliest extensions using a central critic and deep learning is the Multi Agent Deep Deterministic Policy Gradient (MADDPG) algorithm [28]. Training is done with a centralized critic while actors work decentralized only observing local information. Lowe et al. propose to augment the critic in actor critic policy gradient methods with information about the other agents.
But the policy of any agent can get stuck in local optima w.r.t. its other learning partners, that is the policy is only optimal against the current policy deployed by each agent. This leads to suboptimal behavior when other agents show diverse, rarely seen action sequences. In order to alleviate this problem Li et al. [27] propose a minimax extension of MADDPG called M3DDPG to learn robust policies. To deal with intractability in the minimax objective they introduce the concept of Multi-Agent Adversarial Learning (MAAL).
Another model where the critic is shared but the actors are instead trained locally uses Counterfactual Multi Agent (COMA) policy gradients [11]. The central Q-function uses a joint reward to learn, which is used in a counterfactual baseline to train each actor locally. Using the centralized critic, a counterfactual baseline can be computed which allows to compute the contribution of an action via an advantage function. QMIX [38] uses a centralized factorized critic function for learning with decentralized execution. They use one critic per agent and combine them into a global critic with a non-linear mixing network.
So far the algorithms were designed to handle single goal games, as the agents pursued a common goal, either in a cooperative or competetive scenario with other agents. Considering single goal games reduced the complexity of the theoratical frameworks that could be used. Following, not much work has been done for multi-goal systems so far.

CM3: Yang et al. [60] introduce CM3 that is designed to deal with cooperative multi-goal settings. In contrast to other collaborative algorithms, agents do not contribute towards a global goal but exist independently and receive only local rewards. They use a curriculum learning approach and distribute training agents to two stages. They first train a single agent to attain goals in an environment in a reduced single agent MDP and then in a second stage learn cooperation between agents. After the initial training of a single agent, the policy and Q-function are augmented by expanding the input state with observations of the other agents and reinstating the already learned parameters. The Q-function is enhanced to build two resulting networks, a global Q-network and the credit function. Using the credit function they can estimate a credit assignment where the credit of agent m is computed as the Q-value $Q_n^\pi(s, a^m)$ for agent n with goal g_n with the action by agent m. The global Q-function takes instead of a single action the joint action of all agents and evaluates $Q_n^\pi(s, \mathbf{a})$. The problem of estimating a global Q-function is that one necessarily needs access to the global state and the joint action of all agents.

2.2.4. Agent-Aware Learners

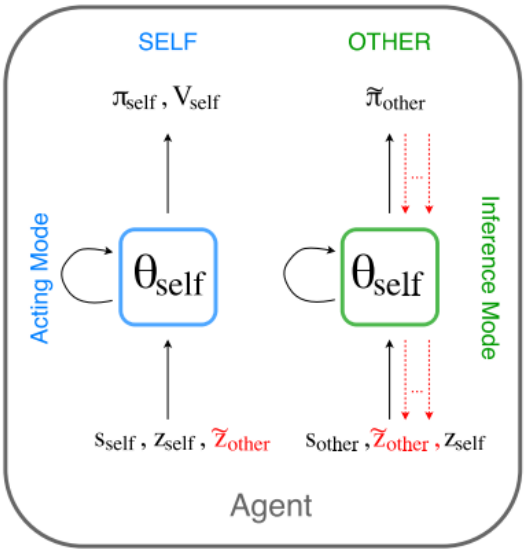


Figure 2.2.: Self Other-Modeling architecture. During an episode each agent takes as input its observation, goal and the inferred hidden goals of another agent. Then in inference mode it assumes the position of the other agent by taking its state and feeding the inferred hidden goal as its own. [37].

The idea to model other agents using oneself has origins in psychology and the studies of human behavior [14, 13]. The capability of humans to create complex beliefs about others mental processes is named Theory of Mind (ToM) [59]. Using ToM models allows to better adapt to behavior of other agents and establish more efficient interactions [7]. We use the concept of modelling others during the learning phase and mentally take the position of others to gauge how much influence actions have on other agents and their respective goals. In partially observable settings with multiple agents, maximizing their own reward, it is important to be aware what the goals of other agents are. To this end Raileanu et al. introduce Self Other-Modeling (SOM) [37], which draws from those ideas rooted in psychology. As the name suggests to predict and infer the hidden goals of other agents the method uses its own behavior function as described in Fig. 2.2. To model the other agents' goals continuously throughout an episode a multi-layer recurrent neural network is used to iteratively update the belief of the hidden states. They show that explicitely modelling the goal of other agents increases the performance in competetive and cooperative games

but comes at the cost of increased model complexity and longer learning times. In our work we shift the active goal inferring completely into the learning process, meaning that there is no seperate inferred goal input during an episode for the policy but instead the goals are inferred only for learning purposes. We decide against an explicit goal modelling during execution to be able to use more simple network architectures in our experiments.

2.3. Evaluation of Reinforcement Learning Algorithms

Validating and testing lie at the core of succesfull algorithm development. In this light we give a brief overview of common evaluation techniques used. In general, evaluation can be done either via theoratical evaluation, yielding convergence guarantees or be of empirical nature, measuring interesting general properties.

2.3.1. Evaluation in Single Agent Settings

One of the most important evaluation concepts in RL is the one of convergence, that is if the behavior of our agent becomes stable at some point [8]. On the theoretical side one can proof that an algorithm converges to a Nash equilibrium [49]. On the empirical side the interesting properties of an algorithm are the following: How good does an agent perform after convergence? How fast does an agent's behavior converge? Does the agent solve the problem at hand? The first and last question can be quickly evaluated by looking at the reward of the agent for each episode and if the agent is above a certain treshold the task is considered as solved. To measure how fast an algorithm converges, one usually considers the sample efficiency as execution time depends on the hardware that is available. The sample effiicency is defined as the amount of samples (transitions) or episodes needed for training to reach convergence.

2.3.2. Evaluation in Multi Agent Settings

MARL algorithms largely use the same evaluation measures as single agent RL but have a few adaptations. Namely, proving the convergence to Nash equilibria might not be so useful in this setting because a more important aspect could be the ability to adapt to the behavior of other agents in an optimal way [62]. Regarding reward in cooperative

settings the reward is either global and we can simply look at the team reward or if agents obtain local individual reward the rewards are usually averaged. If the agents have to maximize a common overarching goal together by definition these evaluation methods function properly but if agents do not share the same overarching goal this measure fails to capture other aspects. When developing a system with multiple actors one might want a certain distribution of rewards and cares about distributional attributes of a policy on a population. This concept is known as cardinal social welfare in economics [16, 17, 19] and is well studied. One important prerequisite how to choose a policy is the veil of ignorance as introduced by Rawls [25]. The veil of ignorance refers to the state of the policy selector in that he does not know how he is affected by the chosen policy, which ensures the fairest selection process by a rational policy maker. To measure the fairness one needs a criterion, which tells how well a population did in regard to that criterion. The classic approach in MARL is to use the sum of all rewards, which corresponds to the utilitarian approach of measuring welfare. We present two other measures to obtain a richer picture of the resulting behavioral characteristics of the emerging policies in MARL settings. The first being the Rawlsian measure [25], where we look at the individual with the lowest income. And the second being the Theil-Index [53], which measures the inequality in the reward distribution. Intuitively, the Theil-index measures the ratio of the income of a person to the average income.

3. Methodology

We present the following problem scenario for our MARL problem. Consider agents sharing an environment and acting independently. We assume that each agent has its own goal that it tries to optimize towards. Rewards for reaching a goal are only given to that particular agent and no other agent benefits from another agent reaching their goal. Agents do not have to know the goals of other agents during execution and rely only on local observations. Inferring of others intentions and view is done in the learning phase to avoid as much computation as possible during execution. Actions are chosen independently according to a policy. While each agent chooses actions separately during an episode, we use parameter sharing resulting in the same policy for each agent. However, their behavior differs because each agent will get a different goal input. In the learning phase each agent has to infer the goal and observation of the other agents to compute gradient updates. In this work we assume perfect inference capabilities for our toy system to show the effects of our modified policy gradient. Nevertheless, in more complex environments one would need to incorporate goal and observation inferring mechanisms. We introduce a modified policy gradient using a credit network alongside an actor and a critic network, all of which can be trained in a decentralized manner, given inference capabilities, without access to global states.
As baseline we use the work of CM3 [60] and the PPO algorithm [45]. We use scenarios of multiple agents with individual goals and local reward functions and local observations. We test the multi-stage setup in a curriculum similarly to the CM3 algorithm. The main contributions of our work are the following:

(1) We estimate how we impact another agent without a central Q-function. Instead, we use a combination of a credit function and a value function, in our case realized with neural networks. The credit function computes the credit an agents' action takes towards solving the goal of the observing agent. The value function receives a single observation and returns the approximated value of being in that specific state. In combination we use them to formulate a new decentralised policy gradient.

(2) We introduce a new scenario in the multi-agent particle environment to test coexisting qualities of agents. To be considered successful in a coexisting environment as an agent, it needs to be able to fulfill its own task as good as possible while reducing negative impact on others in the environment.

(3) In addition to the sum of rewards metric we use metrics from social economics to analyze the different characteristics of the utilized policies with changed requirements in coexisting scenarios.

3.1. Environment

Traditionally used evaluation environments have a strong bias towards strictly cooperative solutions or are created for the purpose of cooperation. Usually only a common global goal for all agents exists and the best behavior as defined beforehand emerges by maximizing the sum of the joint rewards. In contrast our goal is to create an environment where multiple goals exist that in separation do not require cooperation but interference may happen between multiple agents when they pursue their tasks simultaneously. The closest already existing scenarios are autonomous vehicle overtaking and several scenarios in multi agent particle simulations [29] as presented by Yang et al. [60]. We build upon the multi-agent particle environment to define our own scenarios improving the reward design to prevent exploits regarding our research goal. To be able to show effects of our own proposed algorithm we want the following assumptions to hold true.

- At least one agent is able to influence other agents rewards without siginificant changes to its own reward.
- An agent should not depend on other agents to be able to reach a goal.
- An agent should not get punished if other agents take longer or do not fulfill their respective task.

We introduce a new scenario because the first and third point are to our knowledge not fulfilled in the other tasks we use. We did not find a multi-goal multi-agent environment that meets all these criteria in the related work we examined. Published environments to reproduce results are not available except for the ones of Yang et al. [60], which motivated us to create a toy example with equally simple rules. Our toy example adds a new reward structure to the existing environment, where one agent may exert influence over another

agents' reward without an effect on its own. We give a detailed description of our new task in Chapter 4.

