Proceedings of the Fifteenth AAAI Conference on Artificial
Intelligence and Interactive Digital Entertainment (AIIDE-19)

Agent Modeling as Auxiliary Task for Deep Reinforcement Learning

Pablo Hernandez-Leal,* Bilal Kartal,* Matthew E. Taylor
Borealis AI
Edmonoton, Canada
{pablo.hernandez, bilal.kartal, matthew.taylor}@borealisai.com

Abstract

In this paper we explore how actor-critic methods in deep reinforcement learning, in particular Asynchronous Advantage Actor-Critic (A3C), can be extended with agent modeling. Inspired by recent works on representation learning and multiagent deep reinforcement learning, we propose two architectures to perform agent modeling: the first one based on parameter sharing, and the second one based on agent policy features. Both architectures aim to learn other agents' policies as auxiliary tasks, besides the standard actor (policy) and critic (values). We performed experiments in both cooperative and competitive domains. The former is a problem of coordinated multiagent object transportation and the latter is a two-player mini version of the Pommerman game. Our results show that the proposed architectures stabilize learning and outperform the standard A3C architecture when learning a best response in terms of expected rewards.

Introduction

An important ability for agents to have is to reason about the behaviors of other agents by constructing models that make predictions about the modeled agents (Albrecht and Stone 2018). This *agent modeling* (Schadd, Bakkes, and Spronck 2007)[1] area usually takes concepts and algorithms from multiagent systems (since the environment includes at least two agents), game theory (which studies the strategic interactions among agents), and reinforcement learning (since the model may be based on information observed from interactions).

Agent modeling usually serves two purposes in multiagent settings: it improves the coordination efficiency in cooperative scenarios (Chalkiadakis and Boutilier 2003) and, in competitive scenarios, it helps the agent to better optimize (best respond) its actions against the predicted opponent policy (Carmel and Markovitch 1995), e.g., by exploiting opponent mistakes.

Early algorithms for agent modeling came from game theory literature, e.g., fictitious play (Brown 1951). Later, many works adapted reinforcement learning algorithms for this task (Banerjee and Peng 2005). Recently, agent modeling has been also considered in the context of deep reinforcement learning (DRL).

DRL has shown outstanding results in Atari games, Go, Poker and recently in strategy video games (Mnih et al. 2013; Torrado et al. 2018). Due to these successes, it is natural that DRL is now being tested in multiagent environments (Hernandez-Leal, Kartal, and Taylor 2018). Some works have explored using DRL to evaluate emergent behaviors in multiagent environments (Tampuu et al. 2017), and others have proposed algorithms for multiagent DRL (Foerster et al. 2017). In contrast, our goal is to estimate other agents' (opponent or teammate) policies by means of an *auxiliary task* at the same time that the agent is learning its respective (best response) policy. In general, (self-supervised) auxiliary tasks are not used for anything other than shaping the features of the agent, i.e., facilitating the representation learning process (Bellemare et al. 2019), improving learning stability (Jaderberg et al. 2017), and have broadened the horizons of RL to learn from all experience, whether rewarded or not. Self-supervision defines losses via surrogate annotations that are synthesized from unlabeled inputs. Examples are reward prediction which can be cast into a regression task (Jaderberg et al. 2017) and dynamics prediction that captures state, action, and successor relationships. Since the purpose is representation learning and not full modeling of the dynamics and reward, the losses need not form a transition model and proxies can suffice to help tune the representation, i.e., these losses are expected to give gradients not necessarily a generative model (Shelhamer et al. 2017).

In this work we take advantage of auxiliary tasks when learning a best response and the opponent/teammate model. Since these two elements are linked to each other, we propose two architectures that take advantage of this realization.

Recently, *asynchronous* actor-critic methods have become widely used in DRL; Asynchronous Advantage Actor-Critic (A3C) (Mnih et al. 2016) is a major representative of this category, which does not use an experience buffer and learns completely on-policy. Thus, we take A3C as baseline and set off to evaluate and better understand the use of agent modeling as auxiliary task with on-policy actor-critic methods in

*Equal contribution

[1]Sometimes referred as *opponent modelling* since "opponent" is used to refer to another agent in the environment.

DRL with the following contributions:

- Agent modeling in DRL is still an open research area with opportunities in video games (Zhao and Szafron 2009; Torrado et al. 2018; Borovikov et al. 2019). Our experiments are performed in two recent multiagent environments, one cooperative and one competitive domain (mini version of the Pommerman game).
- We propose two new architectures that take inspiration from multiagent DRL and representation learning to do agent modeling. The first architecture, Agent Modeling by parameter Sharing (AMS-A3C), takes inspiration from the concept of *parameter sharing* to learn the opponent/teammate policy as an *auxiliary task* as well as the standard actor and critic.
- The second architecture, Agent Modeling by policy Features (AMF-A3C), leverages the concept of *policy features* to learn latent space features that are used as input when computing the actor and critic of the learning agent.

