Learning to Maximize Return in a Stag Hunt Collaborative Scenario through Deep Reinforcement Learning

Andrei Nica*, Tudor Berariu*†, Florin Gogianu†, and Adina Magda Florea*

**Faculty of Automatic Control and Computers*
University Politehnica of Bucharest
Email: {andreic.nica, tudor.berariu}@gmail.com
adina.florea@cs.pub.ro

†*Bitdefender*
Email: {fgogianu, tberariu}@bitdefender.com

***Abstract*—In this paper we present a deep reinforcement learning approach for learning to play a time extended social dilemma game in a simulated environment. Agents face different types of adversaries with different levels of commitment to a collaborative strategy. Our method builds on recent advances in policy gradient training using deep neural networks. We investigate multiple stochastic gradient algorithms such as Reinforce or Actor Critic with auxiliary tasks for faster convergence.**

***Keywords*-deep reinforcement learning; social dilemmas; policy gradient**

I. INTRODUCTION

Last years' research yielded continuous progress in deep reinforcement learning algorithms for agents placed in simulated scenarios. A plethora of novel model-free algorithms explored the advantages of using deep neural networks for predicting state, or action values, and for approximating policies for continuous control in various visual environments (e.g. Atari [10], Vizdoom [6], or Minecraft [14], or board games (Go). Few recent studies focused on reinforcement learning in multi-agent setups where several learning entities must cooperate in competing, or collaborative games. The problem of non-stationarity, one of the core challenges in reinforcement learning, is aggravated by the continuously changing behaviors of the other actors.

The return scheme of a multi-agent reward-based scenario is sometimes best described from a game theory perspective. A payoff matrix summarizes the gains for all players as a function of their chosen strategies. There is a large corpus of research in Artificial Intelligence on how agents can maximize their expected return through learning from iterated interactions. All those results focused on learning in stateless setups following various payoff schemes (e.g. prisoner's dilemma). Also, theoretical properties such as Nash equilibrium or Pareto optimality are assessed for learned strategies. A more difficult problem is identifying such a situation in a more complex scenario (e.g. during a chess game or a collaborative prey hunting) where the reward is a consequence of a (possibly large) sequence of decisions based on partial raw observations of the environment. This paper presents a deep reinforcement learning approach to such a scenario, where agents are situated in a cooperative episodic game based on stag hunt. Players have no persistent memory from one episode to the next.

The stag hunt (also known as trust dilemma) is a game where each agent needs to choose between social collaboration for a higher reward, and a lower-risk but less rewarding individual solution. The original formulation refers to a group of hunters where each either goes with the group in a stag chase or gets a hare on its own. Of course, a hare is worth less than a stag.

There's a fundamental change between stateless one-shot games and episodic ones, even if the underlaying payoff matrix is identical, and direct communication between players is not allowed. If there is no initial irrevocable commitment to a strategy agents can observe the state of actions and choose to change their minds during the game. In such scenarios understanding the intentions of the other players before committing to a strategy is crucial in achieving high return policies. The game we tackle in this paper exemplifies this by placing together two agents in a toxic environment. They have to choose between cooperating in a swine hunt, and leaving the game for a smaller reward.

One of the goals of our research was to see if agents understand the underlaying macro scheme (which is an instance of the stag hunt game) through deep reinforcement learning techniques and not by being explicitly taught to choose between two strategies. Abstracting a high description of the interactions with the environment would be beneficial in many ways. For example, good performance in this game might offer a valuable prior experience before learning a second similar task (transfer learning). Knowing how to deal with a trust dilemma in general requires an agent facing a new situation just to learn how to interpret perception, and how to affect that specific environment reducing the burden on the learning process.

The approach taken in our work is based on on-line policy gradient methods, more precisely variations of REINFORCE and Actor-Critic algorithms with auxiliary tasks. All predictors are deep convolutional networks followed by recurrent layers trained using gradient-based updates per episode.

Policy Gradient methods are known to suffer from high variance, stability being achieved through combining asynchronous experiences of several players, or through offline learning from a memory. Both approaches tackle the correlation between consecutive observations in a typical reinforcement learning setup.

The scenario we tried to solve, called Malmo Platform PigChase[1] is one of the scenarios in Malmo Platform a reinforcement learning environment built on top of Minecraft. The higher complexity of this medium leads to a high resource, slow environment, which at first glance is incompatible with training deep neural predictors that require millions of samples before achieving high returns. For this reason we implemented a simplified replica of the original setup approximating the dynamics of the original setup which we will call from now on the PigChase Replica environment. We had two objectives in mind for this simplified environment: to be many times faster than the original and to work in batch mode.