3.2. Evaluation

Current evaluation methods only take the sum of rewards of all agents into account. While this is suitable for optimization for joint or global reward systems in a cooperative setting it does not capture more intricate properties during evaluation, such as measures of fairness, in environments, where agents have separate goals. In a system where all agents want to independently solve their own task the global optimum of the sum might starve single agents of reward for the greater good. While in a fully cooperative utilitaristic setting the global maximum reward might be desired, in the real world different entities demand at least a partial resolution of their own problem at hand even if its not the global optimum. To further illustrate the weakness of the sum of all rewards in this scenario, consider the simplest case of two agents Alice and Bob, each gaining a reward between zero and one. In the first case, Alice gets a reward of one and Bob a reward of zero. In the second case, both Alice and Bob get a reward of 0.5. If we consider the agents separately one of them failed at his task at the benefit of the other and in case they belong to different entities, like drivers of vehicles, the satisfaction of the found solution is low for the disadvantaged driver. Therefore, we need a new evaluation measure of inequality for our reinforecement learning system to asses the fairness of the reward distribution of agents. We use two social welfare functions in conjunction to the utilitarian reward measure, the sum of rewards. The first used metric is the Rawlsian metric [25]

$$R_{\text{Rawls}} = \min(R_1, R_2, ..., R_n). \tag{3.1}$$

The Rawlsian metric only cares for the reward of the worst off member, by taking the minimum reward of a member of a population. This metric tells us how well even the worst agent is taken care of and better solutions should yield a higher minimum reward of a member. However, this fails to account for overall better outcomes for the whole population because we can not distinguish between the solution $(1, 0)$ and $(1, 1)$. Thus, it is clear why optimizing towards such metrics is unfavorable but also that no single metric

captures all important aspects that might be of interest.
The second metric we use is the Theil-index [53]

$$R_{\mathrm{T}} = \frac{1}{n} \sum_{i=1}^{n} \left(\frac{r_i}{\mu} \ln \frac{r_i}{\mu} \right). \tag{3.2}$$

r_i denotes the reward agent i received for an episode and μ is the mean of all rewards. If an individual's reward is the same as the average reward its contribution to the score is zero and it gets higher the further away it is from the average. Thus, a better Theil-index should score lower if we aim for a more equal distribution of rewards.
That said, both measures alone do not evaluate the fitness of a policy in isolation but require other factors as the sum of rewards to gauge and analyze how good a policy is. We also want to empaphize that there is no silver bullet solution in that regard and the policy maker has to find a sweet spot in terms of this trade-off depending on the task and desired evaluation metrics.

3.3. Multi-Goal Multi-Agent Setting

In the following section we describe the DM3 framework. We start by formulating our multi-goal multi-agent problem formally, followed by an introduction of the mechanisms used to construct our MARL policy gradient. To define a MARL policy gradient we define a credit function that allows us to compute not only the credit we take towards our own rewards, but also lets us compute the influence we have on other agents. To compute the influence we use a concept known from psychology, the theory of mind. Instead of estimating the influence an agent has on another agent, the agent uses a mental model to estimate the impact of its action from the perspective of the other agent. As a consequence, if we assume that agents share similar knowledge about the world, the agent can use the model of itself to estimate the credit from the other agents perspective. Since we assume a similar knowledge about the world and each agent is supposedly familiar to the same extent on how to reach each goal, we enable parameter sharing between agents.
We present an episodic multi-goal Markov game as it is introduced by Yang et al. [60] and adapt it to our needs. In our multi-goal MARL scenario, each agent gets a local reward and has to maximize it for its own success. A multi-goal Markov game is a tuple $(S, O^n, A^n, P, R, G, N, \gamma)$ with $|N|$ agents labeled by $n \in N$. In each episode, each agent n has one fixed goal $g^n \in G$ that is known only to itself. At time t and global state

$s_t \in S$, each agent n receives an observation $o_t^n := o^n(s_t) \in O^n$ and chooses an action $a_t^n \in A^n$. The environment moves to s_{t+1} due to joint action $\mathbf{a}_t := \{a_t^1, ..., a_t^N\}$, according to transition probability $P(s_{t+1}|s_t, \mathbf{a}_t)$. Each agent receives a reward $R_t^n := R(s_t, \mathbf{a}_t, g_n)$, and the learning task is to find stochastic decentralized policies $\pi^n : O^n \times G \times A^n \rightarrow [0, 1]$, conditioned on their own local observations and goals, to maximize $J(\pi^n) := \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, \mathbf{a}_t, g^n)\right] \quad \forall n \in N$, where $\gamma \in (0, 1)$. We start by presenting the different techniques used.

3.3.1. Curriculum Learning and Function Augmentation

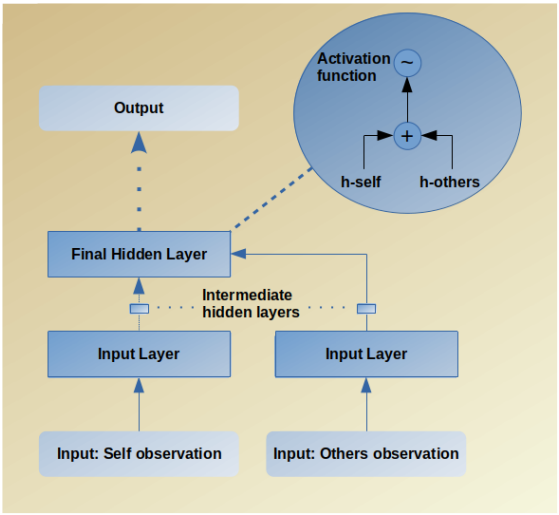


Figure 3.1.: Augmentation scheme used in the policy, value and credit network to enable two stage learning. The observation part about oneself is seperated with observations about others. The left part only gets observations about oneself, while the right network part that augments in stage 2 only gets information about other agents. In the final hidden layer, before applying the activation function, both outputs from the last intermediate hidden layers of the two network parts are added.

Following the reasoning of Yang et al. [60] we introduce curriculum learning and a function augmentation scheme to help alleviate the exploration problem that arises in MARL especially with multiple goals. Exploration vs. exploitation is already a demanding challenge in RL but the problem is further excarbated in multi-agent settings because one

does not only need to learn how to solve an individual task, but do so in the presence of others. To combat this issue we use the curriculum learning setup as in CM3. In the first stage we learn a reduced single player MDP. This is reasonable as most environments in use currently can be decomposed into a self observation part, observation of others and observation of the environment [20]. When training for the single agent game, the policy does only get self observations and observations of the environment. The resulting policy is greedy w.r.t. its goal and is optimal in the single agent case. The reduction of a multi agent MDP can be done by deleting all components of the state vector that conveys information about other agents. As a result, in the first stage we train on the single agent MDP and in the second stage we train with multiple agents in the multi agent environment.
To enable the two stage learning setup we need to split the network architecture into two separate parts as can be seen in Fig 3.1. During the first stage, the agent only observes the environment and observations about itself, whereas in the second, information about other agents is also observed. This seperation is achieved by decomposition of the state space into a self observation part O_{self} and observation of others O_{others}. During the second stage a separate network for O_{others} is introduced and the output, before applying the activation function, of the last hidden layer h_{others} is added onto the last hidden layer h_{self} of the first network. Hence, the final layer is computed as

$$h_{\text{out}} = \sigma(h_{\text{self}} + h_{\text{others}}), \tag{3.3}$$

where σ is the nonlinear activation function of the last hidden layer. The new initialized layers learn to process surrounding agents. By fusing the two parts the greedy initialized networks now need to learn how to solve the task with influence of others and how to coexist in a productive manner.

3.3.2. Credit network

To learn policies in multi-agent environments, one does not need only an estimate of how well a state is, but actually how much each other agent contributes to his own state value. By introducing a credit network that only computes the quality of one action towards the goal of a single agent, we remove the necessity to observe all other agents at any given time. Instead we only need to compute the credit of agents, that are in the vicinity and observable from the observers point of view. The computational complexity can also be reduced to managable levels because one does not need to take the whole state-action

space into account. While this reduction might be insufficient for some problems this line of reasoning is applicable, where environments and interactions are of local nature. If an agent can be influenced by other agents outside of its view, then we would expect learning to be severly harder. Whenever we iterate over other agents in the following formulas $m \rightarrow N_t$, with N_t being the number of observed agents at any given time step t, we implicitly assume that only at that time point observable agents are taken into account for the computation. Additionally, we fixed the amount of agents whose observation is passed into the networks because all scenarios used only deal with local interactions. Generalization to many more agents is not further discussed in this thesis.
Nevertheless, following the idea and proof of Yang et al. [60] we introduce a decentralized credit assignment network $Q_n^\pi(o_n, s_m, a_m, g_n)$. The credit function evaluates how much an agents m action a_m in the observed state o_n contributes towards goal g.

Definition: The credit function for two agents n, m is defined as

$$Q_n^\pi(o_n, s_m, a_m, g_n) := \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t R_t^n | o_n^0 = o, a_m^0 = a_m, s_m^0 = s_m \right]. \tag{3.4}$$

Proposition: For all agents n, m, the credit function 3.4 satisfies the relations

$$\begin{aligned} Q_n(o_n^t, s_m^t, a_m^t, g_n) &= \mathbb{E}\left[R_t^n + \gamma Q_n(o_n^{t+1}, s_m^{t+1}, a_m^{t+1}, g_n) | o_n^t = o, a_m^t = a_m, s_m^t = s_m \right] \\ V_n'(o_n^t, g_n) &= \sum_{a_m} \pi_m(a_m | \hat{o}_m, \hat{g}_m) Q_n(o_n^t, s_m^t, a_m^t, g_n) \end{aligned}$$

The introduced credit function takes the current observation, the goal of an agent n, the observed state and the action of an agent m as input. The observed state of agent m is used as an identifier of the agent in the observation to differentiate whose action is evaluated. This allows the assignment of a credit value to every action of each agent, even oneself, given one's own observation towards one's goal. To train the credit network we use the PPO algorithm and replace the temporal difference estimate to compute returns with

$$\delta_t = \gamma Q_n(o_n^{t+1}, s_m^{t+1}, a_m^{t+1}, g_n) - Q_n(o_n^t, s_m^t, a_m^t, g_n), \tag{3.5}$$

resulting in the credit GAE estimate $A_{q,t}$ using equation 2.3.

3.3.3. Decentralized Credit Advantage

An integral part of this work is the formulation of a cooperative coexisting advantage. It aims to balance the desire to fullfill the own goal and still take into account the effect on others. Additionally, we need to learn which actions actually take credit for an overall improvement in the state-value. To achieve this goal we introduce three components in our advantage estimate utilizing a credit and a value network. The first component is the temporal difference error, the second is a credit estimate for the taken actions and the third component is an influence estimate.

TD-Residual Error

One core contribution is the design of a decentralized credit advantage. In the partially observable case with multiple goals, where agents have to infer the goal of the other agents, estimating the influence each agent has on others is not trivial. Nevertheless, the raw value difference between two subsequent states is a good base estimate of how well an agent is doing, if it does not depend too much on others to reach its goal. Since we want to explore learning and living in cohabitation scenarios it is reasonable to assume a certain degree of independency of agents. The first advantage estimate with goals as input added is the TD-residual error

$$A_{\text{td}}(o_n^{t+1}, o_n^t, g_n) = \gamma V(o_n^{t+1}, g_n) - V(o_n^t, g_n) \tag{3.6}$$

which is introduced in the GAE estimate of PPO [45].

Credit Advantage

The second advantage component estimates the advantage an agents action has for itself. While the TD-residual error already deals with how well an agent is doing, we want to add a learning signal that explicitly incorporates only the own estimated impact on the agents goal. We compute the credit with the learned Q-function from the agents viewpoint and its own action and subtract the expected credit it would recieve in that situation. The estimated credit advantage computes a learning signal that influences how well an agent does towards its own goal.

Definition: The credit advantage of agent n is

$$A_{\text{credit}}(o_n, a_n, s_n, g_n) = \gamma Q_c(o_n, s_n, a_n, g_n) - \sum_{\hat{a}} \pi(o_n, \hat{a}, g_n) Q_c(o_n, s_n, \hat{a}, g_n).$$

Q_c is the credit network that evaluates the quality of the taken action.

Influence Advantage

The third advantage is how much better the impact of an action is on others than expected. Humans use a mental model of others to predict future actions of them. The theory of mind suggests that we can exploit the reasoning capabilities of others to better grasp their understanding of the situation. Humans use available information to infer what others know and how that influences their decision. To take the perspective of others comes with great advantages for humans when trying to coordinate.
We want to benefit from this thought process that humans employ in a similar manner during learning. When taking an action the agent does not only benefit or harm its own reward but changes the environment for other present agents that can see it. While a direct approach without perspective taking would employ an estimator how we change the others reward based on our view, we instead infer what the other agent sees and evaluate our action from its position. Then, given an observation and goal from the other agent m, our own state and action, we compute the influence on another agent using the credit network Q_c. Finally, by incorporating the resulting advantage estimate, this allows us to select actions that are more beneficial towards others.