Our results show that modeling the opponent/teammate increases the expected rewards and improves the stability of the learning process. In particular, in this work we show the benefits of using opponent/teammate policy prediction as an auxiliary task with respect to non-learning stochastic agents in both cooperative and competitive scenarios.

Related Work

In this section, we describe related work on agent modeling in DRL, multiagent DRL and auxiliary tasks.

Deep Reinforcement Opponent Network (DRON) (He et al. 2016) was the first DRL work that performed opponent modeling. DRON's idea is to have two networks: one learns Q values (similar to DQN (Mnih et al. 2013)) and a second learns a representation of the opponent's policy. DRON used hand-crafted features to define the opponent network. In contrast, Deep Policy Inference Q-Network (DPIQN) and Deep Policy Inference Recurrent Q-Network (DPIRQN) (Hong et al. 2018) learned opponent *policy features* directly from raw observations of the other agents. The way to learn these policy features is by means of auxiliary tasks (Jaderberg et al. 2017) that provide additional learning goals; in this case, the auxiliary task is to learn the opponent's policy. Then, the Q value function of the learning agent is conditioned on the policy features, which aim to reduce the non-stationarity of the multiagent environment. In contrast, our proposals do not need an experience replay buffer, learn completely on-policy and we make use of full parameter sharing (Foerster et al. 2017).

Deep Cognitive Hierarchies (Lanctot et al. 2017) is an algorithm that aims to avoid overfitting in two-player games. It uses deep reinforcement learning to compute best responses to a distribution over policies and empirical game-theoretic analysis to compute new meta-strategy distributions. Theory of Mind Network (Rabinowitz et al. 2018) tackles the problem of meta-learning, i.e., the proposed network should acquire a strong prior model for agents' behaviour to bootstrap to richer predictions. DeepBPR+ studies the problem of efficient policy detection and reuse when playing against non-stationary agents in Markov games (Zheng et al. 2018). In contrast, our goal is to estimate the opponent/teammate's policy at the same time that the agent is learning its respective (best response) policy; since these two elements are linked to each other our proposals improve the stability of the learning process as well as increase the obtained rewards.

Self Other Modeling (SOM) (Raileanu et al. 2018) is a recently proposed algorithm that uses the agent's own policy as a means to predict the opponent's goal (and actions). SOM is based on the assumption that the agents are identical, which is more suitable when agents share a fixed set of goals and have similar abilities.

Auxiliary tasks were originally presented as *hints* that improved the network performance and learning time. Suddarth and Kergosien (1990) presented a minimal example of a small neural network where it was shown that adding an auxiliary task effectively removed local minima. Recently, some works have used them in single-agent RL problems, for example, Mirowski et al. (2017) studied self-supervised tasks (like depth prediction) in a navigation problem. Their results show that augmenting an RL agent with auxiliary tasks supports representation learning, which provides richer training signals that enhance data efficiency. Another interesting result is that using the auxiliary task as a loss was better than using the value as input. Another example was presented by Lample and Chaplot (2017) who added an auxiliary task (to predict game feature information such as the presence of enemies or items) to a DQN network to improve learning in First-Person-Shooting games. These ideas also relate to Multi-Task Learning where by learning tasks in parallel using a shared representation, what is learned for each task can help the learning of the others (Caruana 1997).

Preliminaries

We start with the standard reinforcement learning setting of an agent interacting in an environment over a discrete number of steps. At time t the agent in state s_t takes an action a_t and receives a reward r_t. The state-value function is the expected return (sum of discounted rewards) from state s following a policy $\pi(a|s)$:

$$V^{\pi}(s) = \mathbb{E}[R_{t:\infty}|s_t = s, \pi],$$

and the action-value function is the expected return following policy π after taking action a from state s:

$$Q^{\pi}(s, a) = \mathbb{E}[R_{t:\infty}|s_t = s, a_t = a, \pi].$$

Algorithms, such as Q-learning, or its (deep) neural network variant, DQN, approximate the action-value function $Q(s, a; \theta)$ using parameters θ, and then update parameters to minimize the mean-squared error, using the loss function:

$$L_Q(\theta_i) = \mathbb{E}[(r + \gamma max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i))^2]$$

where θ^- represents the parameters of the target network that is held constant, but synchronized to the behaviour network $\theta^- = \theta$, at certain periods to stabilize learning.

A3C (Asynchronous Advantage Actor-Critic) is an algorithm that employs a *parallelized* asynchronous training scheme (e.g., using multiple CPUs) for efficiency; it

is an on-policy RL method that does not need an experience replay buffer. A3C allows multiple workers to simultaneously interact with the environment and compute gradients locally. All the workers pass their computed local gradients to a global network that performs the optimization and synchronizes the updated actor-critic NN parameters with the workers asynchronously. A3C maintains a parameterized policy (actor) $\pi(a|s;\theta)$ and value function (critic) $V(s;\theta_v)$, which are updated as follows: $\triangle\theta = \nabla_\theta \log\pi(a_t|s_t;\theta)A(s_t,a_t;\theta_v)$ and $\triangle\theta_v = A(s_t,a_t;\theta_v)\nabla_{\theta_v}V(s_t)$ where

$$A(s_t,a_t;\theta_v) = \sum_{k=0}^{n-1}\gamma^k r_{t+k} + \gamma^n V(s_{t+n}) - V(s_t),$$

with $A(s,a) = Q(s,a) - V(s)$ representing the *advantage* function, commonly used to reduce variance.