We then trained policy gradient based agents in this fast setup only to fine tune them at the end on the original one. Our intuition was that once the agent understands the general dynamics and the underlying trust dilemma, moving him on a similar environment requires small adaptation.

In what follows, section II formally describes the reinforcement learning problem, III presents the environment, and other section follow. We provide code[2] for all the models discussed in this overview.

II. Reinforcement Learning

A. Background and Notation

In a classic reinforcement learning setting an agent interacts with an environment described by a Markov Decision Process object $(\mathcal{S}, \mathcal{A}, P, r)$. We consider a finite and discrete state-action space within a finite, discounted, horizon. At each time-step t the agent observers the current state $s \in \mathcal{S}$ and takes the action $a \in \mathcal{A}$ by following the policy $\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}_+$, a mapping from the state-action space to the probability of taking action a given the observation. It then moves to the next state $s_{t+1} \sim P(\cdot, s_t, a_t)$, the probability of transitioning from one state to another given action a and to receive reward $r : \mathcal{S}\times\mathcal{A} \to [R_{MIN}, R_{MAX}]$.

The general reinforcement problem is then finding a policy π that maximizes the expected total discounted reward $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$, where $\gamma \in (0, 1)$ is the discount term controlling the importance of future rewards.

We are now in a position to define several functions for describing the utility of the states an agent visits and the actions it takes. The value of a state-action pair (s, a) and that of a state s when following a policy π is given by:

$$Q^{\pi}(s, a) = \mathbb{E}\left[R_t | s_t = s, a_t = a, \pi\right] \tag{1}$$

$$\begin{aligned} V^{\pi}(s) &= \mathbb{E}\left[R_t | s_t = s, \pi\right] \\ &= \mathbb{E}_{a\sim\pi(s)}\left[Q^{\pi}(s, a)\right] \end{aligned} \tag{2}$$

A related function is the difference between the action-value and the value function, which can be seen as a relative measure of the importance of taking action a over the expected performance of the policy from state s:

$$A^{\pi}(s, a) = Q^{\pi}(s, a) - V^{\pi}(s) \tag{3}$$

There are two major model-free approaches to solving and MDP and learning a policy: estimating state-action values and policy learning. The first approach iteratively optimizes a loss based on the temporal difference error, $\mathcal{L}_i(\theta_i) = \mathbb{E}\left[(r + \gamma \max_{a'} Q((s', a'; \theta_i^-) - Q(s, a; \theta_i)^2\right]$ and has been shown to achieve state of the art results in general game playing in discrete domains [10], [12], [15], [17]. From the Q values optimized as such we can then derive a policy using an exploration technique such as epsilon-greedy.

In this setting we are however interested in stochastic methods, directly optimizing the policy we are trying to find. Specifically, we are parameterizing the policy π and optimizing the parameters θ in the direction of the episodic return R_t using gradient ascent. The gradient of the performance function in REINFORCE [18], the first in this family of methods, is then:

$$\nabla_{\theta}\mathbf{J}_{\theta} = \mathbb{E}_{s,a}\nabla_{\theta}\log\pi(s, a)R_t \tag{4}$$

The method above suffers from the high variance of the Rt estimator and can be improved by subtracting a baseline from it, obtaining a gradient $\nabla_{\theta}\mathbb{E}_{s,a}\log\pi(s, a)R_t - b(s)$.

Notice the quantity $R_t - b(s)$ can be seen as the advantage function $A^{\pi}(s, a)$ defined in equation 3 and this gives rise to a family of **actor-critic** methods in which the actor takes actions according to the policy π and is optimized in the direction of the gradient of the advantage function provided by the critic, while the critic is updated with the temporal difference learning error as in the state-action methods described above.

B. Related Work

Our work is concerned with finding an optimum policy in environments that resemble iterated matrix games [1], [11]. While reinforcement learning algorithms have been used in iterated matrix games [3], [19], our setting is closer

[1]https://www.microsoft.com/en-us/research/project/project-malmo/
[2]https://github.com/village-people/flying-pig

to Sequential Social Dilemmas [7] that take into account that real-world social dilemmas are temporally extended. However we are not directly concerned with modeling the payoff matrix of these games, but to learn an optimum policy when playing with a cooperative agent or a defector. We are rather interested in the sample efficiency of an agent trained in a such scenario and the robustness of the policy to environments with different cooperation - defection ratios.

