Deep Reinforcement Learning with Relational Forward Models: Can Simpler Methods Perform Better?

Niklas Höpner

Master of Science
Artificial Intelligence
School of Informatics
University of Edinburgh
2020

Abstract

Learning in multi-agent systems is difficult due to the multi-agent credit assignment problem. Agents must obtain an understanding how their reward depends on ther other agents actions. One approach to encourage coordination between a learning agent and teammate agents is augmenting deep reinforcement learning algorithms with an agent model. Relational forward models (RFM) (Tacchetti et al., 2019), a class of recurrent graph neural networks, try to leverage the relational inductive bias inherent in the set of agents and environment objects to provide a powerful agent model. Augmenting an actor-critic algorithm with action predictions from a RFM has been shown to improve the sample efficiency of the actor-critic algorithm. However, it is an open question whether agents equipped with a RFM also achieve a fixed level of reward in shorter computation time compared to agents without an embedded RFM. Since utilizing a RFM introduces a computational overhead this must not be the case. To investigate this question, we follow the experimental methodology of Tacchetti et al. (2019) but additionally look at the computation time of the learning algorithm with and without an agent model. Further, we look into whether replacing the RFM with a computationally more efficient conditional action frequency (CAF) model, provides an option for a better trade-off between sample efficiency and computational complexity. We replicate the improvement in sample efficiency caused by incorporating an RFM into the learning algorithm on one of the proposed environments from Tacchetti et al. (2019). Further we show that in terms of computation time an agent without an agent model achieves the same reward up to ten hours faster. This suggests that in practice only reporting the sample efficiency gives an incomplete picture of the properties of any newly proposed algorithm. It should therefore become standard to report both, results on the sample efficiency of an algorithm as well as results on the computation time needed to achieve a fixed reward.

Acknowledgements

I would like to thank Prof. Albrecht and Arrasy Rahman, who did a fantastic job advising and guiding me during the project. I can only recommend everyone to write their thesis with the agents group at the University of Edinburgh, they integrate you early on into their research group and give regular feedback. For anyone looking to continue with further research after the thesis it is an invaluable experience.

Table of Contents

1 Introduction **1**

2 Related Work **4**

2.1 Multi-agent systems and agent modelling 4

2.2 Deep reinforcement learning 5

2.3 Agent modelling with DRL 7

2.4 Relational forward models 7

3 Methodology **10**

3.1 Environments 10

3.2 Pretraining of teammate agents 11

3.2.1 Policy architecture and training 12

3.3 Supervised training of agents models 13

3.4 Training of agents with an agent model 14

3.4.1 Measuring Computation Time 15

4 Results **17**

4.1 Pretraining of teammate agents 17

4.1.1 Analysis of trained teammates 17

4.2 Action prediction performance of agent models 18

4.3 Analysis of computation time and sample efficiency 20

4.3.1 Computation time 20

4.3.2 Cooperative Navigation 22

4.3.3 Stag Hunt 23

5 Conclusions **25**

Bibliography **27**

Chapter 1

Introduction

Many real world scenarios, such as traffic light control (Van der Pol and Oliehoek, 2016) and financial markets, can formally be modelled as multi-agent systems (MAS). With the success of deep reinforcement learning (DRL) for single-agent problems on domains such as game-play (Silver et al., 2017; Mnih et al., 2013) and robotic control (Lillicrap et al., 2016), it was just a question of time until DRL algorithms were adapted to multi-agent problems. Over the last few years a plethora of methods has emerged that generally can be structured into two lines of work. Settings in which every agent in the environment can be controlled and therefore take the role of a learning agent (Lowe et al., 2017a; Rashid et al., 2018) and settings in which an agent needs to coordinate with teammates or opponents that follow a policy unknown to the agent (Raileanu et al., 2018; He and Boyd-Graber, 2016). This thesis deals with the latter line of research building up on recent work that leverages graph neural networks for agent modelling (Tacchetti et al., 2019).

Instead of discussing a new method for tackling agent modelling in MAS, we focus on a methodological problem present in all of reinforcement learning research. The two most prominent evaluation metrics for reinforcement learning algorithms are the final reward achieved by the trained agent and the sample efficiency. Recently a set of further metrics was introduced Chan et al. (2020) to assess the reliability of reinforcement learning algorithms. A metric that is often missing from papers, but in practice often plays the most important role, is the computation time needed to train a policy with satisfactory final performance. In a business or research context one often faces a deadline. The question at hand is then what reinforcement learning algorithm gives the best performance in the time given. To choose from the set of existing algorithms one must be aware of how they compare in the computation time needed to achieve a given level of performance. Since the computation time of a newly proposed method is often not discussed in papers, it makes it impossible for someone to figure out whether this method is feasible for a given context.

To show that not reporting computation time can give a misleading idea of the usefulness of a reinforcement learning algorithm, this thesis takes a closer look at the performance of the RFM in the context of agent modelling for MAS. The RFM (Tacchetti et al., 2019), a recurrent graph neural network, has been shown to outperform other neural network based agent models. It achieves a higher action prediction accuracy when trained on episodes produced by agents with a fixed policy. Integrating the predicted actions of the RFM into the policy of the learning agent makes the learning algorithm more sample efficient i.e. fewer environment steps are needed to achieve the same level of reward. However, the final performance of the policy obtained by augmenting the input with action predictions does not differ from the final performance of the policy without the additional information of action predictions.

Having an agent model for action prediction adds a computational cost to the overall learning algorithm since action predictions must be computed before the policy is run. Whether this additional computation is negligible or forms a computational bottleneck depends on the setting. In general any reinforcement learning algorithm applied to agent modelling in MAS can be structured into three computational parts. The simulation of the environment, computation related to the policy (inference and updating) and computation related to the agent model (inference and updating). If either the policy computations or the environment simulations are the computational bottleneck of the learning algorithm, adding an agent model will have little effect on the overall training time. However, for the setting provided by Tacchetti et al. (2019) we will show that the agent model needs as much computation time as the policy itself, while the environment simulation is negligible. Further, simpler agent models such as CAF models could provide an alternative to having no agent modelling at all. Conditional action frequency models could be a an elegant solution to maintain some of the improvement in terms of sample efficiency that an agent model provides without introducing a computational bottleneck.

The thesis investigates the hypothesis that training a policy augmented with action predictions from a RFM *despite being more sample efficient* needs *more computation time* to achieve the same reward compared to training a policy without input from an agent model. Additionally, we hypothesise that augmenting a policy with input from a simpler agent model needs *less computation time* to achieve a given level of reward compared to having no agent model at all or making use of an RFM.

We are able to show that augmenting a learning algorithm with action predictions from an RFM takes up to 10 hours longer to achieve a fixed level of reward, compared to not utilising the RFM. The results of CAF models are inconclusive and more experiments must be conducted to look into the second hypothesis.