The policy and the value function are updated after every t_{max} actions or when a terminal state is reached. It is common to use one softmax output for the policy head $\pi(a_t|s_t;\theta)$ and one linear output for the value function head $V(s_t;\theta_v)$, with all non-output layers shared, see Figure 1 (a).

The loss function for A3C is composed of two terms: policy loss (actor), $\mathcal{L}_\pi$, and value loss (critic), $\mathcal{L}_v$. An entropy loss for the policy, $H(\pi)$, is also commonly added to help to improve exploration by discouraging premature convergence to suboptimal deterministic policies (Mnih et al. 2016). Thus, the loss function is given by:

$$\mathcal{L}_{\text{A3C}} = \lambda_v\mathcal{L}_v + \lambda_\pi\mathcal{L}_\pi - \lambda_H\mathbb{E}_{s\sim\pi}[H(\pi(s,\cdot,\theta)]$$

with $\lambda_v = 0.5$, $\lambda_\pi = 1.0$, and $\lambda_H = 0.01$, being standard weighting terms on the individual loss components.

The UNsupervised REinforcement and Auxiliary Learning (UNREAL) framework (Jaderberg et al. 2017) is built on top of A3C. UNREAL proposes unsupervised *auxiliary tasks* to speed up the learning process that requires no additional feedback from the environment. The idea of additional auxiliary predictions is to help with the representational learning problem (Bengio, Courville, and Vincent 2013) and had also been useful to improve the robustness and stability of the learning process (Jaderberg et al. 2017). UNREAL proposes two auxiliary tasks: auxiliary control and auxiliary prediction that share the previous layers that the base agent uses to act. By using this jointly learned representation, the base agent learns to optimize extrinsic reward much faster and, in many cases, achieves better policies at the end of training. The UNREAL algorithm optimizes a single combined loss function with respect to the joint parameters of the agent that combines the A3C loss, $\mathcal{L}_{\text{A3C}}$, together with an auxiliary control loss $\mathcal{L}_{PC}$, an auxiliary reward prediction loss $\mathcal{L}_{RP}$ and a replayed value loss $\mathcal{L}_{VR}$. In contrast to A3C, UNREAL uses an experience replay buffer that is sampled with more priority given to interactions with positive rewards to improve the critic network.

Agent Modeling with A3C

In this section we first describe the challenges of opponent modeling in the context of reinforcement learning and multiagent systems, then we present our two main contributions: the AMS-A3C and AMF-A3C architectures.

Opponent modeling and multiagent systems

In a multiagent environment, agents interact at the same time with the environment and potentially with each other (Tuyls and Weiss 2012). These environments are commonly formalized as a Markov game $\langle S,\mathcal{N},A,T,R\rangle$, which can be seen as an extension of an MDP to multiple agents (Littman 1994). One key distinction is that the transition, T, and reward function, R, depend on the actions of all, $\mathcal{N}$, agents. Given a learning agent i and using the common shorthand notation $-\boldsymbol{i} = \mathcal{N}\setminus\{i\}$ for the set of opponents, the value function now depends on the joint action $\boldsymbol{a} = (a_i, \boldsymbol{a_{-i}})$, and the joint policy $\boldsymbol{\pi}(s,\boldsymbol{a}) = \prod_j \pi_j(s,a_j)$:

$$V_i^{\boldsymbol{\pi}}(s) = \sum_{\boldsymbol{a}\in A}\boldsymbol{\pi}(s,\boldsymbol{a})\sum_{s'\in S}T(s,a_i,\boldsymbol{a_{-i}},s') \\ [R(s,a_i,\boldsymbol{a_{-i}},s') + \gamma V_i(s')]. \quad (1)$$

The optimal policy is dependant on the other agents' policies:

$$\pi_i^*(s,a_i,\boldsymbol{\pi_{-i}}) = arg\max_{\pi_i} V_i^{(\pi_i,\boldsymbol{\pi_{-i}})}(s).$$

However, if the other agents' policies are stationary (can still be stochastic) then the problem can be reduced to a standard MDP where RL algorithms can be used to effectively learn a best response to those other agents, irrespective if the domain is cooperative or competitive. Our goal therefore is to accurately estimate the opponent/teammate policy at the same time that the agent is learning its respective (best response) policy. Since these two elements are linked to each other, below we propose two architectures that take advantage of this realization. In this work we show advantage of agent policy prediction with respect to non-learning agents. We leave as future work how to deal with learning agents.