For training our models on the PigChase Replica Environment we used methods based on the REINFORCE [18] algorithm coupled with various variance reduction techniques as described in [13]. For the transfer learning experiments with fine-tuning on the PigChase Malmo Platform we employed a distributed architecture inspired by [9] and [2].

While informed by the multi-agent literature, our setting can be seen as having the other agent as part of the environment; we do not train multiple agents as in a multi-agent environment. However our agent is required to learn a behaviour that is effective when playing with collaborative or greedy agents.

III. Pig Chase Challenge

The target learning environment for the models presented in this paper is Pig Chase, a collaborative two-agent game published by Microsoft as part of a competition called Malmo Collaborative AI Challenge[3]. This document builds on our insights from the solution submitted to that contest.

Pig Chase is a Malmo Platform scenario where two players act in a small environment along with a swine. The game is a stag hunt example where the two agents have to collaborate without direct communication in order to catch the running pig. If the players succeed in cornering the swine they both get a high reward (25). Both agents also have the option of exiting the environment through one of the two gates for a smaller reward (5). Since the pig can run away and there's also a negative reward (-1) for each time step spent in an episode, the agents might have to abandon the swine chase if they find it as being ineffective in terms of expected return. Since each player can suddenly switch from chasing the pig to leaving the environment at any step in the episode, a trust issue arises. Therefore in committing to a strategy agents need to take into account not just the personal benefit brought by it, but also the risk of the other agent abandoning the collaboration.

The game setup offered two state representations: an image of the 3D environment from the player's point of view, and also an abstract top view representation of the map as shown in Figure 1. The visual representation is more challenging as offers a partial perception of the current state of the world, while also needing more complicated feature extraction. The abstract top-view simplifies the observation space as it removes the uncertainty over unperceived parts of the world, and it also drastically reduces the input space dimensionality. In our work we chose to use the abstract representation for two reasons. First, we were interested in investigating the feasibility of learning to predict the other agent's intentions (i.e. the probability of it committing to the collaborative strategy) as recent years showed that DQN can already successfully learn to induce good policies from raw visual perceptions. Second, we wanted the ability to approximately reproduce the scenario in the PigChase Replica environment and only concerning ourselves with the abstract top-view we managed to replicate the approximate dynamics of the original Malmo Platform PigChase scenario but at a much higher frames per second rate. Specifically we were able to generate batches of up to 1024 games long of up to 25 steps every two seconds.

[3]https://www.microsoft.com/en-us/research/academic-program/collaborative-ai-challenge/

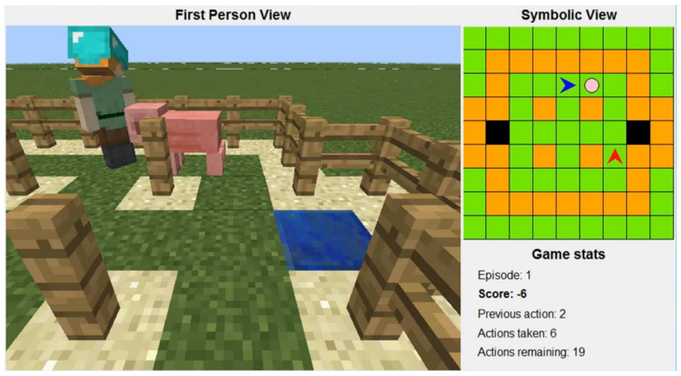

Figure 1. Showing the two possible state representations provided by the Microsoft Malmo Platform PigChase environment. Left: actual 3D image as provided by the Mamlo Platform. Right: symbolic view encoding the agents' positions and orientations and the swine's position along with a representation of the map.

IV. Approach

Early experimentation involved training feed-forward parametrized estimators with DQN [10], Double DQN [15] in order to compensate for over-estimation effects early in the training and policy-gradient based methods.

Following these early trials we decided that policy-gradient based methods with recurrent units could provide us with a good baseline to build upon. Our agent consisted of a deep convolutional neural network trained to learn a policy through stochastic gradient updates. We implemented an online learning algorithm since we tackled the large variance that is characteristic to direct policy learning by using batches of episodic experiences.