Definition: The influence advantage of an agent n with other agents $m \in N$ is

$$\begin{aligned} A_{\text{influence}}(\mathbf{o}, a_n, s_n, \mathbf{g}) &= \sum_{m, m \neq n}^{N} \gamma Q_c(o_m, s_n, a_n, g_m) - \mathbb{E}_{\hat{a} \sim \pi_\theta} \left[Q_c(o_m, s_n, \hat{a}, g_m) \right] \\ &= \sum_{m, m \neq n}^{N} \gamma Q_c \left(o_m, s_n, a_n, g_m \right) - \sum_{\hat{a}} \pi(o_n, \hat{a}, g_n) Q_c(o_m, s_n, \hat{a}, g_m). \end{aligned}$$

The vectors o and g denote all observations and goals of each agent respectively. Since we work in a decentralised manner the observations and goals have to be inferred. The complexity of inferring those strongly depends on the environments the agents have to act in. In this work we choose fairly simple environments in the observation space, which

leaves inference of those very simple as we can always assume the goal and observation, provided we assume knowledge of the world.
The final advantage formulation is a weighted mixture of the three previously introduced measures and using the GAE 2.3 equation we define

$$\delta_t = w_{\text{td}} A_{\text{td}} + w_{\text{credit}} A_{\text{credit}} + w_{\text{influence}} A_{\text{influence}}. \tag{3.7}$$

We propose that the mixture of state advantage is nudged by the influence estimate to lead to overall favorable outcomes for the whole population, while still trying to reach one's own goal.

3.3.4. Algorithm

We formulate the algorithm decentralised multi-goal multi-stage multi-agent reinforcement learning (DM3) Alg. 1 to learn in multi-agent scenarios. Only the second stage of learning is presented as the first stage is equivalent to the learning of PPO.
DM3 deploys a two stage curriculum learning approach using function augmentation. In the first stage, training is done according to PPO, where we use a separate value and policy network, instead of sharing parameters between the networks with separate heads. In the second stage a credit network is introduced. The policy, value and credit network use the pretrained parameters from the first stage, where both the credit and value network use the same parameters and are augmented as described in 3.3.1. After the initialization we begin with the training process. Next we conduct as many training episodes as are necessary or wished for. A training episode consists of two parts. We begin with collecting samples in the environment until a replay buffer is full. Once the buffer is filled we begin training. Training consists of several epochs and in each epoch we train all networks on minibatches sampled from the buffer until all samples have been used.

Algorithm 1: Decentralized multi-goal multi-stage multi-agent reinforcement learning (DM3) - Stage 2

Input : $\theta_{V^1}, \theta_{\pi^1}$, E number training episodes, N number agents, J number epochs
Output : π
Initialize networks $\theta_V, \theta_Q, \theta_\pi$ with stage 1 parameters $\theta_{V^1}, \theta_{V^1}, \theta_{\pi^1}$
Initialize Replay Buffer B
for *training episode* e *in* E **do**
  Assign goals to agents
  reset environment and get initial state s_0 and observation o_0
  for $t = 0$ *to* T **do**
    if *Replay buffer* B *is full* **then**
      break
    Sample actions $a_n^t \sim \pi(a_n^t | o_n^t, g_n; \theta_p i)$ for all agents
    Execute $\mathbf{a}$ and receive local rewards $\mathbf{r}_t$, observations $\mathbf{o}_{t+1}$, global state s_{t+1}
    Store $(s_t, \mathbf{o}_t, \mathbf{g}, \mathbf{a}_t, \mathbf{r}_t, s_{t+1}, \mathbf{o}_{t+1})$ into B
    $s_t \leftarrow s_{t+1}, \mathbf{o}_t \leftarrow \mathbf{o}_{t+1}$
  if *Replay buffer* B *is full* **then** conduct training
    Compute $A_{q,t}$ according to 2.3 where δ is computed according to 3.5
    Compute $A_{v,t}$ according to 2.3 where δ is computed according to 3.6
    Compute A_t according to 2.3 where δ is computed according to 3.7
    Compute $T_{q,t} = A_{q,t} + Q_t, \quad T_v = A_{v,t} + V_t$
    for *epoch 1 to J* **do**
      while B *not empty* **do**
        Sample Minibatch $(s_t, \mathbf{o}_t, \mathbf{g}, \mathbf{a}_t, \mathbf{r}_t, s_{t+1}, \mathbf{o}_{t+1}, A_t, T_{q,t}, T_{v,t})$ with K transitions without replacing from B, A, T_q, T_v
        Perform GD (Gradient Descent): $L(\theta_V) = \frac{1}{K} \sum_i^K (T_{v,i} - V(o_n^i, g_n; \theta_V))^2$
        Perform GD: $L(\theta_Q) = \frac{1}{K} \sum_i^K (T_{q,i} - Q(o_n^i, s_m^i, a_m^i, g_n; \theta_Q))^2 \quad \forall m \in N$
        Compute importance sampling weights: $\rho = \frac{\pi(a_n | o_n, g_n; \theta_\pi)}{\pi(a_n | o_n, g_n; \theta_{\pi,\text{old}})}$
        Compute entropy loss: $S(s_t)$
        Compute the clipped objective: L^{CLIP} according to 2.4
        Perform GD: $L(\theta) = L^{\text{CLIP}} + c * S(s_t)$
    Reset Buffer B

4. Experiments

In this section we present three experiments with variations. We use different environments to show how DM3 performs in varying reward structures. We give an introduction to each environment and define them regarding state, reward, goals and actions. We lay out the adaptations done to each algorithm in each environment. Then training and test results are shown and discussed for different scenarios.

4.1. Simulation of Urban Mobility (SUMO)

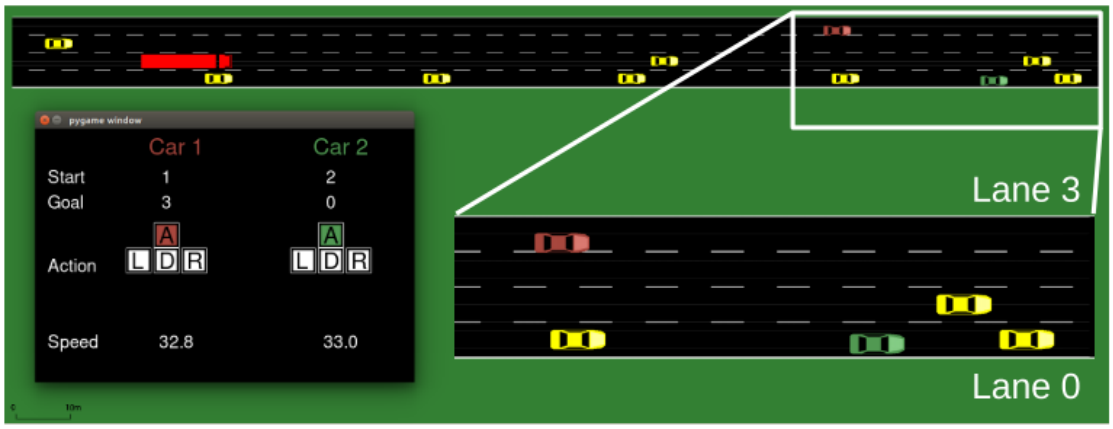


Figure 4.1.: The SUMO environment, where agents have to maneuver to a given goal lane and reach the end of the road without colliding with other agents. Each agent gets information regarding its own position, velocity and a local observation around its position [60].

The first task we present is the SUMO environment [60]. SUMO is a simulation framework for urban mobility, where vehicles can be controlled in custom traffic scenarios. In this environment an agent controls a vehicle and has to deal with other agents also learning simultaneously. SUMO offers sub-lane resolution, which allows to implement complex scenarios such as double merging as seen in [60]. We reuse the same scenarios in order to be comparable to CM3. Agents drive on a straight 200m long road, which consists of 4 lanes and is 12.8m wide. Vehicles start with an average speed of 30m/s and the simulation resolution is set to 0.2s. The state s is a vector with the (x, y) position, normalized by the length and width of the road to traverse, and the horizontal speed normalized by 29m/s. The self observation o_{self} is then the agents corresponding part of the state vector. The observation of others is a discretized tensor of shape [13,9,2] centered on the agent.
The action space is discrete and each agent has five actions available. No-op (preserving current velocity), accelerate, decelerate, shift one sub-lane to the left and shift one sub-lane to the right.
The goal is to reach the assigned sub-lane and travel a distance of 190m. We assign a static setup with probability 0.8 and a random start and goal sub-lane otherwise, During the static setup agent one starts on sub-lane 1, having to reach sub-lane 3, and agent two starts on sub-lane 2 and has to reach sub-lane 0. The reward of the agent is calculated each time step as follows: -1 for a collision; -10 if the number of maximal time steps (33) is reached; $10(1 - \Delta)$ for reaching 190m with a normalized sub-lane difference of Δ from the center of the goal lane; -0.1 if the speed is over 35.7m/s.
The episode terminates if the maximum number of time steps is reached or all agents reached the end of the road.
The SUMO merging task requires cooperation from both agents. An agent can not influence another agent, i.e. by occupying its lane or forcing a collision, without reducing its own reward. Due to this, agents egoistically maximizing their own reward may still converge towards a globally satisfying solution, even without the capability for coordination. We use this task nevertheless because it has been made available by the original authors and allows us to compare DM3 against CM3 and PPO.

4.1.1. Network Architectures

We use the tanh functions as activation function after each layer for PPO and DM3, while the CM3 implementation uses the ReLU function. An overview of the networks used by each algorithm is given in Table 4.1. We give a comprehensive overview of the layers used in 4.2 and give a longer description only for this task as the general structure regarding observations and states remains the same.

Algorithm	Value Net	Policy Net	Credit Net	Global Q Net
PPO	yes	yes	no	no
DM3	yes	yes	yes	no
CM3	no	yes	yes	yes

Table 4.1.: Overview of networks used by each algorithm.

The policy network π is the same for DM3, CM3 and PPO algorithms. During stage one the policy network feeds the concatenated self observation and goal to a layer with 32 units. The concatenated output of the first layer is connected to the layer h_1 with 64 units, which is connected to an output layer with 5 units, one for each action. for each environment and predicts the logit values that are input to a categorical distribution. In stage two, the input observation of the other agents o_{others} is fed into a convolutional layer with 4 filters of size 5x3 and stride 1x1. The resulting feature output is flattened and put into a layer with 64 units and subsequently connected to the layer h^1 of π.

The value network V used by DM3, PPO has the exact same layout as the policy network in both stages.

The stage one local Q_1 network as used by CM3 has the same net architecture as V but has additionally the action a_n as input concatenated to the own goal g and self observation o_{self}.

The Credit net version of Q_l is used by CM3 in stage two and reuses the stage one architecture of Q_1.

The Credit net version of DM3 augments V and gets an additional input in the form of action a_m as one-hot vector and o_m concatenated and feeds the input into two layers with 128 units and 64 units respectively. The last layer is then again connected to the layer h^1.

The global Q-network, only used by CM3, uses the concatenated actions a_{others} as input and o_{others} and then two layers with 128 and 64 units respectively. The last layer is then again connected to the layer h^1.