First, an overview over recent progress in combining deep reinforcement learning algorithms with agent modelling for learning in multi-agent settings is given, followed by a description of RFMs (Tacchetti et al., 2019). The methodology is divided into three parts and follows Tacchetti et al. (2019) to provide an optimal comparison. The complete analysis is done for two different gridworld environments (Tacchetti et al., 2019). First, teammate policies are trained. Those teammate policies are used later to have non-learning agents in the environment with which the controlled agent needs to learn to coordinate. To assess the predictive capabilities of different agent models, we train them on episodes produced using the teammate policies and look at their action prediction accuracy. In the last part a learning agent is trained to coordinate with a pretrained agent. The policy from the learning agent either receives no additional input, action predictions from the RFM or action predictions from a CAF model. For each part, the conducted experiments are explained. Finally, the results are presented. The conclusion discusses limitations and possibilities for future work. The code to reproduce the presented experiments is publicly available at *https://github.com/NikeHop/thesis*.

Chapter 2

Related Work

We begin by introducing MAS and their properties, before discussing CAF models. The main goal of the thesis is to emphasise the importance of analysing the computational properties of newly developed DRL algorithms. Therefore, an overview over relevant DRL algorithms is given. The next section looks at approaches to integrate an agent model into existing DRL algorithms. We also contrast the field of multi-agent-reinforcement learning (MARL) with the setting of a single learning agent in a MAS. Finally the architecture of the RFM is discussed in detail.

2.1 Multi-agent systems and agent modelling

MAS describe settings in which multiple agents act in a shared environment. Each agent has its own observation, actions and reward and the state dynamics and reward may depend on actions of all agents. MAS come with additional challenges compared to single agent problems. The presence of other potentially learning agents leads to nonstationary environment dynamics (Papoudakis et al., 2019) and breaks the Markov assumption that the state at the next timestep only depends on the current state and the agents action. Another issue is the credit assignment problem. For a single agent problem it only has a temporal dimension, but in a MAS the agent must disambiguate how its reward depends on other agents actions (Nguyen et al., 2018).

One approach to deal with the nonstationarity and multi-agent credit assignment problem is to explicitly model other agents. An agent model takes information about past trajectories of other agents as input and outputs an estimate about the modelled agent such as a type, its next action or preferences (Albrecht and Stone, 2018). Albrecht and Stone (2018) give an overview of different methods to model agents. One of them are (CAF) models. They are considered part of the larger class of methods known as policy reconstruction methods, that aim at reconstructing an agents decision making by fitting a model of the agents policy too its observed behaviour. A CAF model maintains a frequency distribution over an agents actions for each of the modelled agents. It can be conditioned on different information such as characteristics of the state. Determining what information to condition on, is the central challenge and methods to automatise the conditioning decision at least partially have been proposed (Chakraborty and Stone, 2014).

2.2 Deep reinforcement learning

Deep reinforcement learning originates from the idea of using deep neural networks to parameterize either the action value function or the policy of the reinforcement learning problem. The most prominent action-value method is DQN (Mnih et al., 2013), which builds up on Q-learning (Watkins and Dayan, 1992) by parameterizing the action value function via a deep neural network. Here we will focus on policy gradient methods (Williams, 1992), since the learning algorithm employed by Tacchetti et al. (2019) belongs to this class. Let $J(\theta) = E_{\pi_\theta}[r(\tau)]$ denote the expected return following policy π_θ parameterized by θ in an underlying MDP. Here τ refers to a trajectory in the MDP, i.e. a sequence of states, actions and rewards ($\tau_i = (s_0, a_1, r_1, s_1, a_2, r_2, ...)$) and $r(\tau)$ refers to the associated return of the trajectory. The goal is to find a set of parameters for the policy that maximises the expected return. To employ simple optimisation methods such as stochastic gradient descent, one needs to have an easy way to obtain the gradients of the policy w.r.t to the parameters. The policy gradient theorem (Williams, 1992) provides a useful identity for that

$$\nabla_\theta J(\theta) = E_{\pi_\theta}[r(\tau)\nabla_\theta log(\pi_\theta(\tau))]. \tag{2.1}$$

The REINFORCE algorithm (Williams, 1992) estimates the expected return $r(\tau)$ following policy π_θ by sampling episodes and computing Monte Carlo returns. The REINFORCE policy gradient estimate is known to suffer from high variance. One way to combat this, is by introducing a baseline. Choosing to estimate a value function via the TD error and using it as a baseline for the policy gradient estimate leads to actor-critic methods (Sutton and Barto, 1998). If the estimate of the value function as well as the policy are both parameterized via neural networks it can be useful to share parameters

(Mnih et al., 2016). This can simplify learning if the first layers of the neural network are likely to learn lower level features of the input which can be useful for both, the value function and the policy.

To stabilise learning with neural networks, it has been found crucial to break correlation in the data produced by an agent acting in an environment. For the off-policy DQN algorithm, this can be achieved by collecting experience in a replay buffer. A batch of non-sequential experience can then be sampled and used for updating the action value function. To achieve the same effect for on-policy actor critic methods A3C proposed the A3C algorithm. Here, multiple copies of an agent act in their own instance of the environment. The so called worker processes produce decorrelated experience and compute gradients, which are send regularly to the learner process that performs the update. In case all environments are synchronized and the update is based on the batch of experience produced by the parallel environments, one oobtains the A2C algorithm.

Other aspects that have been found crucial to stabilise training with A3C are entropy regularisation and estimating expected returns via N-step returns (Mnih et al., 2016). Entropy regularisation encourages the agent to maintain exploration by penalising a policy that becomes too certain early on in training. This often prevents policies from learning a suboptimal policy. The way Mnih et al. (2016) make use of N-step returns is as follows. The policy is run for N steps and visited states, action probabilities, chosen actions and received rewards are stored. Then one loops backward through the experience and estimates the return at each step by the reward received at the update timestep and the discounted return computed at the timestep before.

To train agents in their gridworld environments Tacchetti et al. (2019) use the IMPALA architecture (Espeholt et al., 2018). IMPALA aims at maximising the throughput of experience measured by the number of environment frames per second used for training. It modifies the actor-learner framework introduced above. Now the different worker processes in which copies of the same agent act in multiple instances of the environment refrain from calculating the gradient updates. They only communicate trajectories of experience to the learner. Now the learning algorithm is off-policy, but Espeholt et al. (2018) introduce a V-trace update which accounts for the difference in the policy that is updated and the policy that was used to produce the experience based on which the update is computed.

2.3 Agent modelling with DRL

The central challenge for agent modelling with DRL is how to integrate the information obtained by modelling the agent into one of the existing DRL methods described above. He and Boyd-Graber (2016) extended the DQN-algorithm by using a mixture of expert network structure for the Q-network whose action values outputs are weighted by a gating network that takes observation about the opponent as input. Another approach is to utilise the agents own policy to simulate the behaviour of the other agent and use this simulation to enhance its own decision making (Raileanu et al., 2018). RFMs are the attempt to utilise the relational inductive bias present in graph neural networks to provide an agent model (Tacchetti et al., 2019) that exploits the relational structure between agents and objects in the environment. The action predictions probabilities from the RFM are concatenated to the RGB image input of the policy network. Recently graph-based policy learning (Rahman et al., 2020) has been proposed, which leverages factorized Q-functions and a GNN based agent model to let an agent learn robust behaviour with respect to a wide range of teammate policies in an open environment. By weighting the action values with their likelihood of being realised, graph-based policy learning finds an intuitive way of integrating agent predictions with Q-learning, without needing the learning algorithm to rediscover the meaning of the predicted action probabilities.