AMS-A3C: Agent Modeling by parameter Sharing

This architecture builds on the concepts of *parameter sharing* and *auxiliary tasks*. Parameter sharing has been proposed in multiagent DRL as a way to reduce the number of parameters to learn and improve the performance. The idea is to perform centralized learning where agents share the same network (i.e., parameters) but the outputs represent different agent actions (Foerster et al. 2017).

Building on the same principle, in our architecture we want to also predict the opponent/teammate policies as well as the standard actor and critic, with the key characteristic that the previous layers will share all their parameters, see Figure 1 (b). The change in the architecture is accompanied by a modification in the loss function. In this case, we treat the learning of the other agents' policies as auxiliary tasks (Jaderberg et al. 2017) by refining the loss function as:

$$\mathcal{L}_{\text{AMS-A3C}} = \mathcal{L}_{\text{A3C}} + \frac{1}{\mathcal{N}}\sum_i^{\mathcal{N}}\lambda_{AM_i}\mathcal{L}_{AM_i}$$

where and λ_{AM_i} is weight term and $\mathcal{L}_{AM_i}$ is an auxiliary loss for opponent/teammate i:

$$\mathcal{L}_{AM_i} = -\frac{1}{M}\sum_j^M a_i^j\log(\hat{a}_i^j)$$

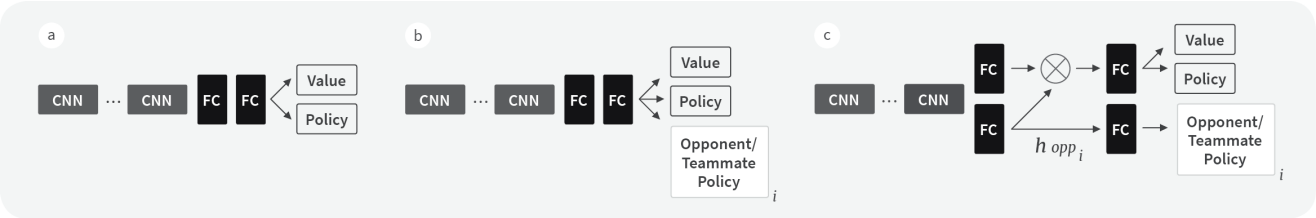


Figure 1: CNN represents convolutional layers, FC represents fully connected layers, and $\otimes$ represents an element-wise vector multiplication. (a) A3C outputs values and the agent's policy. (b) AMS-A3C is similar to A3C but adds a head that predicts the other agents' policies. (c) AMF-A3C aims to learn other agents' policy features in the latent space, h_{opp_i}, which are then used to compute the value and policy of the learning agent. Both approaches can be generalized to $\mathcal{N}$ opponents/teammates.

which is the supervised cross entropy loss between the observed one-hot encoded opponent/teammate action (ground truth), a_i^j, and the prediction, $\hat{a}_i^j$, for a trajectory of length M.

AMF-A3C: Agent Modeling by policy Features

The second architecture uses the concepts of *policy features* and auxiliary tasks. Hong et al. (2018) proposed a modified DQN architecture that conditions Q-values of the learning agent on *features* in the latent space[2] that also predict the opponent/teammate policy, i.e., policy features.

We take a similar approach but in the context of on-policy actor-critic methods, which means that policy features condition on both actor and critic. In this case, after the convolutional layers, the fully connected layers are divided in two sections, one specialized in the opponent/teammate policy and the other in the actor and critic (of the learning agent). Then, we directly use opponent/teammate policy features, h_{opp_i} vector, to be conditioned (via an element-wise multiplication) when computing the actor and critic, see Figure 1 (c). The loss function is similarly refined as follows:

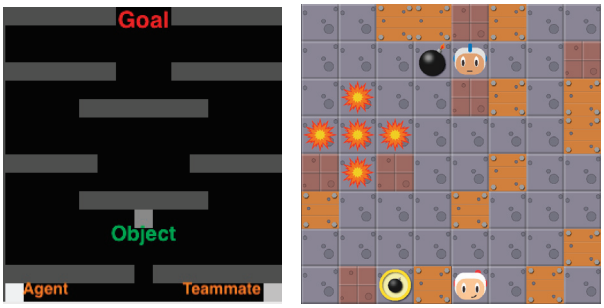
$\mathcal{L}_{\text{AMF-A3C}} = \mathcal{L}_{\text{A3C}} + \frac{1}{\mathcal{N}} \sum_i^{\mathcal{N}} \lambda_{AM_i} \mathcal{L}_{AM_i}$

Note that we described AMS-A3C and AMF-A3C in the general case with $\mathcal{N}$ agents in the environment. In the experiments we evaluated scenarios with one opponent or one teammate.