Algorithm	Stage)ne	Stage Two (Augmentation only)
Value Net	[32, 64, 1]	[Conv (2 channel, 4 filter, 5x3 size, 1x1 stride), 64]
Policy Net	[32, 64, 5]	[Conv (2 channel, 4 filter, 5x3 size, 1x1 stride), 64]
Credit Net (DM3)	[32, 64, 1]	[Conv (2 channel, 4 filter, 5x3 size, 1x1 stride), 64]
Global Q Net	[256, 256, 1]	[128, 128, 256]
Credit Net (CM3)	[256, 256, 1]	[128, 128, 256]

Table 4.2.: Layer sizes used for the SUMO environment for the curriculum learning.

4.1.2. Results and Discussion

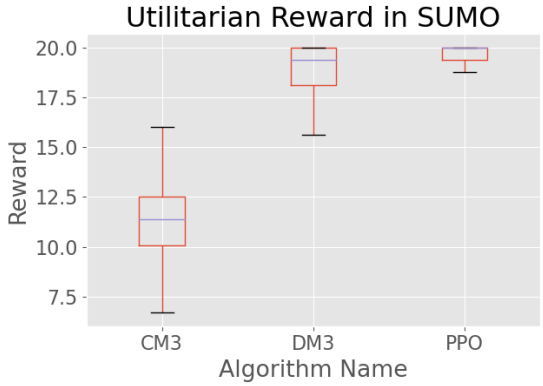


Figure 4.2.: Results of the CM3, DM3 and PPO algorithms in the SUMO environment task. Summed rewards of two agents, trying to reach an assigned goal lane. PPO solves the task in a flawless manner pretty much all the time. DM3 is just shy of PPO's result but also solves the task. CM3 does not manage to learn an efficient solution.

We present results for the SUMO merge task in Fig 4.2. Due to the structure of the state space, where observation of others are encoded with an image and self observations are a vector, we do not test a fully connected architecture and instead only test the curriculum learning setup in this environment. Furthermore, only results for the utilitarian rewards are

Algorithm	Stage One	Stage Two (Augmentation only)
Value Net	[64, 64, 64, 1]	[64, 64]
Policy Net	[64, 64, 64, 5]	[64, 64]
Credit Net (DM3)	[64, 64, 64, 1]	[64, 64]
Global Q Net	[64, 64, 1]	[128, 64]
Credit Net (CM3)	[64, 64, 1]	[128, 64]

Table 4.3.: Presented are the layer sizes used for the Cooperative Navigation environment used for both the curriculum learning and the fully connected architecture. The fully connected architecture simply omits the second network part.

shown because the other metrics do not provide additional insight beyond the utilitarian rewards in this case, but for the interested reader they can be found in the appendix B. Evaluation is performed by training 10 policies for each algorithm setup until convergence and then performing 100 evaluation runs using the static setup with each policy. PPO performs the best of the three algorithms and reaches the desired sublane in almost all cases. DM3 results are close in performance to PPO, whereas CM3 does not seem to learn well on this task. We reason that the task does not have a lot of interaction between the two agents and as a result we do not need social awarness or a global optimization to perform well. This leads us to conclude that due to the sparse interactions PPO is by far best suited in this task because it locally finds the best behavior. However, the performance of CM3 is also far below the reported score in the CM3 paper [60] and their reported performance is closer to the results of DM3. We did use the provided code and tried to replicate their results to the best of our knowledge. We did train all algorithms with neural networks of similar complexity and did not perform Hyperparameter tuning on any algorithm but used the provided default values, which are reported in appendix A. We conclude that the environment setup used in this experiment is too simple and no additional mechanisms are needed to learn cooperative or social behavior but instead one can simply optimize their behavior egoistically to achieve the best results even for the whole population. Nevertheless, DM3 learns good policies and proofs to be able to learn towards a goal.

4.2. Cooperative Navigation

This environment was introduced by Lowe et al. [28] and the specific scenarios were developed by Yang et al. [60]. In this environment each agent is a particle and goals are static landmarks. The world is unbounded and agents can move freely but experience contact forces during collisions. The world employs a model of inertia and friction, that allows agents to navigate. The agents have five actions available. No-op, accelerate down, up, left or right. The observation of each agent is its absolute position and its velocity. The observation of other agents is relative to ones own position.
Goals are landmarks and an agent receives exactly one goal position in 2D space it has to reach. The reward is a penalty based on the distance to the agents' goal and agents are punished for collisions. The reward each time step is thus computed as $r^t = |pos^t_{\text{agent}} - pos^t_{\text{goal}}| - c^t_{\text{agent}}$, where c^t_{agent} is the number of collisions of the agent during a single time step t. A collision is defined as two agents centers being closer than the sum of their radii. Each agent tries to reach a given location as fast as possible to collect the least amount of penalty. The episode ends once all agents have reached their goal or the maximum number of time steps (50) has been reached. A goal is reached if the distance to it is lower than 0.05.
We reuse three scenarios provided in the CM3 [60] paper and introduce one more scenarios to observe more different behaviors. The different tasks start configurations can be seen in Fig 4.3. The different scenarios all rely on cooperation of all agents to achieve the maximum summed up reward. Cooperation is needed because the most efficient path of all agents crosses the path of one or more other agents. To not make inefficient decisions the agents need to coordinate very precisely. Only when all agents follow the best cooperative routes, the overall reward is maximized. In contrast, if one agent were to act selfishly and other agents have to avoid it the global reward would go down in favor of one agent.

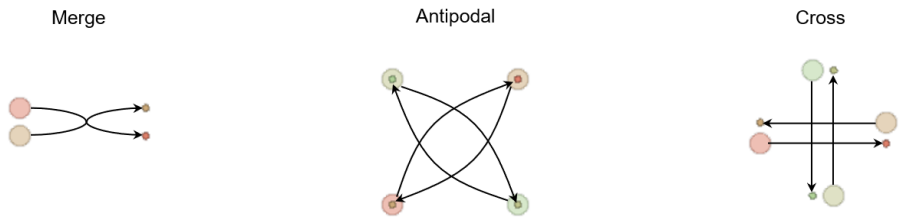
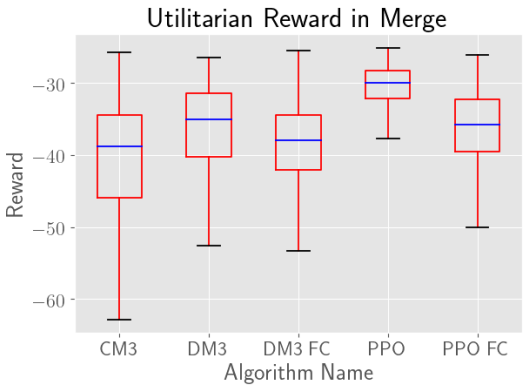


Figure 4.3.: These are the starting configurations of the cooperative navigation tasks merge, anitpodal and cross. Each agent has to reach exactly one goal in a given time frame. The arrows show which agent has to go to which goal. Agents are bigger circles and goals are small circles.

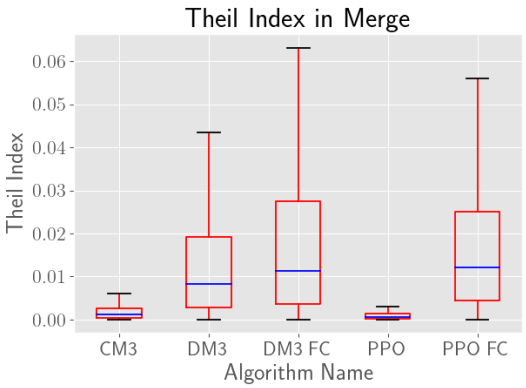
4.2.1. Network Architectures

We use the function augmentation architecture for CM3, DM3 and PPO to perform curriculum learning, where again the input is split into self observation and observation of others. We show the layers used for the different setups in Table 4.3. The biggest difference to SUMO is that we do not have to use a convolutional layer as no visual input is given. Additionally, for DM3 and PPO, we test a fully connected architecture, where the layers are the same as the stage one parameters of the function augmentation, but instead of the split input the complete concatenated observation is fed into the network.

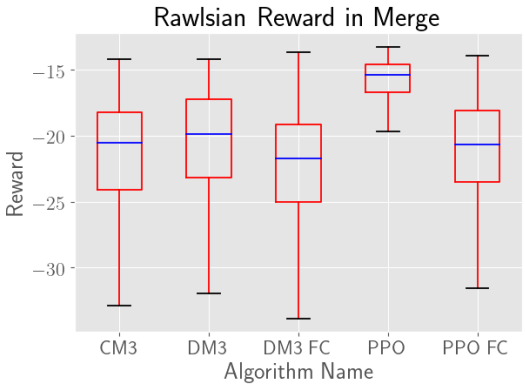
4.2.2. Merge - Results and Discussion



(a)



(b)



(c)

Figure 4.4.: Results of the CM3, DM3 and PPO algorithms for both network architectures in the merge scenario. (a) PPO with curriculum learning finds the best solution. (b) The Theil-index of DM3 shows that the found policies of DM3 do not find equally good solutions for all agents in terms of reward. (c) Only PPO with curriculum learning finds a solution, where both agents get optimal reward.

In this scenario two agents have to try to swap positions on a forward path to reach their respective goal, and need to learn to avoid collisions while doing so.
The merge scenario results of the evaluation can be seen in Fig 4.4 and the training is displayed in Fig 4.5. PPO converges the fastest followed by DM3 and then CM3. We suspect that PPO and DM3 converge faster because of the different techniques originally used by PPO, namely the generalized advantage estimate, objective clipping and importance sampling. In contrast, CM3 does not use any of the mentioned techniques and therefore it is not surprising that it takes longer to converge.
When looking at the results of the final trained policies PPO outperforms both DM3 and CM3 in all metrics, with DM3 being better than CM3 according to the utilitarian reward but worse using the Theil index. Using the curriculum learning approach performs better than the setup without curriculum learning for both PPO and DM3, telling us that this approach provides a net performance gain.

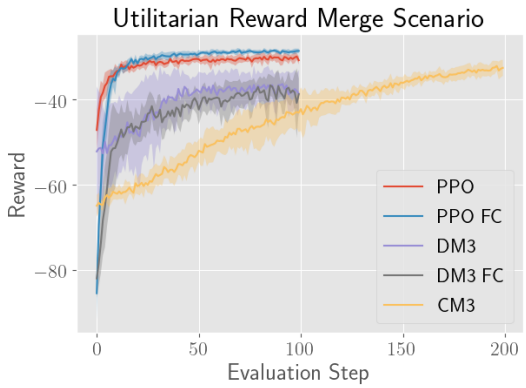


Figure 4.5.: Illustrated is the training graph in the merge scenario for CM3, PPO and DM3. CM3 takes significantly longer to converge than PPO and DM3. PPO converges fastest and the curriculum learning variants are more sample efficient than their direct learning counterparts.

Looking at the Theil Index that is achieved by all algorithms we can see that only PPO with curriculum learning and CM3 achieve high scores consistently. The curriculum learning approach is needed to find policies that act in a similar manner for both agents. DM3 and PPO without curriculum learning learn behaviors, where one agent dodges in a far more complex manner than necessary to avoid collisions. We reason that PPO without curriculum learning and DM3 in both variants show this behavior because the policies do not find the simplest solution but rather overcompensate for unnecessary maneuvers

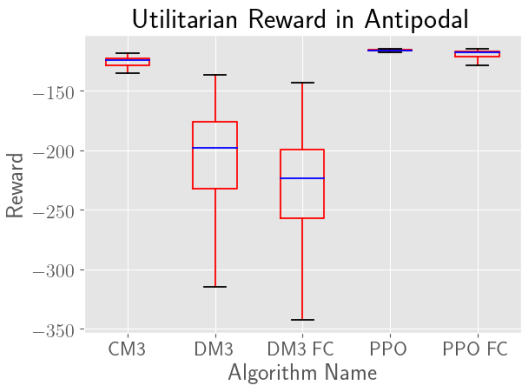


Figure 4.6.: Results of the DM3 and PPO algorithms for both network architectures in the circle spawn scneario. PPO consistently finds the best solutions and the resulting policies are more stable than the ones produced by DM3. Curriculum learning also significantly improved the stability and average results for both algorithms in comparison to the fully connected architecture.

from others. This leads to suboptimal behavior as the best solution requires trust from both agents and is rather unstable once one agent has diverged from it. PPO Agents ignore the incoming danger because of the pretraining and then stay on the best behavior. DM3's additional gradient terms make learning unstable by making the agents overreact to nearby agents and coordination cannot be achieved. CM3 on the other hand does not even have to start in a good local optimum since it finds the best outcome through the centralized optimization during the learning using global states.