In contrast to agent modelling approaches, MARL tackles MAS with a different premise, i.e. all agents can be controlled. The focus in agent modelling is to train a policy that is robust against different teammate policies (Rahman et al., 2020), while MARL algorithms focus on learning a set of policies for optimal coordination. In MARL a recurrent theme has been centralised training - decentralised execution. That is during the training a centralised critic is employed to learn agents local policies only depending on their observation (Lowe et al., 2017b; Foerster et al., 2018). MARL inherently uses more information (reward for all agents) to learn coordination than is available for in agent modelling scenarios.

2.4 Relational forward models

RFMs are recurrent graph neural networks designed at predicting teammate agents actions. The model is based on the notion of a graph neural network introduced by Battaglia et al. (2018). A graph is seen as a triplet $G = (V, E, u)$, where V and E are the usual set of vertices and edges respectively and u is a global property of the graph. A basic graph neural network layer takes as input a graph characterised by its node, edge and global features and outputs a graph of the same structure but with processed node, edge and global representations. The computation of the new representations is structured into three steps:

$$\begin{aligned}
&(1) \quad e_i = \phi_e(e_i, v_{s_i}, v_{r_i}, u), \quad \bar{e}_i = p_e(\bar{E}_i) \quad \bar{E}_i = \{e_k | r_k = i\} \\
&(2) \quad v_i = \phi_v(v_i, \bar{e}_i, u), \quad \bar{v} = p_v(V), \quad \bar{e} = p_u(E) \\
&(3) \quad u = \phi_u(\bar{v}, \bar{e}, u),
\end{aligned}$$

where s_i and r_i refer to the sender and receiver node of edge i respectively. A relational forward model consists of an encoder, a recurrent unit and a decoder. For the encoder and decoder unit the ϕ_e, ϕ_v and ϕ_u are all 64-unit-single-layer MLPs with ReLU activation and the p_*'s are all summations. A recurrent unit maintains a state graph for each timestep, whose node, edge and global features are the hidden states of three recurrent units, here a GRU (Cho et al., 2014). At each timestep the node, edge and global values of the output graph of the encoder unit are used as input into the corresponding recurrent unit to obtain the state graph for the next timestep. The computed state graph is then used as input into the decoder unit (Figure 2.1).

Given a gridworld, the environment state s is parsed into a graph where each landmark and each agent are represented by a node. The graph is used as input into an RFM and the computed node representations of the agents are used as input into a 64-unit-single-layer MLP with ReLU activation and softmax output layer to compute the action prediction probabilities for the agent's next actions. Since the graph neural network uses recurrent units, gradients from the cross-entropy loss between the action predictions and ground truth actions at timestep t are backpropagated through all previous encoder and recurrent graph units.

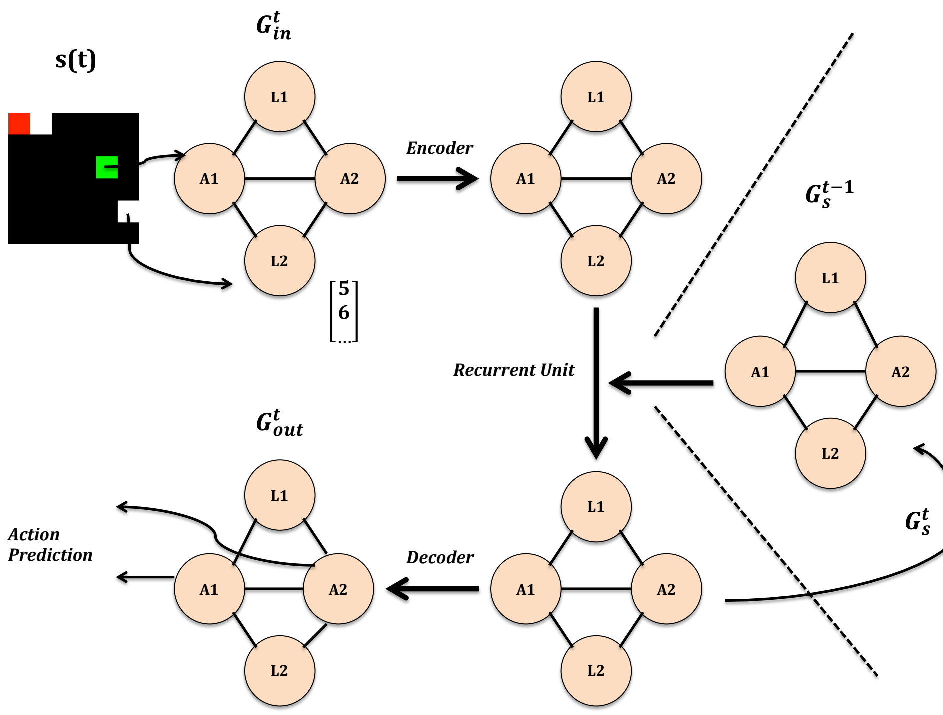


Figure 2.1: Relational forward model architecture. An environment state s is parsed into a graph G_{in}^{t}, by mapping different environment information into the node, edge and global values of the graph. The input graph is encoded before it is combined with the state graph from the recurrent graph unit. The output of the recurrent unit is used as the state graph for the next timestep. The output is then passed through a decoder, whose node, edge and global values can be used for further computations such as action prediction in agent modelling.

Chapter 3

Methodology

We start off by explaining the environments which are used for evaluation. The next section illustrates the policy used to control agents and the training set-up employed to train it. It further points out how our training set up differs slightly from the set up used by (Tacchetti et al., 2019) to train their agents. Details on the different agent models that were implemented are given. We further explain how the predictive accuracy of different agent models is evaluated on episodes produced by pretrained agents. Next, it is explained how an agent in combination with an agent model is trained. Finally, the method to measure the computational time used by the different algorithms is outlined.

3.1 Environments

We implement two gridworld environments used by Tacchetti et al. (2019) to evaluate their algorithms. We dispense of extending our evaluation to their third environment called Coin Game. Augmenting the input of an agents policy with predictions from an RFM did not lead to a significant increase in sample efficiency for the Coin Game. One of the main points of the thesis is to show the importance of having a computational analysis whenever new reinforcement learning algorithms are proposed. Only illustrating improved sample efficiency, can give a misleading image of the usefulness of an algorithm, especially when time is a limiting factor. If there is no sample efficiency in the first place, there is little to show. One important aspect of both gridworld environments is that they are fast to simulate. They are not the computational bottleneck of the learning algorithm. Details on the exact compute time needed for different parts of the learning algorithms are given in section 4.3.1.