Implementation details

For A3C, AMS-A3C, and AMF-A3C we used 3 or 4 convolutional layers (depending on the domain), with 32 filters, 3×3 kernels, stride and padding of 1. For A3C and AMS-A3C the convolutional layers are followed with 2 fully connected layers with 128 hidden units each, followed by 2-heads: the critic has a single output for state-value estimate for the observation, and the actor has $|A|$ outputs for the policy probabilities for the given observation. For AMF-A3C, the fully connected layers have 64 units (to keep the same number of weights as AMS-A3C). For AMS-A3C and AMF-A3C, the opponent/teammate policy head has $|A_{opp}|$ outputs corresponding to the opponent/teammate policy. We used ELU activation functions. The parameters of all architectures have entropy weight of 0.01, a value loss weight of 0.5, a policy loss weight of 1, and a discount factor of 0.99. The parameters of the learning agent's policy are optimized using Adam with $lr = 0.0001, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 1 \times 10^{-8}$, and weight decay 1×10^{-5}. In the next section we compare different settings for λ_{AM}.

[2]Not to be confused with latent variables.



(a) (b)

Figure 2: (a) The coordinated multiagent object transport moving problem (Palmer et al. 2018). Two agents need to coordinate to pick up an object and delivery it to the goal zone. Our experiments use a stochastic teammate that moves with higher probability towards the object and then to the goal. (b) An example of the mini Pommerman with board size 8×8. The board is randomly generated varying the number of wood (light brown cell), walls (dark brown cell) and power-ups (yellow circle). Initial positions of the agents are randomized close to any of the 4 corners of the board.

Experiments

This section describes the two experimental domains: a cooperative multiagent transport moving problem and a competitive mini version of Pommerman. We then present the experimental results in terms of sensitivity of the loss weight parameter λ_{AM} for AMS-A3C and AMF-A3C in the coordination domain, and then we compare with A3C in terms of rewards for the two domains.

Domains and setup

Coordination This domain is inspired by Coordinated Multi-Agent Object Transportation Problems (CMOTPs) (Palmer et al. 2018), in which two agents are tasked with delivering one object to a goal within a

grid-world. The agents must locate and pick up the object by standing in the grid cells on the left and right hand side. The task is fully cooperative, i.e., objects can only be transported upon both agents grasping the item (this happens automatically when situated next to the object) and choosing to move in the same direction. Agents only receive a positive reward after placing the object in the goal, see Figure 2(a). Agents have 1900 time steps to complete this task, otherwise the object is reset to the starting position. The actions available to each agent are to either stay in place or move left, right, up, or down. We tested two teammates: one *hesitant* agent which moves randomly but with higher probability towards the object and once it has grasped it then moves with higher probability towards the goal; and a *stubborn* agent which prefers to follow a certain path after grasping the object (i.e., some actions are fully deterministic). Agents receive one observation per time step from the environment as a 16×16 pixel representation. We used 12 CPU workers in these experiments.

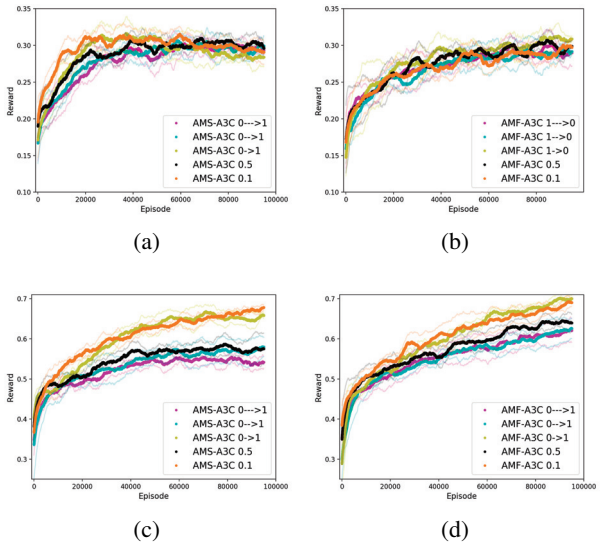
Competition The Pommerman environment (Resnick et al. 2018) is based on the classic console game Bomberman. Our experiments use the simulator in a mode with two agents, see Figure 2(b). Each agent can execute one of 6 actions at every time step: move in any of four directions, stay put, or place a bomb. Each cell on the board can be a passage, a rigid wall, or wood. The maps are generated randomly, albeit there is always a guaranteed path between any two agents. The winner of the game is the last agent standing and receives a reward of 1. Whenever an agent places a bomb it explodes after 10 time steps, producing flames that have a lifetime of 2 time steps. Flames destroy wood and kill any agents within their blast radius. When wood is destroyed either a passage or a power-up is revealed. Power-ups can be of three types: increase the blast radius of bombs, increase the number of bombs the agent can place, or give the ability to kick bombs. A single game of two-player Pommerman is finished when an agent dies or when reaching 800 timesteps.

We considered the *rule-based* opponent baseline that comes within the simulator (a.k.a. SimpleAgent). Its behaviour is stochastic since it collects power-ups and places bombs when it is near an opponent. It is skilled in avoiding blasts from bombs. It uses Dijkstra's algorithm on each time-step, resulting in longer training times.