4.2.3. Antipodal and Cross - Results and Discussion

We show the results of the antipodal and cross scenario together because the results are very similar. Additionally, we only look at the utilitarian reward as the distribution of the Theil-index and rawlsian metric lead to the same conclusions as in the merge scenario. The omitted figures can be found in the appendix B
In the antipodal scenario each agent starts in a corner of a square and has to reach the opposite side corner from where it started and in the cross scenario each agent starts on a side and has to go to the goal on the other side. The results for both scenarios can be seen in Fig 4.6. In this scenario PPO in both variants outperforms again both CM3 and

DM3. However, CM3 is very close in performance, whereas DM3 in both variants fails to learn optimal policies. CM3 is able to learn a very good solution by maximizing the global reward and is put into a good local optimum from the very beginning by the curriculum learning. PPO profits from the curriculum learning but surprisingly has very good results even without it. PPO quickly finds optimal solutions and through the stability of the updates does not diverge anymore from the good policy. With curriculum learning PPO is slightly more efficient in its movement because of the favorable initialization. Despite also employing curriculum learning DM3 does not manage to learn policies that are efficient at solving both tasks. This happens because the policies correct too much to avoid collisions and take very inefficient paths to reach their goal. Additionally, by not taking a similar route for all agents, learning becomes much harder through the non converging behavior of each agent. This makes it prohibitetively hard to learn once the behaviors start to prevent reaching their goal in an efficient manner. DM3 first breaks out of the highly symmetric situations by moving in a random direction. This creates a situation where all agents get to a chaotic situation from the symmetric spawn situation. From there on all agents try to reach their destined goal, while trying to avoid others. Even in trying to avoid others the behavior is overly cautious and evasion maneuvers are made. This behavior impacts the results heavily in a negative way as agents take a lot longer to reach their goals and agents receive punishment every time step. DM3 does show this behavior so because of the additional awareness of the other agents terms. In contrast, PPO does not care for other agents and simply optimizes for its own goal. Since this locally optimal behavior is also the best behavior possible for everyone because of the symmetric spawn configurations, PPO manages to learn the globally optimal policy despite only optimizing on local rewards. As a consequence despite the cooperative nature of the task, it is solvable by local optimization methods, rendering it much less useful to evaluate the merits of social awareness, as it is not needed in these scenarios.

4.3. Social Navigation

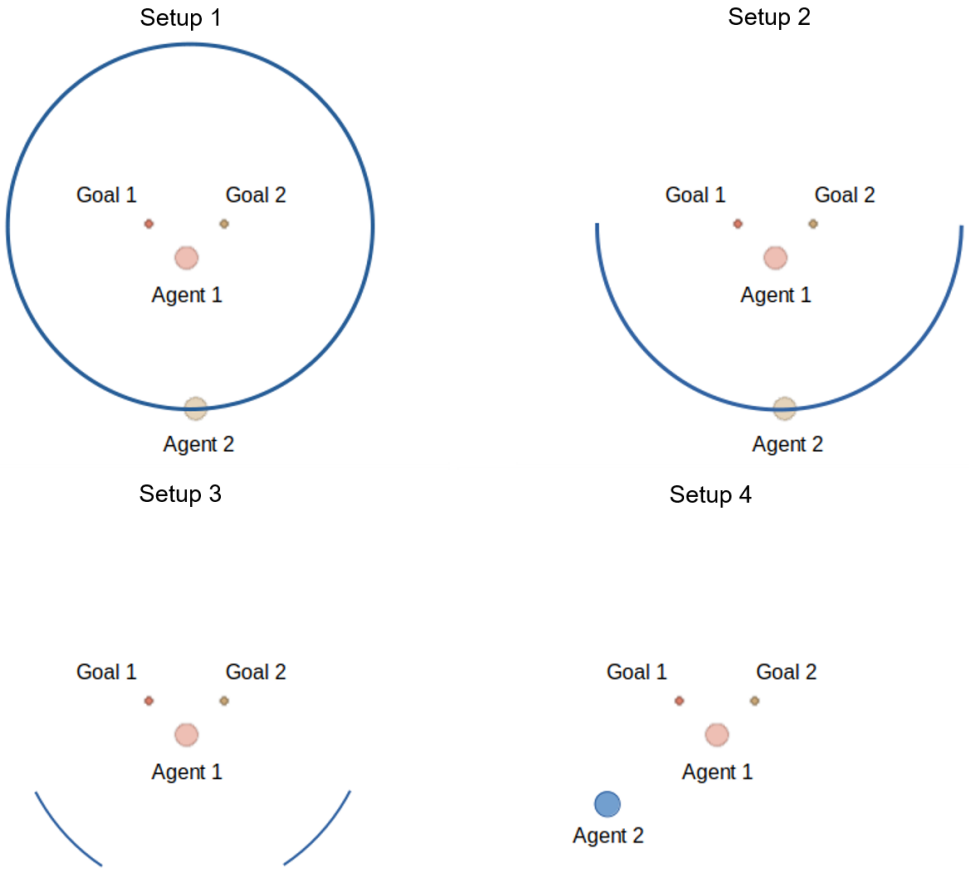


Figure 4.7.: These are the starting configurations of the common goal task in the co-operative navigation environment. The small circles indicate the goals the agents and the larger circles are the two agents. Agent 1 is positioned to have the exact same distance to both goal positions. Agent 2 is always further away from both goals than agent 1. The spawn of agent 2 is indicated by the blue line. In setup 4 the agent always spawns at the same spot, while in the other setups the agents spawn position may change between runs.

We introduce a new set of scenarios in this section that reuses the same environment as the cooperative navigation scenarios. The cooperative navigation scenarios suffered, like SUMO, from a reward structure that made locally optimal behavior also globally optimal. The new scenarios instead give one agent power over the other agent, where the powerful agent has to make a choice that does only affect the other agent but not itself. We show the basic setups we test in Fig 4.7. In all setups one agent spawns close to two goals with the same distance to each, while the other agent spawns somewhere much farther away around the two goals, giving the first agent power over the second. Rather than coordinating with other agents to find the most efficient path, it is now necessary for the first agent to act in a social manner.

We describe changes to the goal and reward structure, everything else has remained the same as the underlying framework has stayed the same. In the environment exist two agents and two goals. Each agent is allowed to go to either goal to claim it. Once a goal has been claimed it can not be claimed again and the agent that claimed the goal can not claim another goal. The observation o_n of agent n is its velocity, position and a flag that is one if the agent already claimed a goal and zero otherwise. Goals g are given as concatenated position vectors and an additional flag for each goal indicating, if it has been claimed already.
The reward penalty for each agent in this scenario is the distance to the closest goal, meaning that if the closest goal is claimed the agent will receive less reward as a consequence and has to navigate to a different goal. After an agent has claimed a goal it does not receive further distance penalty but can still get a penalty on collisions, which incentivizes the agent to not collide with other agents after finishing their own task and not obstruct others.
The episode terminates after each agent has claimed a goal or the maximum number of time steps (50) has been reached.

We argue that the combination of the different spawn setups alongside the new reward design prevents locally optimal solutions from always being globally optimal. The agent with power now has to make a choice, which of the two goals it should navigate to. The crucial point is that one goal is closer to the other agent than the other. We expect that an agent that does not care about the other agents' reward chooses randomly which goal to select or always selects the same one, disregarding from which direction the other agent is coming.

During training we use the predefined spawn scenarios with a probability of 0.8 and random goal and agent positions with a probability of 0.2. The randomization of the

start positions is done in the same manner as in the cooperative navigation scenarios and should help to avoid local minima during the optimization process.

In this environment we only evaluate the performance of PPO and DM3 as the code of CM3 did not provide a modular implementation that would have allowed to adapt to the different environment needs, without major adaptations to the existing code. However, according to the results of the last experiments PPO is a strong baseline for comparison and is sufficient to draw meaningful conclusions about the behavior of both algorithms, even without having CM3 available to compare.

4.3.1. Network Architectures

The network architectures are analogous to the ones used in the cooperative navigation scenarios and can be seen in Table 4.3. The input is constructed in the same manner but the goal observation vectors are adjusted to account for the additional input flags.

4.3.2. Setup 1 - Results and Discussion

In the first spawn setup the second agent spawns in a circle around the the center of the two goals. We show the training graph of PPO and DM3 in Fig 4.9 and use it examplary for all following scenarios. The fully connected architectures converge faster in these scenarios and yield higher results on average. The convergence speed of PPO is faster but plateus at a lower reward than DM3. Curriculum learning does not yield any advantage over the traditional approach.
The evaluation results can be seen in Fig 4.8. We trained 10 policies per setup and evaluated 100 times using each final policy. Evaluation episodes always use the configured setup that was used during training without any random configurations. Both DM3 and PPO achieve high results in the utilitarian reward. The fully connected variants of PPO and DM3 both outperform their counterpart using curriculum learning. Without curriculum learning DM3 yields the best results. The Theil index only for DM3 without curriculum learning is higher than the rest. This discrepancy is because of the frequency with which the different algorithms choose the cooperative goal, where both agents get the best reward. That is if agent two spawns to the right of agent one, agent one has to choose the left goal and vice versa to maximize the global return.

Since PPO only optimizes on local rewards agent one has no obvious signal to choose a goal based on the spawn position of agent one and explains the observation that PPO does not

Figure 4.8.: Results of the DM3 and PPO algorithms for both network architectures in the circle spawn scneario. Only DM3 without curriculum learning manages to consistently learn a globally optimal policy.

learn this assignment, whereas DM3 can adapt its policy. However, the curriculum learning approach of DM3 is significantly worse and does not learn a cooperative goal assignment as well. The cause of this result might be because of the different network architecture as everything else is exactly the same. The function augmentation scheme does not allow for more complex behaviors involving other agents in this scenario causing the DM3 with curriculum learning to fail to learn as good as the fully connected architecture. One possible reason could be that the architecture does not have enough depth after combining the last layer of the single agent mdp network with the other agent network, which leads to an inability to learn more complex behaviors in regard to the other agents. At last the

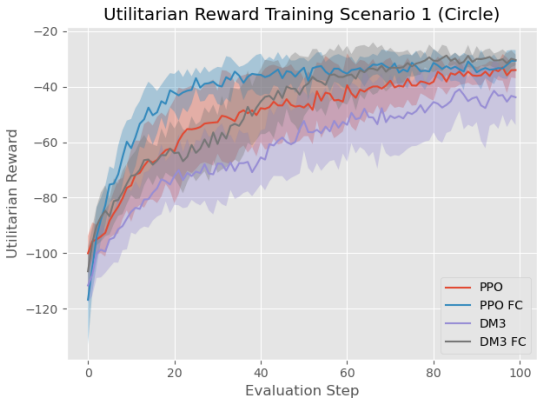


Figure 4.9.: Illustrated is the training graph in the first setup scenario, where the second agent spawns in a circle around the two goals, for PPO and DM3. DM3 takes longer to train than PPO and curriculum learning is faster than a direct learning approach.

rawlisian reward for DM3 without curriculum learning is also the highest because the second agent has pretty much always the closer goal available as can be seen in 4.8 (d), giving it more reward. Despite the favorable outcome the utilitarian rewards between PPO and DM3 are much closer than expected, showing that DM3 might learn social behavior, but is not optimal in its navigation to any goal, making the rewards much closer than they could be.