Cooperative Navigation (Lowe et al., 2017a): Two agents and two tiles are randomly

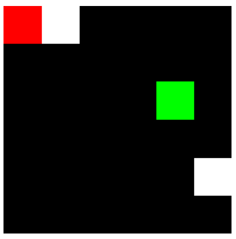


Figure 3.1: Rendered Cooperative Navigation environment. The red and green square are the agents which need to move to cover the white squares to get a reward of 1 per occupied timestep.

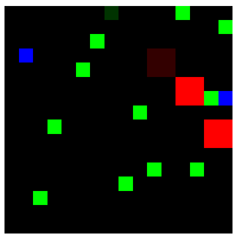


Figure 3.2: Rendered Stag Hunt environment. The red squares are stags, the green squares are apples and the blue squares the agents. Any collected items are displayed shaded.

placed in a 6×6 grid at the beginning of an episode. Each episodes lasts 30 timesteps and both agent receive a reward of +1 for each timestep they cover both of the tiles (Figure 3.1).

Stag Hunt (Lerer and Peysakhovich, 2017): Two agents move in a 16×16 grid filled with 12 apples and 3 stags. Each stag covers a square of 4 grid cells. Episodes end after 32 timesteps and for each collected apple an agent receives a reward of $+5$ at the moment of collection. A stag can only be collected if both agents are located on the stag at the same time and results in a reward of $+10$. Each collected item is respawned with probability 0.1. The position of the agents and landmarks are determined randomly at the beginning of each episode (Figure 3.2).

3.2 Pretraining of teammate agents

We trained agents to solve both environments via an independent learner approach (Tan, 1993) using the A2C algorithm (Mnih et al., 2016). From now on we refer to these agent as pretrained agents. They fullfill the purpose of having non-learning agents to interact with for the learning agent, when we explore how agent models influence the sample efficiency and computational time of the learning algorithm. Further, they are used to produce episodes as training data to asses the predictive capabilities of the different agent models presented later. In the independent learner approach each agent has its own policy network and no information between agent is shared such that other agents are seen as part of the environment. That means that the only information an agent has about other agents is by changes in the environment state. Agents are not explicitly modelled. For the validity of the experimental setup it is important that teammates learn to behave in a predictable manner on par with pretrained agents from Tacchetti et al. (2019). Otherwise the usefulness of an agent model in our experimental setup and the setup of Tacchetti et al. (2019) may differ. Take for example a teammate agent acting randomly, the additional information given by the agent model should not help in simplifying coordination with the random teammate. As an indicator of whether agents learn predictable behaviour we analyze final rewards of the teammate agents and visualize trained agents behaviour.

It is important to note that Tacchetti et al. (2019) used a slightly different training setup. They combined the idea of population based training with the IMPALA training infrastructure. In population based training each time a new episode starts, the agents that will act in this environment are sampled from the population of existing agents (Jaderberg et al., 2018). Since the clusters available to us make it difficult to set up this distributed training architecture, agents only compete against the same agent during training. As a consequence our pretrained agents may show qualitatively different behaviour, compared to the ones trained by Tacchetti et al. (2019). This may be the cause of any differences that emerge between our results and the results presented in Tacchetti et al. (2019).

3.2.1 Policy architecture and training

The architecture of the policy controlling the pretrained agents follows Tacchetti et al. (2019). The state of the gridworld is represented as an RGB image with values between $[0,1]$, which is used as input into a convolutional layer with a kernel of size 3×3 and six output channels. The output is flattened, concatenated with the reward and one-hot encoded action of the agent of the previous timestep and used as input into a 256-unit-single-layer MLP with ReLU activation. The output is further processed by an LSTM with a 256-dimensional hidden state. The action prediction and state value prediction are produced by two separate 64-unit-single-layer MLPs based on the current hidden state of the LSTM. The policy is updated via policy gradients with N-step returns (N=5). Each 60.000 environment steps the training procedure is halted and the current policies are evaluated over 240 episodes by recording average reward for each agent. If the achieved mean reward is a new maximum for the agent the policy is saved. The final saved policies are evaluated over 100 episodes.

3.3 Supervised training of agents models

We look at the prediction accuracy of four different agent models. One of them is the RFM model described earlier. We refrain from repeating the description and explain how each environment is parsed into the input graph G_{in}^{t} at timestep t. Further we look at three variations of CAF models, which differ with respect to the information they are conditioned on.

RFM: For both environments each agent and landmark are mapped to a node of the input graph. All agent nodes are connected with each other and with all landmarks. For both environments the node vectors contain the corresponding coordinates, the one-hot encoded entity type, the one-hot encoded last action and whether the entity is collected or not. If an information for a node is not available, i.e. last action of an apple in the Stag Hunt environment, the information is zero-padded. The input graph does not have edge values or a global value. For the corresponding functions ϕ_{*}, the input is reduced to the available information, for example (e, v_s, v_r, u) becomes (v_s, v_r).

CAF-Basic: For the Stag Hunt environment we condition on the distance on the x- and y-axis from the agent to the closest apple on the grid as well as the agents last action. For the Cooperative Navigation environment we condition on the distance on the x- and y-axis from the agent to the closest tile and the last action of the agent.

CAF-OA: The same as CAF-Basic, but for each environment we additionally condition on the position of the other agent. In the following this model is referred to as ”CAF-OA (other agent)”.

CAF-AH: We tune the number of previous actions to condition on using a validation set of episodes. For the Stag Hunt and Cooperative Navigation environment this leads to a model that takes into account the last 5 and 6 actions respectively. This model will be denoted by ”CAF-AH (action history)”.

To asses the predictive capabilities of the different agent models, pretrained agents are used to generate 500.000 episodes as a training set and 2500 further episodes as a test set (Tacchetti et al., 2019). Two metrics are used for the evaluation of the agent models. The mean length of perfect rollout i.e. the number of timesteps from the beginning of an episodes for which all actions are predicted correctly and the fraction of overall correct predicted actions.

3.4 Training of agents with an agent model

Following Tacchetti et al. (2019), experiments are conducted to show that equipping an agent with an agent model makes learning more sample efficient. That is an agent is paired with a pretrained agent to learn to solve the environment. An agent is either equipped with an untrained RFM model and or no agent model at all. Compared to Tacchetti et al. (2019) two more variants of the learning algorithms are added. For the ”true model” variant, the agents receive the action prediction from the perfect agent model, that is the pretrained agent makes its decision first and the output of its policy is given to the learning agent. For the ”caf model” variant the third variant of the CAF models (CAF-AH) is used to equip the learning agent with an agent model.

Predictions from an agent model are incorporated into the actor-critic method in the same way for all models. The initial three-channel RGB image of the gridworld becomes a four channel image. The additional channel contains the action predictions probabilities as pixel values at the grid positions the modelled agent could land in the next timestep. For the RFM the gradient from the policy loss is not backpropagated to the RFM (Tacchetti et al., 2019), such that learning the policy and learning the action prediction via the agent model does not interfere with each other. The training procedure is the same as it was for the pretrained agents. Each training algorithm is run for four seeds for the Cooperative Navigation environment and for two seeds for the larger and therefore also more computationally demanding Stag Hunt environment. Additionally to evaluating the agents performance every 60.000 environment steps, the accuracy per episode of the agent model’s predictions is measured.