We evaluated our two proposed architectures and the standard A3C against the opponents mentioned above. In all cases we provided learning agents with dense rewards and we did not tune those reward terms. In our setting the entire board is visible and agents receive one observation per time step from the environment as a $18 \times 8 \times 8$ matrix which contains the current time step board description of the board for the current time step, similar to Resnick et al. (2019).

Results

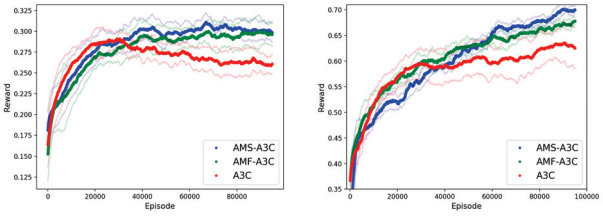
Sensitivity of λ_{AM} In the first set of experiments we used the coordination domain to evaluate different weights for the opponent modeling loss value: annealing $\lambda_{AM} = 1.0 \rightarrow 0.0$ varying discount rates exponentially



(a) (b) (c) (d)

Figure 3: Comparison for the weight for the opponent modeling loss value, λ_{AM}, annealing $1.0 \rightarrow 0.0$ with varying discount rates (exponentially) or fixing the value. Learning curves in the coordination domain with the *hesitant* teammate for (a) AMS-A3C and (b) AMF-A3C: no significant variation; with the *stubborn* teammate for (c) AMS-A3C and (d) AMF-A3C: best results were obtained with $\lambda_{AM} = 0.1$

$\{0.999, 0.9999, 0.99999\}$ or keeping the value fixed with $\lambda_{AM} = \{0.1, 0.5\}$. With the *hesitant* teammate both AMS-A3C and AMF-A3C show similar behavior for all the evaluated parameters (better than A3C), see Figures 3(a)-(b). When testing with the *stubborn* teammate we observed more variation among parameters, for both AMS-A3C and AMS-A3C using a fixed $\lambda_{AM} = 0.1$ or quickly annealing with 0.999 gave the best results, see Figures 3(c)-(d). Our hypothesis is that this teammate is easier to learn and the network does not need too much weight on their modeling; instead it can focus on policy learning.



(a) (b)

Figure 4: Coordination domain: Learning curves with two different teammates (a) *hesitant* and (b) *stubborn* in the coordination problem. Vanilla A3C shows instability and even reduces its rewards after some episodes, in contrast, AMS-A3C and AMF-A3C are more stable, with lower variance and higher rewards.

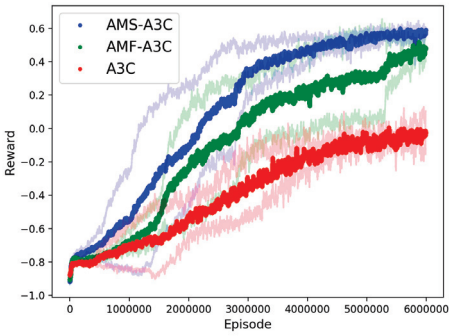
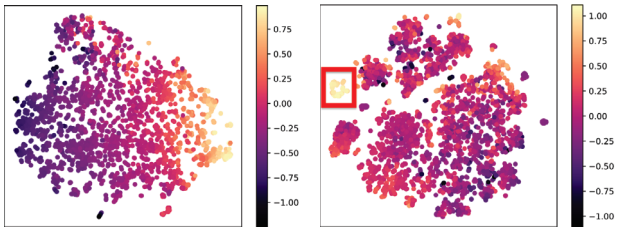


Figure 5: Competition domain: Moving average over 10k games of the rewards (shaded lines are standard deviation over 5 runs) obtained by the two proposed architectures and A3C against the *rule-based* opponent: AMS-A3C and AMF-A3C obtained significantly higher scores than A3C.

Coordination Using the best parameters for AMS-A3C and AMF-A3C we compare to A3C. Figure 4 depicts learning curves[3] (average with standard deviations of 10 runs) where it can be seen that in the first part of the learning (30k episodes), all learning agents behave similarly, however, in the long run AMS-A3C and AMF-A3C obtained higher rewards than A3C (AMS-A3C was statistically significantly better than A3C from episode 60k, $\alpha = 0.05$). We noted that against the *hesitant* teammate A3C decreases its rewards, likely because of its on-policy nature, see Figure 4(a). In contrast, AMS-A3C and AMF-A3C show stability and start increasing their rewards. When facing the *stubborn* teammate, AMS-A3C and AMF-A3C show less variance than A3C due to their accurate agent modeling (AMS-A3C is statistically significant over A3C from episode 90k with $\alpha = 0.05$), see Figure 4(b). Examining the trained agents, AMS-A3C and AMF-A3C show better coordination skills once the object is grasped compared to vanilla A3C, i.e., agents reached the goal faster once grabbing the object.