4.3.3. Setup 2 and 3 - Results and Discussion

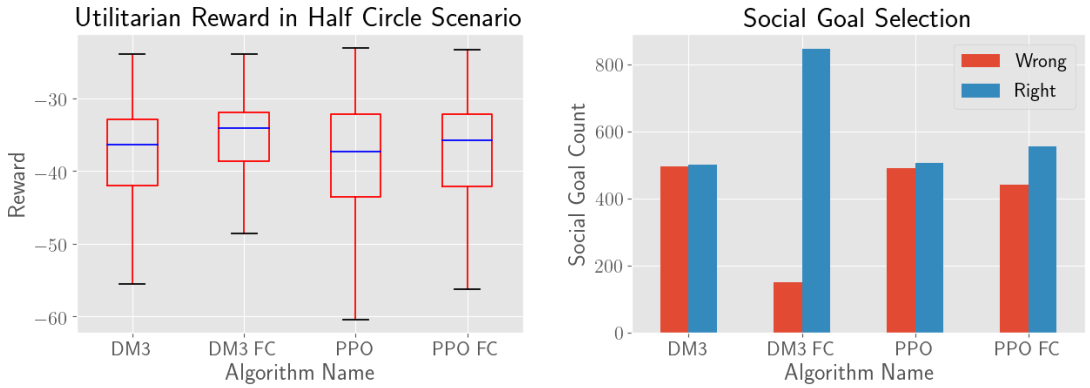


Figure 4.10.: Results of the DM3 and PPO algorithms for both network architectures in the half circle spawn scneario. The algorithms without state seperation into self and other observations outperform the splitted network architecture. DM3 has a lower Theil Index score regardless of the neural net architecture and DM3 outperforms PPO consistently.

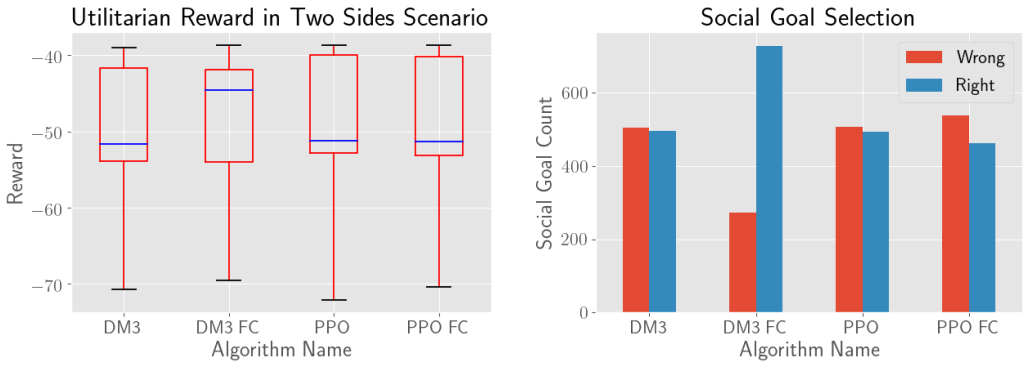


Figure 4.11.: Results of the DM3 and PPO algorithms for both network architectures in the two sides spawn scneario. DM3 without curriculum learning manages to occasionally learn to adapt to the second agent. All other algorithms do not learn social behavior.

We discuss and present the results of setup 2 and 3 together. In setup 2 the second agent spawns in the lower half circle and in the third setup the second agent spawns on one of two sides slighlty below the first agent.
The half circle spawn scenario results can be seen in Fig 4.10 and for the two sided spawn in Fig 4.11. The two stages setup performs for both PPO and DM3 worse than the fully connected single stage setup. With the fully connected architecture DM3 outperforms the PPO baseline and DM3 without curriculum learning. The Theil index and rawlsian reward are following for DM3 the best because again it finds the optimal solution with regard for both agents, which can be confirmed by looking at the actual ratio of with which the algorithms chose the cooperative goal.
These findings support the claim that DM3 is able to learn to take other agents into account and correctly predict its influence on others reward by computing the credit taken from their perspective. Again the curriculum learning approach could not learn the favorable outcomes even with credit assignment, evidencing that the way function augmentation is performed is hindering the ability to learn complex behaviors involving and reacting to other agents.

4.3.4. Setup 4 - Results and Discussion

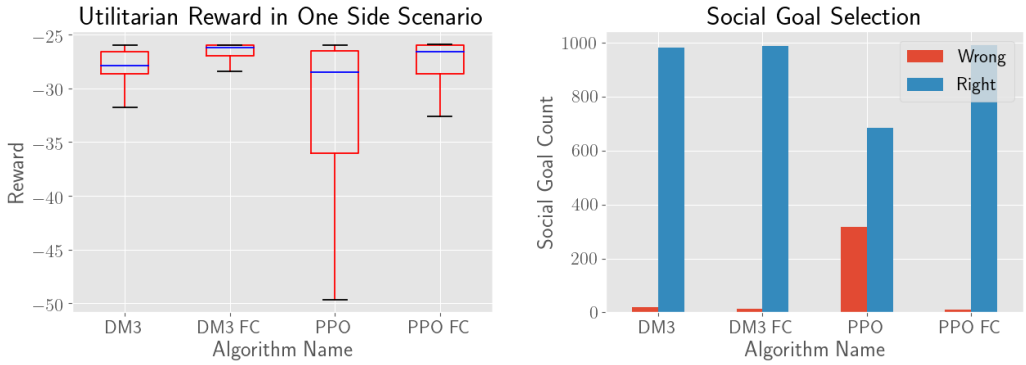


Figure 4.12.: Results of the DM3 and PPO algorithms for both network architectures in the one side spawn scenario. DM3 learns both with and without curriculum learning consistently to learn a globally satisfying solution. PPO with curriculum learning has problems to do so, while PPO without does not. In addition, curriculum learning finds worse local optima.

In the last scenario we discuss, the second agent always starts on one side at the same spot behind the first agent. The results can be seen in Fig 4.12. Every setup except PPO with curriculum learning manages to consistently solve the issue in a social manner. This result is interesting because intuitively the agent has no direct incentive to favor the globally optimal behavior. However, it seems that because the agent always spawns on one side, the first agent becomes biased to choose the globally optimal goal. We suspect that this happens because in the early training process the policies might collide on the side of the second agent, teaching the first agent to go to the other side. This hypothesis also explains why the curriculum learning variant of PPO does not pickup on the position of the second agent consistently, as collisions occur less often because the agents already go more efficiently for the goals. Since DM3 can learn in this task with curriculum learning it is evident that, DM3 does help to learn in direct comparison to PPO in such social settings. In contrast, the function augmentation is not working in the more complex scenarios, where an agent has to generalize depending on the start position of the other agent,

5. Discussion

We investigated how we can use a credit mechanism in multi-agent multi-goal reinforcement learning systems such that agents behave in a social way. We explored a decentralized approach that uses perspective taking to approximate influence on other agents' reward. To evaluate the system from different angles we used the Theil-index and the Rawlsian metric in conjunction with the commonly used utilitarian reward. We compared the existing algorithms CM3 and PPO to our novel approach DM3 and tested the impact of a curriculum learning approach using a function augmentation scheme as proposed by Yang et al. [60] in three experiments with different scenarios. As we found out, those scenarios feature highly symmetric setups in the cooperative navigation case or only sparse interactivity, such that locally optimal behavior is also globally optimal. We designed a new set of social navigation scenarios in the multi-agent particle environment, where an agent has influence over another agent without punishment or reward for their behavior. We introduced a new scenario based on the multi-agent particle environment, because to our knowledge there does not exist an available environment, which allows a distinct test of social behavior without many confounding factors that make interpreting results hard.

The social navigation scenario can successfully highlight the differences between locally optimal behavior and globally optimal behavior through social awareness. Cooperative behavior is not implicitly encouraged through the reward design and traditional single agent learning methods like PPO generally do not learn social behavior. In contrast, the existing environments reused from the CM3 paper showed a strong favor towards cooperative behavior and even further could be perfectly solved with selfish local optimization. In that regard our newly introduced environment offers a new test for social behavior in a multi-goal setting, albeit we encourage future work to incorporate tasks closer to real-world scenarios.

For the evaluation we introduced two new metrics from the field of social economics to get insights on the resulting policies from each algorithm. These metrics allowed us to draw conclusions beyond the sum of returns and helped identify weaknesses in the

local optimization of DM3. Given more complex scenarios, where the quality of an action or behavior is not clearly the best according to all metrics, one can retrieve much more information in such cases.

DM3 achieved promising results as it is able to show social behavior as can be seen in the social navigation experiments. It outperforms PPO in social settings, where the global optimization is not locally optimal. However, the found solutions were not as efficient in terms of navigation as the ones found by PPO. Despite being able to learn in most scenarios, if the situation is too complex i.e. a symmetric setup with multiple agents, then DM3 has problems to converge to a stable behavior and does not find a good policy. The performance of DM3 is comparable, albeit slightly worse, to PPO and superior to CM3 in the SUMO environment task and inferior in the cooperative navigation. DM3 shows that it can learn social and cooperative behavior through perspective taking but has challenges regarding convergence with concurrent learners.

The curriculum learning approach does help learning, when the interaction between agents was not complex and other agents do not have to be considered to reach globally optimal behavior. The approach is limited because of the function augmentation that did not allow for learning of more complex behavior. The results in the social navigation task also proved that the curriculum learning performs worse in constrast to the results achieved in the cooperative navigation task. We conclude that the function augmentation approach needs refinement for example by adding another layer after connecting the other network layer.

6. Outlook

The presented approach to decentralized multi-goal multi-agent reinforcement learning is a foundation for additional research in more complex environments. The decentralized components can be further studied in terms of their impact on learning. This work can be seen as a baseline of what is possible with the assumption of perfect inference capabilities of others and allows a modularized approach to test mental modelling systems of other agents. That said, this approach can conceptually be used in robotics, when the robot shares a workspace with humans. When living alongside humans robots will carry out their tasks not necessarily in a seperated environment. In that regard autonomous driving is one of the first applications that sees wider adoption. But even further in the future cooking alongside a robot or using the same workspace, while the robot pursues its task, will promote the integration of robots into human spaces. Our method helps bridge the conceptual gap from a robot that can only do either its task or requires full cooperation, to autonomous systems that can actually coexist without predefined behavior rules.
However, as we have seen in this work, it is important to be aware which kind of task is adressed and how it distinguishes between social cooperation and straightforward local optimization. It is necessary to compare non-socially aware algorithms and to make adaptations when needed. Our proposal of alternative measures to evaluate multi-agent performance will help to identify such tasks more easily. In general, we believe that future automated systems should consider explicit learning mechanisms, like the one proposed here, to inherently guarantee human-friendly behaviors without having to rely on completeness of some handcrafted explicit rules.

Bibliography

[1] Dario Amodei et al. “Concrete problems in AI safety”. In: *arXiv preprint arXiv: 1606.06565* (2016).

[2] Nicolas Anastassacos, Stephen Hailes, and Mirco Musolesi. “Partner selection for the emergence of cooperation in multi-agent systems using reinforcement learning”. In: 34.05 (2020), pp. 7047–7054.