3.4.1 Measuring Computation Time

Algorithm 1 Abstract N-step A2C algorithm with agent model

```
Let S denote the state, A the actions, T the total number of training steps
Let n denote the current step
As,Ss,Rs = [],[],[]
done = True
for $t \leftarrow 1$ to $T$ do
    if done then
        S = Env.reset()
        done = False
    end
    n = 0
    T0 = current time
    while $n \leq$ N-step and not done do
        n = n + 1
        T1 = current time
        Pred = AgentModel(S)
        T2 = current time
        A = Policy(S,Pred)
        As.append(A)
        T3 = current time
        AgentModel.Update(Pred,A)
        T4 = current time
        S, R1, done = Env(S,A)
        T5 = current time
    end
    T6 = current time
    AgentModel.Update(As,Ss,Rs)
    T7 = current time
end
```

To test the hypothesis that variants of the learning algorithm with simpler agent models or no agent model achieve similar reward in less computation time than the RFM-agent, we measure the time each variant of the learning algorithm takes for its different subparts. The computations performed during the training loop are separated into five different operations, illustrated in abstract pseudocode in Algorithm 1. Recording the number of N-step updates that are performed during training, we can recover the total computation time of the algorithm using the average time for an N-step update, measured by T7-T0.

For a fair test of the hypothesis it is important that each agent model is implemented as efficiently as possible. Querying as well as updating the conditional action frequency model can be implemented as a look-up in a dictionary such that both computations are O(1). The computational cost of both operations is also independent of the size of the conditioning set, however memory costs can become high if the number of action frequency distributions to store becomes too large. The RFM model was implemented using the Deep Graph Library (Wang et al., 2019) and made use of its automating batching functionality, parallelizing as many operations as possible. Since DeepMind has not released their graph neural network framework, we were unable to replicate the RFM implementation 1-to-1, but the Deep Graph Library outperforms other open source graph neural network frameworks such as PyTorch geomtric (Fey and Lenssen, 2019) in terms of speed by leveraging fused message passing.

Chapter 4

Results

First we provide statistics for the training of teammate agent policies, which serve mainly as a sanity checks that agents show the desired behaviour. Then we discuss the action prediction accuracies achieved by the different agent models from section 3.3. We provide the results regarding the computation time of the different learning algorithms.

4.1 Pretraining of teammate agents

4.1.1 Analysis of trained teammates

Environment	Average Reward (100 episodes - 5 runs)
Cooperative Navigation	24.13 ± 0.2826
Stag Hunt	23.73 ± 0.39

Table 4.1: Transient rewards of pretrained agents.

Unfortunately (Tacchetti et al., 2019) do not provide any statistics of the behaviour of their pretrained agents. Judging from the performance of agents with an agent model illustrated in (Figure 4.3) similar limit performance is achieved by our pre-trained agents (Figure 4.1). Visualizing the trained policies shows that agents in the Cooperate Navigation environment learn to move directly to the uncovered tiles and remain there for the rest of the episode. Agents in the Stag Hunt environment compete for the closest apples, such that in both environments agents show sensible behaviour. However, the pretrained agents do not show any cooperation. That is agents in the Stag Hunt environment compete for the closest apple instead of aiming to cover a stag with

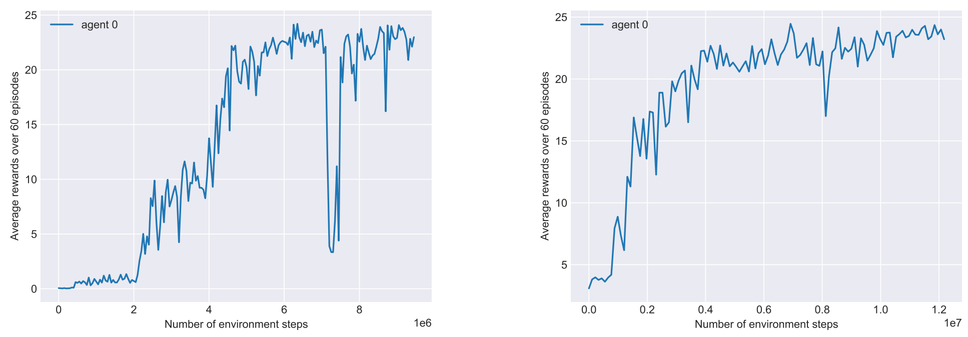


Figure 4.1: Average rewards over training for one of the two agents in the Cooperative Navigation environment (left) Stag Hunt environment (right).

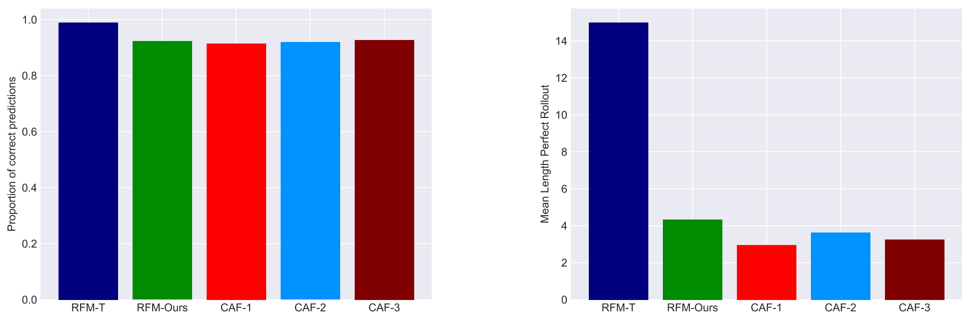


Figure 4.2: Evaluation of the agent models on episodes from the Cooperative Navigation via prediction accuracy (left) mean length of perfect rollout (right).

the teammate. Similarly agents in the Cooperative Navigation environment often move to the same tile first until one of the agent has covered it.

4.2 Action prediction performance of agent models

In both environments RFM outperforms all action frequency models in terms of mean length of perfect rollout. The high prediction accuracy in the Coopertation Navigation environment stems from the fact that pretrained agents learn to move to a tile and stay there for the rest of the episode, such that simply predicting the action stay gives an accuracy of 0.88. In both environments the conditional action frequency model shows the same pattern, i.e. conditioning on more previous timesteps does not improve the mean length of perfect rollout but the prediction accuracy, while conditioning on the position of the other agent improves the mean length of perfect rollout but not the

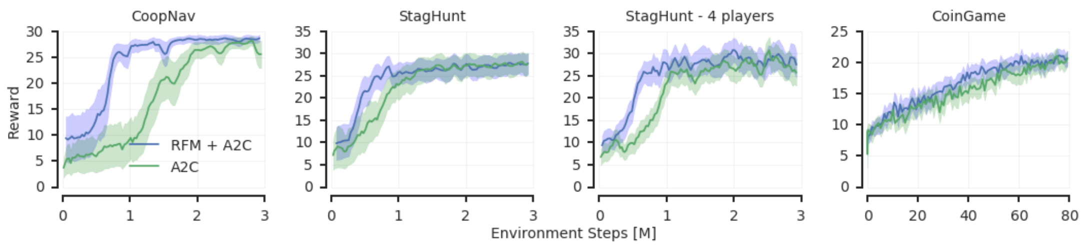


Figure 4.3: Peformance of agents augmented by an RFM and without an agent model on the Cooperative Navigation environment and the Stag Hunt environment. Agents that were augmented with an RFM needed less samples to learn. Retrieved from (Tacchetti et al., 2019)

action prediction accuracy. The metric mean length of perfect rollout is biased toward agent models that perform well at the beginning of an episode. An agent model that would predict everything perfectly but the first timestep would have a mean length of perfect rollout of 0. This suggests that information about the position of the other agent at the beginning of an episode makes predicting actions easier. Knowing where the other agents is located may help the agent to decide to which tile to move to or which apple to chase.