We experimented with REINFORCE, a stochastic policy gradient method [18], and also we ran experiments with a an advantage-like baseline as described in [9]. Finally we experimented with auxiliary reward heads.

A. Neural Network Model

The network we ran all our experiments with is a four layer convolutional neural network for feature extraction fed into two successive GRU layers. Next are two fully connected layers and the final softmax, value, and auxiliary reward heads.

The state representation we used during training was a 18x9x9 tensor, with three layers for sand, grass and lapis blocks and five layers for each of the two agents and the pig, encoding their position and orientation.

B. The REINFORCE algorithm

In our approach we tried several baselines for the REINFORCE algorithm, but one in particular yielded outstanding results. Since training was based on batches of experiences from this short-horizon extremely toxic episodic environment, we considered that the current time step is a crucial factor in deciding the risk for one strategy or the other (assuming that agents are indeed learning to abstract such a binary decision) therefore we computed a baseline from all discounted returns observed in states from all episodes at the same time step. This averaged return provided a different baseline at each time step in the episode.

$$\nabla_{\theta}^{(t)}\pi = \nabla_{\theta} \log \pi \left(R_t - \mathbb{E}\left[R_{\pi}^{(t)} \right] \right) \tag{5}$$

In Formula 5 the expected return for time step t is estimated from the current batch according to Formula 6

$$\mathbb{E}\left[R_{\pi}^{(t)} \right] \approx \sum_{i}^{BS} \sum_{\tau=t}^{T_i} \gamma^{\tau-t} r_i^{(\tau)} \tag{6}$$

C. Auxiliary Tasks

While the recurrent policy gradient model was able to learn a good policy with good sample-efficiency we tried to provide our model with additional cost functions designed to help learning relevant features for the present task as first developed in [4], [8].

For the first auxiliary task we trained the agent on predicting the instantaneous reward at the next step in order for our model to learn faster about states and situations leading to high reward.

The second auxiliary task we trained with was next map prediction. We first considered fully generating the next map, complete with the future position of the Challenger Agent and the Pig, hoping that this would help our agent determine the unknown policy of the Challenger Agent based on its moves. We first considered feeding the hidden states of the recurrent layers into a deconvolution for generating the next state of the map, however we observed a severe slowdown during learning when training this way. Therefore we set up to predict a random coordinate on the (18, 9, 9) state representation we used for our agents. At the start of each episode we picked a random coordinate to be predicted at each time-step. We hypothesize this additional cost function helps our agent to learn faster the dynamics of the environment and the given policy of the Challenger Agent during each episode.

D. Training

When running experiments designed to target the evaluation procedure of the Malmo Platform we employed a two-stage training process as described below.

Pre-training on the PigChase Replica Environment. As mentioned above we developed a secondary environment that approximates the dynamics of the Malmo-Challenge world in the top-down view. We used this environment to generate large batches of 1024 variable length episodes, doing an optimization step on each batch using RMSProp. We used batch normalization between the convolutional layers as we noticed it improves the sample-complexity of our model and allows for higher learning rates. This initial pre-training phase allowed us easy quick experimentation with various models and, more importantly, a good prior when training our model on the Malmo-Challenge.

Training on the Malmo-Challenge environment. We used the full pre-trained model and a custom StateBuilder to further train our agent on the Malmo-Challenge environment. For this phase we started multiple environments and employed a training scheme inspired by GA3C [2], collecting prediction requests from all the workers and doing batched prediction on a single model. A separate training process is doing optimization steps on batches of 128 episodes. We noticed best results in this phase using Adam optimisation with a smaller-learning rate.

V. Results

In what follows insights from our comparative tests are presented with this section ending with the results obtained with our final model being evaluated on the original Malmo Platform Pig Chase environment.

The model we selected for fine tuning and evaluation on Malmo Pig Chase is the advantage REINFORCE training method with return, next state depth, and instant reward predictions trained as auxiliary tasks. The three auxiliary losses are all mean squared errors. We refer to this model as the `standard` one. In the following results whenever a parameter, or another training aspect is not explicitly mentioned one can assume it is as in our standard model.