Competition One clear distinction from the previous domain is that it is more elaborate and stochastic (board is randomized and changes depending on the agents' actions). In this experiments we set $\lambda_{AM} = 0.01$ and we evaluate against the *rule-based* opponent. In this case, we let the learning agents train for 6 million episodes to guarantee convergence ($\approx$ 3 days of training with 50 workers). Results are depicted in Figure 5 (with standard deviations over 5 runs), where it can be seen that AMS-A3C and AMF-A3C both clearly outperform A3C in terms of rewards (AMS-A3C is statistically significant over A3C from episode 3.5M and AMF-A3C from 5.5M, $\alpha = 0.05$). When observing the policies generated we noted that during game play the agents trained with AMS-A3C and AMF-A3C were able to make the opponent commit suicide by blocking its moves (in Pommerman, if two agents simultaneously want to move to the same cell, they both stay in their current locations) and make it stand on the path of the flames, in contrast to A3C which was unable to learn this strategy and obtained lower rewards.

[3]Because of the stochasticity of the opponent actions an upper bound of the expected reward is ≈ 0.7 (experimentally computed) with the selected parameters.



(a) (b)

Figure 6: T-SNE analysis from trained (a) A3C and (b) AMS-A3C agents from 100 episodes (colors are obtained from the value head). AMS-A3C t-SNE shows many more clusters, in particular, the cluster highlighted on the left corresponds to states when AMS-A3C is about to win the game (value close to 1).

Lastly, we performed a visual analysis similar to Zahavy, Ben-Zrihem and Mannor (2016). We took trained agents of A3C and AMS-A3C and for 100 episodes we recorded both the activations of the last layer and the value output. We applied t-SNE (Maaten and Hinton 2008) on the activations data (as input) and the value outputs (as labels). Figure 6 depicts the t-SNEs where it can be seen that AMS-A3C has more well-defined clusters than A3C's, in particular the highlighted cluster on the left represents states when AMS-A3C is about to win the game because it can accurately predict the opponent's moves, which implies values close to 1.

Conclusions

Deep reinforcement learning has shown outstanding results in recent years. However, there are still many open questions regarding different recent learning algorithms. We take as base a major representative of actor-critic methods, i.e., A3C, and propose two architectures that are designed to do agent modelling as an auxiliary task. This means that at the same time the network improves the representation learning, it will also aim to learn other agents policies. Even though auxiliary tasks are not new, their use in deep RL and opponent modeling is still not well studied. Our work serves as an important stepping stone in this direction by proposing two architectures that improve learning when doing opponent/teammate modeling in deep RL. Our architectures AMS-A3C and AMF-A3C are inspired by multi-agent DRL concepts: parameter sharing and opponent policy features. We experimented in both cooperative and competitive domains. In the former, our proposals were able to learn coordination faster and more robustly compared to the vanilla A3C. In the latter, our agents were able to predict opponent moves in complex simultaneous move, Pommerman, and successfully obtain a best response that resulted in higher scores in terms of rewards. As future work, we are interested in exploring self-play, learning agents, and mixed (coordination-competition) environments.

References

Albrecht, S. V., and Stone, P. 2018. Autonomous agents modelling other agents: A comprehensive survey and open problems. *Artificial Intelligence* 258:66–95.

Banerjee, B., and Peng, J. 2005. Efficient learning of multi-step best response. In *AAMAS*, 60–66.

Bellemare, M. G.; Dabney, W.; Dadashi, R.; Taiga, A. A.; Castro, P. S.; Roux, N. L.; Schuurmans, D.; Lattimore, T.; and Lyle, C. 2019. A geometric perspective on optimal representations for reinforcement learning. *arXiv preprint arXiv:1901.11530*.

Bengio, Y.; Courville, A.; and Vincent, P. 2013. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence* 35(8):1798–1828.

Borovikov, I.; Zhao, Y.; Beirami, A.; Harder, J.; Kolen, J.; Pestrak, J.; Pinto, J.; Pourabolghasem, R.; Chaput, H.; Sardari, M.; et al. 2019. Winning isn't everything: Training agents to playtest modern games. In *AAAI Workshop on Reinforcement Learning in Games*.

Brown, G. W. 1951. Iterative solution of games by fictitious play. *Activity analysis of production and allocation* 13(1):374–376.

Carmel, D., and Markovitch, S. 1995. Opponent Modeling in Multi-Agent Systems. In *IJCAI*. Springer-Verlag.

Caruana, R. 1997. Multitask learning. *Machine learning* 28(1):41–75.

Chalkiadakis, G., and Boutilier, C. 2003. Coordination in Multiagent Reinforcement Learning: A Bayesian Approach. In *AAMAS*.

Foerster, J. N.; Nardelli, N.; Farquhar, G.; Afouras, T.; Torr, P. H. S.; Kohli, P.; and Whiteson, S. 2017. Stabilising Experience Replay for Deep Multi-Agent Reinforcement Learning. In *ICML*.

He, H.; Boyd-Graber, J.; Kwok, K.; and Daume, H. 2016. Opponent modeling in deep reinforcement learning. In *ICML*, 2675–2684.