Comparing the predictive accuracy with results from Tacchetti et al. (2019), our RFM implementation falls short by a large margin for both environment (figure 4.2, 4.4). One potential reason could be that the episodes produced by the pretrained agents contain less information useful for prediction than in Tacchetti et al. (2019). For example, any powerful agent model would perform poorly if faced with episodes from a random agent. To investigate this claim we implement another agent model investigated by (Tacchetti et al., 2019) and train in on the same data. We implement the MLP + LSTM baseline. The vectors forming the node representation of the input graph are concatenated to form the input. The encoder is a single layer 64-unit MLP whose output is used as input to a LSTM with a hidden unit of size 32. The decoder is formed by a two layer MLP with hidden size of 32. In Tacchetti et al. (2019) this baseline performs equally well on the Cooperative Navigation environment. For us it achieved a prediction accuracy of 0.944 ± 0.0004. This indicates that independently of which baseline we implement, the algorithm has worse predictive performance compared to the reported performance metrics in Tacchetti et al. (2019). Since the final accuracy of agent models was quite insensitive to different hyperparameter settings during training, it is unlikely that the observed difference in action prediction accuracy is caused

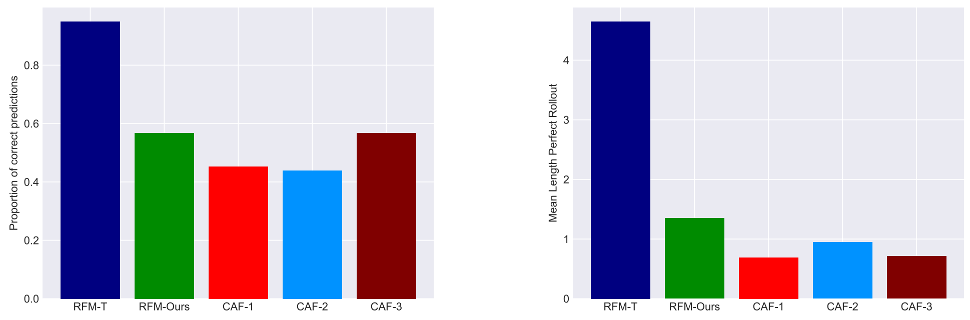


Figure 4.4: Evaluation of the agent models on episodes from the Stag Hunt via mean length of perfect rollout (right) and prediction accuracy (left).

by a suboptimal choice of hyperparameters. Having an agent model that predicts the next action of teammates less well could imply that any observed improvement in the sample efficiency could be further increased. Therefore any results regarding a computational advantage of learning algorithms without an agent model would face the argument that the agent model analysed is not powerful enough. We want to argue that this is not true. First of all it is not clear whether action prediction accuracy translates into sample efficiency. Secondly the true model should provide an upper bound on how an optimal agent model can perform. By combining the sample performance of the learning algorithm augmented by the true model with the computational cost of the learning algorithm augmented with a RFM, one could estimate an upper bound for the computational performance of augmenting the learning algorithm with a RFM.

4.3 Analysis of computation time and sample efficiency

4.3.1 Computation time

From Tables 4.2 and 4.3, it becomes clear that the computational overhead an agent model provides depends on the time it takes to compute and update the policy and the time it takes to simulate the environment. Further, the additional computational cost of a CAF agent model is negligible up to a hundredth of a second. For the smaller Cooperation Navigation environment an N-step update takes almost three times as long with an RFM agent model compared to not having an agent model. This means the learning algorithm augmented by a RFM must learn to achieve similar reward with a third of the number of N-step updates that can be used by learning algorithm with no computational model to be computationally competitive. For the larger Stag Hunt environment the same analysis holds, with the difference that the learning algorithm augmented by a RFM must learn with half the number of N-step updates to be computational competitive. Also of interest is the absolute time difference between one N-step update to understand the total computation time that can be saved with each algorithm. However to provide actual time estimates we would need estimate of the number of N-step updates the different learning algorithms need for training. For neural net-based agent models especially the updating introduces computational overhead, due to the backpropagation operation.

	No model	RFM	CAF3
Action Prediction (T2-T1)	/	**0.011**	**0.0006**
Action Inference (T3-T2)	0.012	0.013	0.0132
Environment Simulation (T4-T3)	0.0014	0.0015	0.0013
Updating Agent Model (T5-T4)	/	**0.032**	**0.0008**
Updating Policy (T7-T6)	0.029	0.039	0.029
N-Step Loop (T7-T0)	0.110 ± 0.0008	0.328 ± 0.014	0.110 ± 0.0015

Table 4.2: Estimates (5 runs - 1000 iterations) of the computation time in seconds of the different training algorithms divided up into individual components for the Cooperative Navigation. The meaning of the different timing intervals can be taken from Algorithm 1.

	No model	RFM	CAF3
Action Prediction (T2-T1)	/	**0.014**	**0.0007**
Action Inference (T3-T2)	0.054	0.056	0.0588
Updating Agent Model (T4-T3)	/	**0.050**	**0.0008**
Environment Simulation (T5-T4)	0.0023	0.0022	0.00224
Updating Policy (T7-T6)	0.055	0.066	0.060
N-Step Loop (T7-T0)	0.322 ± 0.011	0.627 ± 0.013	0.35 ± 0.013

Table 4.3: Estimates (5 runs - 1000 iterations) of the computation time in seconds of the different training algorithms divided up into individual components for the Stag Hunt environment. The meaning of the different timing intervals can be taken from Algorithm 1.

4.3.2 Cooperative Navigation

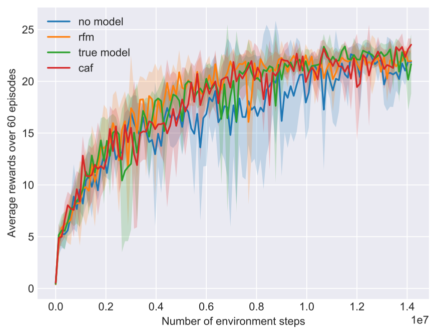
The sample efficiency achieved by the agent training with a RFM from Tacchetti et al. (2019) on the Cooperative Navigation environment (Figure **?**) can not be observed (Figure 4.5 (left)). None of the variations of the learning algorithm differ noticeably in the number of samples they need to achieve a fixed reward (Figure 4.5 (left)). Having no model performs slightly worse in terms of sample efficiency. These variations however are not significant. This cannot be explained by arguing that action predictions of the agent models have poor quality. From Figure 4.7 one can see that both agent models achieve an accuracy rate of roughly 0.8 shortly after training started.