[3] Kai Arulkumaran et al. “A brief survey of deep reinforcement learning”. In: *arXiv preprint arXiv:1708.05866* (2017).

[4] Ana LC Bazzan. “Opportunities for multiagent systems and multiagent reinforcement learning in traffic control”. In: *Autonomous Agents and Multi-Agent Systems* 18.3 (2009), pp. 342–375.

[5] Richard Bellman. “On the theory of dynamic programming”. In: *Proceedings of the National Academy of Sciences of the United States of America* 38.8 (1952), p. 716.

[6] Julian H Blau. “The Existence of Social Welfare Functions Author”. In: *Econometrica: Journal of the Econometric Society* 25.2 (1957), pp. 302–313.

[7] Moritz C Buehler and Thomas H Weisswange. *Theory of Mind based Communication for Human Agent Cooperation*. 2020, pp. 1–6.

[8] Lucian Buşoniu, Robert Babuška, and Bart De Schutter. “A comprehensive survey of multiagent reinforcement learning”. In: *IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews* 38.2 (2008), pp. 156–172. ISSN: 10946977. DOI: 10.1109/TSMCC.2007.913919.

[9] Karl Cobbe et al. “Phasic policy gradient”. In: *arXiv preprint arXiv:2009.04416* (2020).

[10] Yan Duan et al. “Benchmarking Deep Reinforcement Learning for Continuous Control”. In: *International Conference on Machine Learning* (2016), pp. 1329–1338. arXiv: arXiv:1604.06778v3.

[11] Jakob Foerster et al. “Counterfactual multi-agent policy gradients”. In: 32.1 (2018).

[12] Jakob Foerster et al. “Stabilising experience replay for deep multi-agent reinforcement learning”. In: (2017), pp. 1146–1155.

[13] Vittorio Gallese and Alvin Goldman. “Mirror neurons and the theory of mind rreading”. In: *Trends in Cognitive Sciences* 2.12 (1998), pp. 493–501.

[14] Robert M. Gordon. “Article Folk Psychology”. In: *Language* 1.2 (1986).

[15] Mance E. Harmon and Leemon C. Baird III. “Multi-player residual advantage learning with general function approximation”. In: *Wright Laboratory, WL/AACF, Wright-Patterson Air Force Base, OH* (1996), pp. 45433–7308. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.15.1565%7B%5C&%7Drep=rep1%7B%5C&%7Dtype=pdf.

[16] John C . Harsanyi. “Cardinal Utility in Welfare Economics and in the Theory of Risk-taking”. In: *The Journal of Political Economy* 61.5 (1953), pp. 434–435.

[17] John C. Harsanyi. “Cardinal Welfare , Individualistic Ethics , and Interpersonal Comparisons of Utility”. In: *The Journal of Political Economy* 63.4 (1955), pp. 309–321.

[18] Matthew Hausknecht and Peter Stone. “Deep recurrent q-learning for partially observable MDPs”. In: *AAAI Fall Symposium - Technical Report* FS-15-06 (2015), pp. 29–37. arXiv: 1507.06527.

[19] Hoda Heidari et al. “Fairness behind a veil of ignorance: A welfare analysis for automated decision making”. In: *arXiv preprint arXiv:1806.04959* (2018).

[20] Pablo Hernandez-Leal, Bilal Kartal, and Matthew E Taylor. “Is multiagent deep reinforcement learning the answer or the question? A brief survey”. In: *Learning* 21 (2018), p. 22.

[21] Pablo Hernandez-Leal, Bilal Kartal, and Matthew E. Taylor. *A survey and critique of multiagent deep reinforcement learning*. Vol. 33. 6. Springer US, 2019, pp. 750–797. ISBN: 1045801909. DOI: 10.1007/s10458-019-09421-1. arXiv: 1810.05587. URL: https://doi.org/10.1007/s10458-019-09421-1.

[22] Sepp Hochreiter and Jürgen Schmidhuber. “Long Short-Term Memory”. In: *Neural Computation* 9.8 (1997), pp. 1735–1780. ISSN: 08997667. DOI: 10.1162/neco.1997.9.8.1735.

[23] Baird Iii et al. “Advantage Updating”. In: *Security* (1993).

[24] Natasha Jaques et al. “Social influence as intrinsic motivation for multi-agent deep reinforcement learning”. In: (2019), pp. 3040–3049.

[25] Rawls. John. *A theory of social justice*. Vol. 4. Harvard university press, 2009. ISBN: 9781317853251.

[26] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Communications of the ACM* 60.6 (June 2017), pp. 84–90. ISSN: 15577317. DOI: 10.1145/3065386.

[27] Shihui Li et al. "Robust Multi-Agent Reinforcement Learning via Minimax Deep Deterministic Policy Gradient". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 33 (2019), pp. 4213–4220. ISSN: 2159-5399. DOI: 10.1609/aaai.v33i01.33014213.

[28] Ryan Lowe et al. "Multi-agent actor-critic". In: *Nips* Nips (2017), pp. 6379–6390. ISSN: 00232173. DOI: 10.1007/BF01744832. arXiv: 1706.02275. URL: https://papers.nips.cc/paper/7217-multi-agent-actor-critic-for-mixed-cooperative-competitive-environments%7B%5C%%7D0Ahttp://arxiv.org/abs/1706.02275.

[29] Ryan Lowe et al. "Multi-agent actor-critic for mixed cooperative-competitive environments". In: *Advances in Neural Information Processing Systems* 2017-Decem (2017), pp. 6380–6391. ISSN: 10495258. arXiv: 1706.02275.

[30] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *Nature* 518.7540 (2015), pp. 529–533. ISSN: 14764687. DOI: 10.1038/nature14236. URL: http://dx.doi.org/10.1038/nature14236.

[31] Volodymyr Mnih et al. "Playing Atari with Deep Reinforcement Learning". In: (2013), pp. 1–9. arXiv: 1312.5602. URL: http://arxiv.org/abs/1312.5602.

[32] Seyed Sajad Mousavi, Michael Schukat, and Enda Howley. "Deep Reinforcement Learning: An Overview". In: (June 2018). DOI: 10.1007/978-3-319-56991-8_32. arXiv: 1806.08894. URL: http://arxiv.org/abs/1806.08894%20http://dx.doi.org/10.1007/978-3-319-56991-8%7B%5C_%7D32.

[33] OpenAI et al. "Dota 2 with Large Scale Deep Reinforcement Learning". In: (2019). arXiv: 1912.06680. URL: http://arxiv.org/abs/1912.06680.

[34] Afshin OroojlooyJadid and Davood Hajinezhad. "A review of cooperative multi-agent deep reinforcement learning". In: *arXiv preprint arXiv:1908.03963* (2019).

[35] Julien Perolat, Charles Beattie, and Karl Tuyls. "A multi-agent reinforcement learning model of common-pool resource appropriation". In: Nips (2017).

[36] Jan Peters and Stefan Schaal. “Reinforcement learning of motor skills with policy gradients”. In: *Neural Networks* 21.4 (2008), pp. 682–697. ISSN: 08936080. DOI: 10.1016/j.neunet.2008.02.003.

[37] Roberta Raileanu et al. “Modeling others using oneself in multi-agent reinforcement learning”. In: *35th International Conference on Machine Learning, ICML 2018* 10 (2018), pp. 6779–6788. arXiv: 1802.09640.

[38] Tabish Rashid et al. “QMIX: Monotonic value function factorisation for deep multi-agent reinforcement Learning”. In: *35th International Conference on Machine Learning, ICML 2018* 10 (2018), pp. 6846–6859. arXiv: 1803.11485.

[39] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. “Learning internal representations by error propagation.” In: *MIT Press, Cambridge, MA* 1.V (1986), pp. 318–362. URL: https://apps.dtic.mil/docs/citations/ADA164453.

[40] Gavin A Rummery and Mahesan Niranjan. *On-line Q-learning using connectionist systems*. Vol. 37. University of Cambridge, Department of Engineering Cambridge, UK, 1994.

[41] Mikayel Samvelyan et al. “The StarCraft multi-agent challenge”. In: *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS* 4.NeurIPS (2019), pp. 2186–2188. ISSN: 15582914. arXiv: 1902.04043.

[42] Tuomas W Sandholm and Robert H Crites. “Multiagent Reinforcement Learning in the Iterated Prisoner ’ s Dilemma”. In: *Biosystems* 37.1-2 (1996), pp. 147–166.

[43] Tom Schaul et al. “Prioritized experience replay”. In: *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings* (2016), pp. 1–21. arXiv: 1511.05952.

[44] John Schulman et al. “High-dimensional continuous control using generalized advantage estimation”. In: *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings* (2016), pp. 1–14. arXiv: 1506.02438.

[45] John Schulman et al. “Proximal policy optimization algorithms”. In: *arXiv preprint arXiv:1707.06347* (2017).

[46] John Schulman et al. “Trust region policy optimization”. In: *32nd International Conference on Machine Learning, ICML 2015* 3 (2015), pp. 1889–1897. arXiv: 1502.05477.

[47] David Silver et al. “A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play”. In: *Science* 362.6419 (2018), pp. 1140–1144. ISSN: 10959203. DOI: 10.1126/science.aar6404.

[48] David Silver et al. “Deterministic policy gradient algorithms”. In: *31st International Conference on Machine Learning, ICML 2014* 1 (2014), pp. 605–619.

[49] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An Introduction*. MIT press, 2018. ISBN: 0262193981. DOI: 10.1179/175622708X282910.

[50] Ming Tan. “Multi-agent reinforcement learning: Independent vs. cooperative agents”. In: (1993), pp. 330–337.

[51] Covid Tesau and Gerald Tesau. “Temporal Difference Learning and TD-Gammon”. In: *Communications of the ACM* 38.3 (1995), pp. 58–68. ISSN: 15577317. DOI: 10.1145/203330.203343.

[52] Gerald Tesauro. “TD-Gammon, a Self-Teaching Backgammon Program, Achieves Master-Level Play”. In: *Neural Computation* 6.2 (1994), pp. 215–219. ISSN: 0899-7667. DOI: 10.1162/neco.1994.6.2.215.

[53] Henri Theil. *Economics and information theory*. Tech. rep. 1967.

[54] Hado Van Hasselt, Arthur Guez, and David Silver. “Deep reinforcement learning with double Q-Learning”. In: *30th AAAI Conference on Artificial Intelligence, AAAI 2016* (2016), pp. 2094–2100. arXiv: 1509.06461.

[55] Oriol Vinyals et al. “Grandmaster level in StarCraft II using multi-agent reinforcement learning”. In: *Nature* 575.7782 (2019), pp. 350–354. ISSN: 14764687. DOI: 10.1038/s41586-019-1724-z. URL: http://dx.doi.org/10.1038/s41586-019-1724-z.

[56] Ziyu Wang et al. “Dueling Network Architectures for Deep Reinforcement Learning”. In: *33rd International Conference on Machine Learning, ICML 2016* 4.9 (2016), pp. 2939–2947. arXiv: 1511.06581.

[57] Peter Watkins, Christopher JCH and Dayan. “Q-Learning”. In: *Machine learning* 8.3-4 (1992), pp. 279–292. DOI: 10.4018/978-1-59140-993-9.ch026.

[58] Ronald J. Williams. “Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning”. In: *Machine Learning* 8.3 (1992), pp. 229–256. ISSN: 15730565. DOI: 10.1023/A:1022672621406.

[59] Heinz Wimmer and Josef Perner. “Beliefs about beliefs: Representation and constraining function of wrong beliefs in young children’s understanding of deception”. In: *Cognition* 13.1 (1983), pp. 103–128.