Hernandez-Leal, P.; Kartal, B.; and Taylor, M. E. 2018. Is multiagent deep reinforcement learning the answer or the question? A brief survey. *arXiv preprint arXiv:1810.05587*.

Hong, Z.-W.; Su, S.-Y.; Shann, T.-Y.; Chang, Y.-H.; and Lee, C.-Y. 2018. A Deep Policy Inference Q-Network for Multi-Agent Systems. In *AAMAS*.

Jaderberg, M.; Mnih, V.; Czarnecki, W. M.; Schaul, T.; Leibo, J. Z.; Silver, D.; and Kavukcuoglu, K. 2017. Reinforcement Learning with Unsupervised Auxiliary Tasks. In *ICLR*.

Lample, G., and Chaplot, D. S. 2017. Playing FPS Games with Deep Reinforcement Learning. In *AAAI*, 2140–2146.

Lanctot, M.; Zambaldi, V. F.; Gruslys, A.; Lazaridou, A.; Tuyls, K.; Pérolat, J.; Silver, D.; and Graepel, T. 2017. A Unified Game-Theoretic Approach to Multiagent Reinforcement Learning. In *NIPS*.

Littman, M. L. 1994. Markov games as a framework for multi-agent reinforcement learning. In *ICML*, 157–163.

Maaten, L. v. d., and Hinton, G. 2008. Visualizing data using t-SNE. *Journal of Machine Learning Research* 9(Nov).

Mirowski, P.; Pascanu, R.; Viola, F.; Soyer, H.; Ballard, A. J.; Banino, A.; Denil, M.; Goroshin, R.; Sifre, L.; Kavukcuoglu, K.; et al. 2017. Learning to navigate in complex environments. *ICLR*.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; and Riedmiller, M. 2013. Playing Atari with Deep Reinforcement Learning. *arXiv preprint arXiv:1312.5602v1*.

Mnih, V.; Badia, A. P.; Mirza, M.; Graves, A.; Lillicrap, T.; Harley, T.; Silver, D.; and Kavukcuoglu, K. 2016. Asynchronous methods for deep reinforcement learning. In *ICML*, 1928–1937.

Palmer, G.; Tuyls, K.; Bloembergen, D.; and Savani, R. 2018. Lenient Multi-Agent Deep Reinforcement Learning. In *AAMAS*.

Rabinowitz, N. C.; Perbet, F.; Song, H. F.; Zhang, C.; Eslami, S. M. A.; and Botvinick, M. 2018. Machine Theory of Mind. In *ICML*.

Raileanu, R.; Denton, E.; Szlam, A.; and Fergus, R. 2018. Modeling Others using Oneself in Multi-Agent Reinforcement Learning. In *ICML*.

Resnick, C.; Eldridge, W.; Ha, D.; Britz, D.; Foerster, J.; Togelius, J.; Cho, K.; and Bruna, J. 2018. Pommerman: A multi-agent playground. *AIIDE Multi-Agent Workshop*.

Resnick, C.; Raileanu, R.; Kapoor, S.; Peysakhovich, A.; Cho, K.; and Bruna, J. 2019. Backplay:" man muss immer umkehren". *AAAI-19 Workshop on Reinforcement Learning in Games*.

Schadd, F.; Bakkes, S.; and Spronck, P. 2007. Opponent modeling in real-time strategy games. In *GAMEON*, 61–70.

Shelhamer, E.; Mahmoudieh, P.; Argus, M.; and Darrell, T. 2017. Loss is its own reward: Self-supervision for reinforcement learning. *ICLR workshops*.

Suddarth, S. C., and Kergosien, Y. 1990. Rule-injection hints as a means of improving network performance and learning time. In *Neural Networks*. Springer. 120–129.

Tampuu, A.; Matiisen, T.; Kodelja, D.; Kuzovkin, I.; Korjus, K.; Aru, J.; Aru, J.; and Vicente, R. 2017. Multiagent cooperation and competition with deep reinforcement learning. *PLOS ONE* 12(4):e0172395.

Torrado, R. R.; Bontrager, P.; Togelius, J.; Liu, J.; and Perez-Liebana, D. 2018. Deep Reinforcement Learning for General Video Game AI. *arXiv preprint arXiv:1806.02448*.

Tuyls, K., and Weiss, G. 2012. Multiagent learning: Basics, challenges, and prospects. *AI Magazine* 33(3):41–52.

Zahavy, T.; Ben-Zrihem, N.; and Mannor, S. 2016. Graying the black box: Understanding DQNs. In *ICML*.

Zhao, R., and Szafron, D. 2009. Learning character behaviors using agent modeling in games. In *AIIDE*.

Zheng, Y.; Meng, Z.; Hao, J.; Zhang, Z.; Yang, T.; and Fan, C. 2018. A deep bayesian policy reuse approach against non-stationary agents. In *NeurIPS*. 962–972.