What could be possible reasons for the lack of sample efficiency of the RFM agent? The first reason that comes to mind is that the RFM of Tacchetti et al. (2019) had a higher prediction accuracy and a higher mean length of perfect rollout. However, if that would be the problem then the true model should outperform all other baseline algorithms in terms of sample efficiency but it fails to do so.

This suggests that the information given by an agent model is not used by the policy network for decision making. There are two potential reasons why the policy learns to ignore the action prediction input.

First, the way action predictions are integrated into the learning algorithm, makes it difficult for the policy to make use of the additional information. Since probabilities are turned into a form of pixel intensity the policy needs to relearn the meaning of the values as probabilities of next actions. Another reason could be that agents are able to learn a good policy without taking the other agent into account. Since Tacchetti et al. (2019) used the same technique to augment environment observations with action predictions and achieved higher sample efficiency, the latter reason seems to be more likely. Visualising the behaviour of agents, the policy learned by the agents seems independent of the other agent. Both agents move to the tile closest to them. If the tile is already occupied, since the other agent arrived first, the other tile is targeted.

As a consequence of the lack in sample efficiency of the agent with RFM, it is imminent that the agent with RFM needs more computational time to achieve a fixed reward. The time difference between the first time a reward of over 20 is achieved for the CAF augmented agent and the RFM augmented agent is about six hours in favour of the former (figure 4.6).



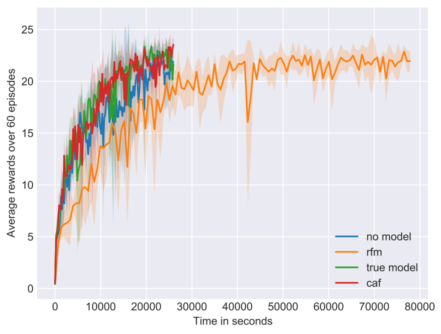


Figure 4.5: Reward per environment step (left) and per computation time in second (right) achieved by the learning agent on the Cooperative Navigation environment.

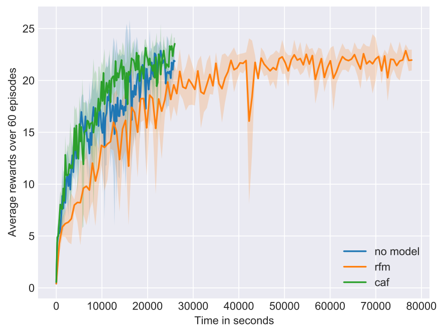


Figure 4.6: Reward after computation time in seconds for both types of agent models and the no model variation.

4.3.3 Stag Hunt

For the Stag Hunt environment the improvement in sample efficiency by adding an RFM can be replicated (figure 4.8). The agent augmented by an RFM takes on average roughly three million environment steps less to reach a reward of 20. This relation reverses if we look at the reward after computation time in seconds. It takes up to 10 hours less for the agent without an agent model to reach an average reward of 20. This suggests that having no agent model at all gives a policy of the same quality in less time. This supports the first part of our initial hypothesis that despite the fact that the learning algorithm with RFM is more sample efficient, training a policy without an agent model leads to similar final reward in less time.

Unfortunately the variability of the training results for the ”true model” variant and the ”CAF” model variant were to high to give interpret able results. Therefore the second part of the hypothesis remains open until more experiments are run with CAF

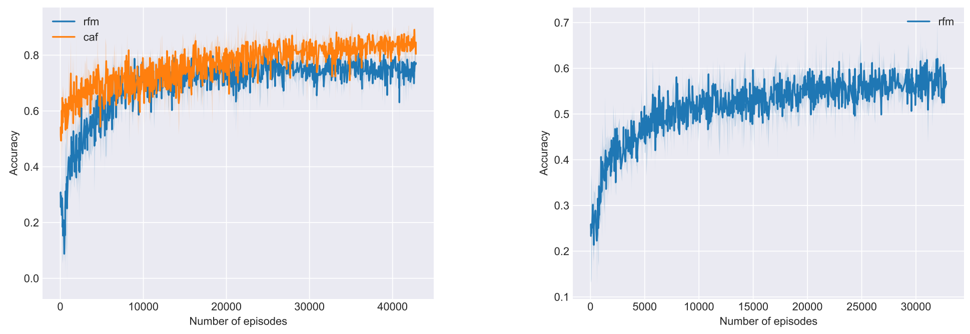


Figure 4.7: Action prediction accuracy of the RFM and CAF model during training on the Cooperative Navigation environment (left) and for the RFM model on the Stag Hunt environment (right).

models as the agent model.

Similar to the Cooperative Navigation environment the RFM reaches a predictive accuracy of the level expected from the supervised learning experiments early on in training (figure 4.7).

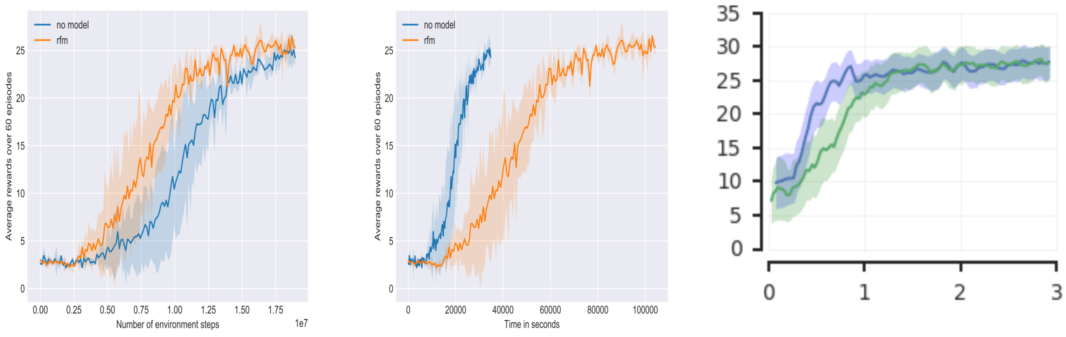


Figure 4.8: Reward per environment step (left) and per computation time in seconds (middle) achieved by the learning agent with and without RFM on the Stag Hunt environment. On the right, the reward achieved during training on the Stag Hunt environment by agents from Tacchetti et al. (2019). The image is retrieved from Tacchetti et al. (2019).

Chapter 5

Conclusions

This thesis investigates whether simpler agent models such as conditional action frequency models can outperform relational forward models Tacchetti et al. (2019) in terms of reward achieved after computation time in seconds. Relational forward models have previously be shown to outperform other agent models in terms of action prediction accuracy and to improve the sample efficiency of actor critic methods.