[60] Jiachen Yang et al. “Cm3: Cooperative multi-goal multi-stage multi-agent reinforcement learning”. In: *arXiv preprint arXiv:1809.05188* (2018).

[61] Kaiqing Zhang, Zhuoran Yang, and Tamer Başar. “Multi-agent reinforcement learning: A selective overview of theories and algorithms”. In: *arXiv preprint arXiv: 1911.10635* (2019).

[62] Kaiqing Zhang et al. “Fully decentralized multi-agent reinforcement learning with networked agents”. In: *35th International Conference on Machine Learning, ICML 2018* 13 (2018), pp. 9340–9371. arXiv: 1802.08757.

A. Hyperparameter Configurations

PPO

gamma: 0.99
max grad norm: 0.5
clip coef: 0.2
batch size: 512
n minibatch: 8
minibatch size: 64
gae lambda: 0.95
ent coef: 0.01
epochs: 5

DM3

gamma: 0.99
max grad norm: 0.5
clip coef: 0.2
batch size: 512
n minibatch: 8
minibatch size: 64
gae lambda: 0.95
ent coef: 0.01
epochs: 5
td advantage: 1.0
credit advantage: 0.2
influence advantage: 0.8

B. Results Tables

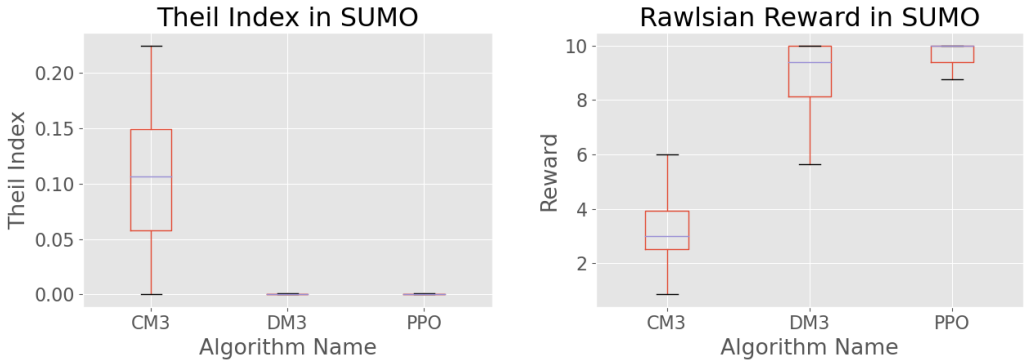


Figure B.1.: Results of the CM3, DM3 and PPO algorithms in the SUMO environment task. The Theil-index results show that DM3 and PPO always achieves the same reward for both agents, while CM3 does not. The Rawlsian reward plot illustrates that CM3 the worse agent does not reach the desired goal lane most of the time.

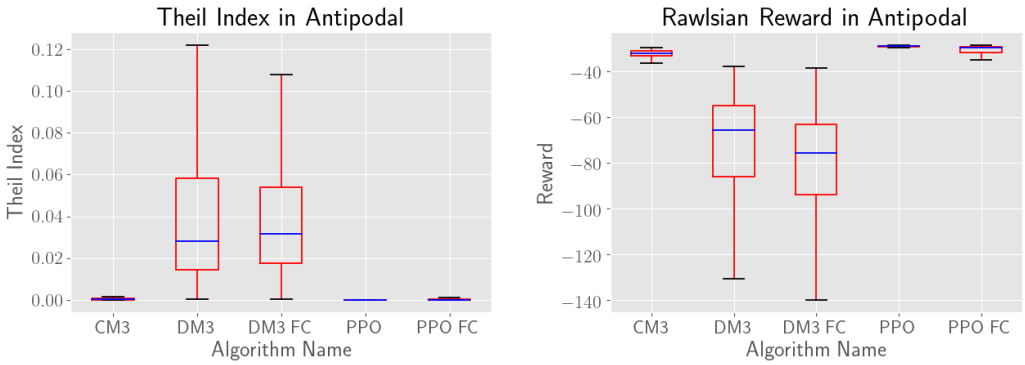


Figure B.2.: Results of the CM3, DM3 and PPO algorithms in the antipodal task. DM3 does not learn behaviors that benefits agents equivalently and the worst agents of DM3 are much worse than the ones of PPO or CM3.

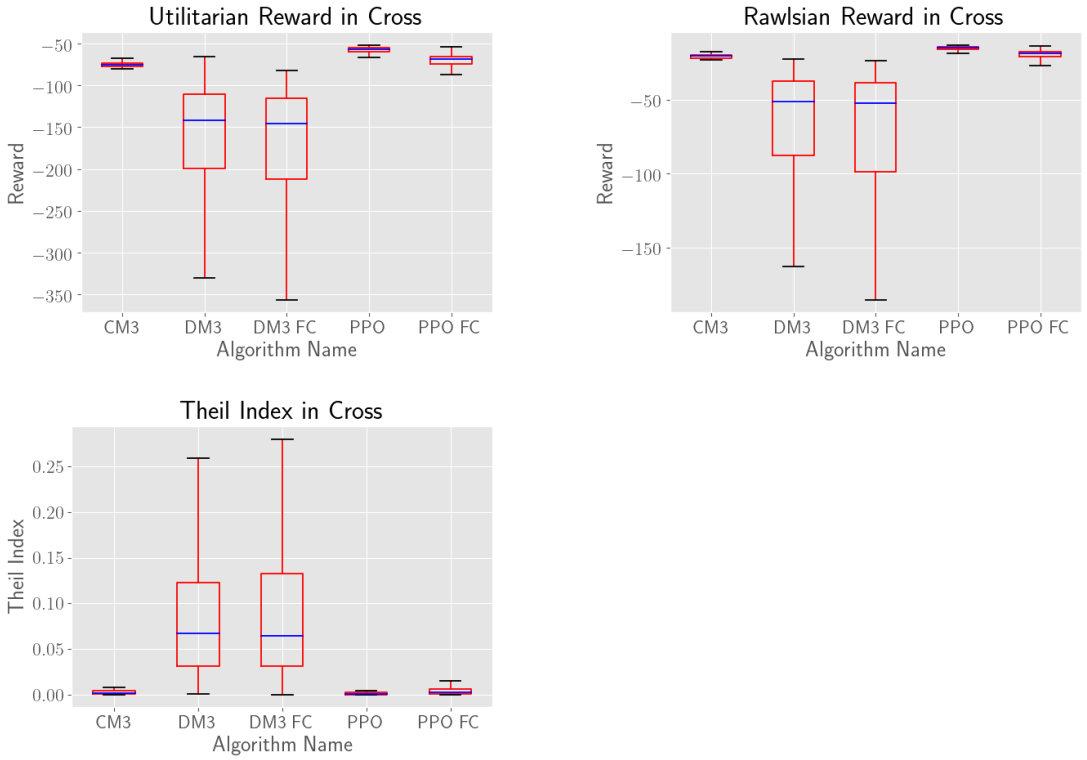


Figure B.3.: Results of the CM3, DM3 and PPO algorithms in the cross task. DM3 performs significantly worse than CM3 and PPO. PPO finds slighlty better solutions than CM3.

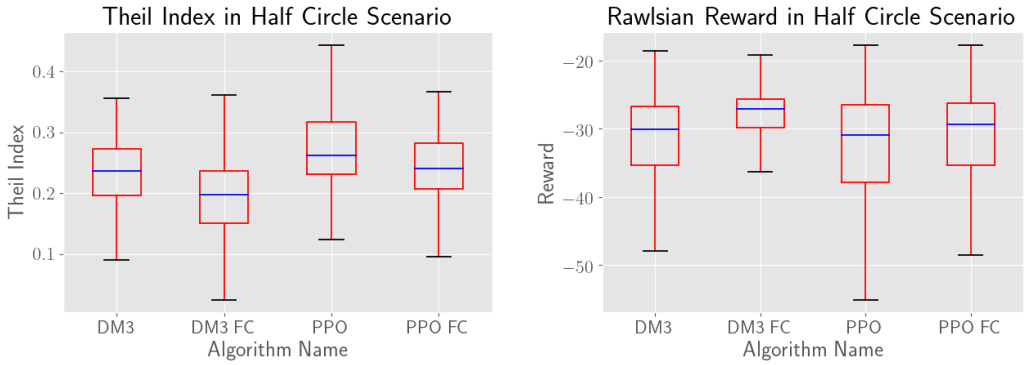


Figure B.4.: Results of the DM3 and PPO algorithms in the Setup 2 task, where the second agent spawns in a half circle. DM3 without curriculum learning using a fully connected architecture finds solutions, where agents receive more equally distributed rewards and the worse agent is better off on average.

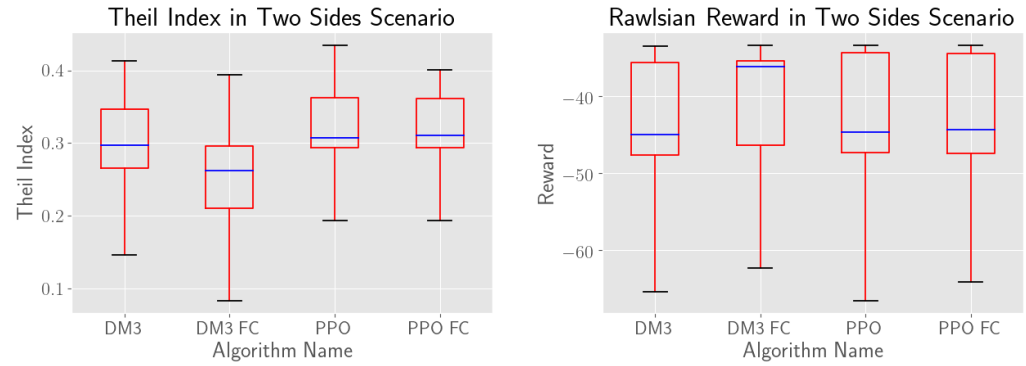


Figure B.5.: Results of the DM3 and PPO algorithms in the Setup 2 task, where the second agent spawns on either side. Again DM3 without curriculum learning using a fully connected architecture finds solutions, where agents receive more equally distributed rewards and the worse agent is better off on average.

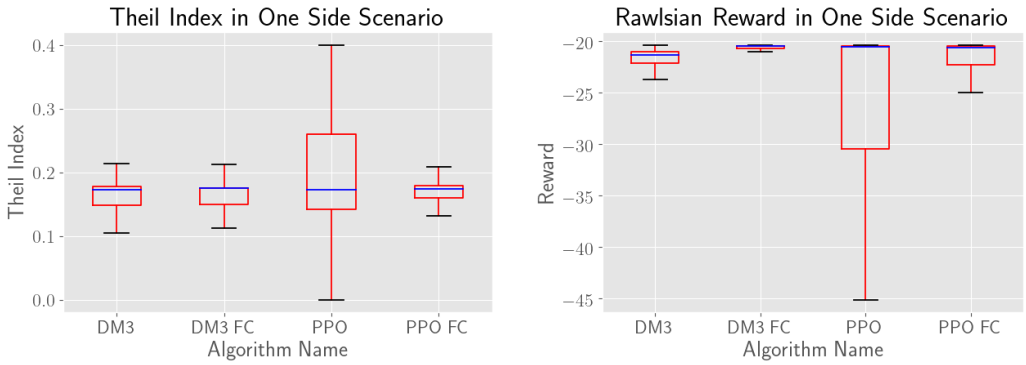


Figure B.6.: Results of the DM3 and PPO algorithms in the Setup 2 task, where the second agent always spawns on the same side. PPO shows large instabilities in its resulting rewards for the worst agent, which causes the reward distribution to be more unequal than the other algorithm setups.