The thesis aims at providing evidence that for any newly proposed reinforcement learning algorithm next to its sample efficiency and final performance, its computational properties should be analysed and compared to existing methods. This claim was investigated by analyzing the sample and computational efficiency of a learning algorithm for for a single learner in a MAS (Tacchetti et al., 2019). The problem of improving single agent learner algorithms for MAS via agent modelling provides an interesting starting point for the analysis. Agent models with different computational complexity and predicitve accuracy can variably be chosen to augment the learning algorithm or not. Specifically, we looked at whether the improved sample efficiency of augmenting an actor-critic algorithm by action predictions from the RFM (Tacchetti et al., 2019) also translates into a computational advantage. We formulated two hypotheses. First, despite improving the sample effciency of the learning algorithm, the learning algorithm augmented by the RFM needs more time to achieve similar reward to a policy without agent model. We found support for this claim on a gridworld environment. Training with an RFM took up to 10 hours more computation time than without for similar final reward.

We followed the methodology of Tacchetti et al. (2019) to the best of our knowledge, but it can always be the case that some part of it was misunderstood which would explain the failure to replicate the action prediction results. This also shows the importance of making ones code publicly available, to mitigate this risk of not being able to replicate results. Another potential reason could be that our pretrained agent showed different, less predictable behaviour.

It is important to notice the conditions under which the insights from this thesis hold. That is the agent model must be the computational bottleneck of the method. If the environment simulation or the inference and update operations of the policy are computationally dominant, than sample efficiency matters more. Here it could be interesting to see how well conditional action frequency models perform as they need less computational time compared to RFMs but also may improve sample efficiency compared to having no agent model. Further, one must also take into consideration that there may exist a more efficient implementation of the relational forward model.

The next step is to understand why our action prediction accuracy results differ from the results presented in Tacchetti et al. (2019). Then the results presented in section 4.3.3 must be replicated with the differences between our method and the method presented in Tacchetti et al. (2019) removed. CAF models may provide an elegant trade-off between sample efficiency and computational complexity, but experimental evidence on their performance is inconclusive or missing.

As previously mentioned, other methods based on Q-learning (Rahman et al., 2020) have a naturally way of integrating action predictions by weighting the different action-values by their likelihood of being realised. An equivalent method for policy gradient based methods does not seem to exist and could be interesting and useful to explore.

Bibliography

S. V. Albrecht and P. Stone. Autonomous agents modelling other agents: A comprehensive survey and open problems. *Artif. Intell.*, 2018.

P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. F. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, Ç. Gülçehre, H. F. Song, A. J. Ballard, J. Gilmer, G. E. Dahl, A. Vaswani, K. R. Allen, C. Nash, V. Langston, C. Dyer, N. Heess, D. Wierstra, P. Kohli, M. Botvinick, O. Vinyals, Y. Li, and R. Pascanu. Relational inductive biases, deep learning, and graph networks. *CoRR*, 2018.

D. Chakraborty and P. Stone. Multiagent learning in the presence of memory-bounded agents. *Auton. Agents Multi Agent Syst.*, 2014.

S. C. Y. Chan, S. Fishman, A. Korattikara, J. Canny, and S. Guadarrama. Measuring the reliability of reinforcement learning algorithms. 2020.

K. Cho, B. van Merrienboer, Ç. Gülçehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP*, 2014.

L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning, S. Legg, and K. Kavukcuoglu. IMPALA: scalable distributed deep-rl with importance weighted actor-learner architectures. In *Proceedings of the 35th International Conference on Machine Learning, ICML*, 2018.

M. Fey and J. E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

J. N. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson. Counterfactual multi-agent policy gradients. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence,*, 2018.

H. He and J. L. Boyd-Graber. Opponent modeling in deep reinforcement learning. In *Proceedings of the 33nd International Conference on Machine Learning, ICML*, 2016.

M. Jaderberg, W. M. Czarnecki, I. Dunning, L. Marris, G. Lever, A. G. Castañeda, C. Beattie, N. C. Rabinowitz, A. S. Morcos, A. Ruderman, N. Sonnerat, T. Green, L. Deason, J. Z. Leibo, D. Silver, D. Hassabis, K. Kavukcuoglu, and T. Graepel. Human-level performance in first-person multiplayer games with population-based deep reinforcement learning. *CoRR*, 2018.

A. Lerer and A. Peysakhovich. Maintaining cooperation in complex social dilemmas using deep reinforcement learning. *CoRR*, 2017.

T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. In *4th International Conference on Learning Representations, ICLR*, 2016.

R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in Neural Information Processing Systems 30, NIPS*, 2017a.

R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems,*, 2017b.

V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, 2013.

V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *Proceedings of the 33nd International Conference on Machine Learning, ICML*, 2016.

D. T. Nguyen, A. Kumar, and H. C. Lau. Credit assignment for collective multiagent RL with global rewards. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems, NeurIPS*, 2018.

G. Papoudakis, F. Christianos, A. Rahman, and S. V. Albrecht. Dealing with non-stationarity in multi-agent deep reinforcement learning. *CoRR*, 2019.

A. Rahman, N. Hopner, F. Christianos, and S. V. Albrecht. Open ad hoc teamwork using graph-based policy learning. *CoRR*, 2020.

R. Raileanu, E. Denton, A. Szlam, and R. Fergus. Modeling others using oneself in multi-agent reinforcement learning. In *Proceedings of the 35th International Conference on Machine Learning, ICML*, 2018.

T. Rashid, M. Samvelyan, C. S. de Witt, G. Farquhar, J. N. Foerster, and S. Whiteson. QMIX: monotonic value function factorisation for deep multi-agent reinforcement learning. In *Proceedings of the 35th International Conference on Machine Learning, ICML*, 2018.

D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

R. S. Sutton and A. G. Barto. *Reinforcement learning - an introduction*. 1998.

A. Tacchetti, H. F. Song, P. A. M. Mediano, V. F. Zambaldi, J. Kramár, N. C. Rabinowitz, T. Graepel, M. Botvinick, and P. W. Battaglia. Relational forward models for multi-agent learning. In *7th International Conference on Learning Representations, ICLR*, 2019.

M. Tan. Multi-agent reinforcement learning: Independent versus cooperative agents. In P. E. Utgoff, editor, *Machine Learning, Proceedings of the Tenth International Conference*, 1993.

E. Van der Pol and F. A. Oliehoek. Coordinated deep reinforcement learners for traffic light control. *Proceedings of Learning, Inference and Control of Multi-Agent Systems (at NIPS 2016)*, 2016.

M. Wang, L. Yu, D. Zheng, Q. Gan, Y. Gai, Z. Ye, M. Li, J. Zhou, Q. Huang, C. Ma, Z. Huang, Q. Guo, H. Zhang, H. Lin, J. Zhao, J. Li, A. J. Smola, and Z. Zhang. Deep graph library: Towards efficient and scalable deep learning on graphs. *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019. URL `https://arxiv.org/abs/1909.01315`.

C. J. C. H. Watkins and P. Dayan. Technical note q-learning. *Mach. Learn.*, 1992.

R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.*, 1992.