If it is not otherwise specified the following plots are for $1024 * 10^4$ episodes. That means that for batches of 1024 episodes 10000 optimization steps were performed and reported in the plots. For batches of 128 the Adam optimizer took 80000 steps. Plot lines are synchronized based on the number of episodes used in training. Each plot line shows values obtained by averaging $50*(1024/batch\_size)$ consecutive observations in order to smooth the lines and make them comparable between different batch sizes. All

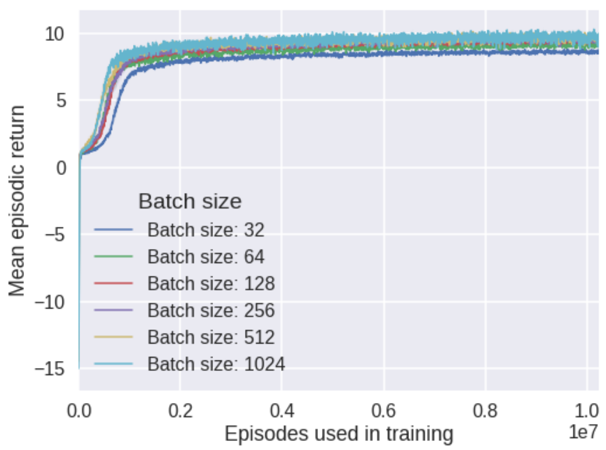

Figure 2. The standard model trained on different batch sizes for the same number of episodes. Values represent mean un-discounted episodic return averaged from five identical runs with different seeds.

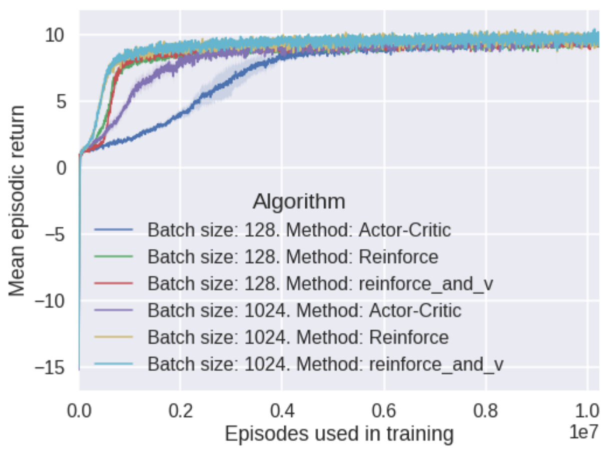

Figure 3. Comparing various policy learning methods for batches of size 1024 and 128.

plot lines represent the mean of five identical experiments with different seeds for the random number generators. The adversary is an A* focused agent with probability $p = .7$, the others being random players.

Our algorithms learn stochastic policies in an on-line fashion, therefore no separate evaluation was performed. We tried in a few cases to freeze training and to evaluate by eliminating noise, and/or taking the action with the highest probability instead of sampling from the policy, but we did not observe a consistent benefit from this. We leave this for further investigations.

A. *Choosing the batch size*

Since one of the pillars of our approach was the use of the Pig Chase Replica environment as a fast pre-training solution, we investigated first how training is affected by the batch size. Small batches lead to high-variance due to the bad approximation of the real cost function at each step, but larger batches sometimes harm training as well [5].

Figure 2 shows that the optimizing the `standard` model with Adam leads to similar sample complexities for batches between 32 and 1024. The largest batches proved to be advantageous both in terms of training speed, and final performance. Plot in Figure 2 and further investigation of the results showed that batches of 1024 and 512 yield similar models in terms of performance. We chose to continue our experiments with batches of 1024 as they are the most efficient in terms of time consumption due to efficient tensor operations on modern GPUs.

In the following experiments (Figures 3 through 5) we trained with batches of 1024 when explored various hyperparameters, but we doubled the observations with tests on batches of 128 just to make sure there is no important difference when doing faster optimization steps on smaller batches.

B. *Policy Gradient methods*

We first went for an on-line batch advantage actor-critic approach but we had difficulties training the critic to provide good state-values to bootstrap the training objective for the actor. We then removed the critic and policy trained faster through standard REINFORCE. Since the games had a small horizon of maximum 25 steps, we considered a different baseline as described in Equation 5. Keeping the value prediction as a second loss but not for bootstrapping in policy optimization, proved to be beneficial as plots in Figure 3 shows.

It is worth mentioning that actor-critic methods reach similar performance but with worse sample complexity than the advantage reinforce update.

We kept the advantage-based reinforce learning with value prediction as the reference model for the next experiments.

C. *Auxiliary Tasks*

As described in previous section, we investigated the effect of adding auxiliary tasks to the optimized objective. The auxiliary tasks help in finding better representations faster in preliminary studies on small batches and for agents trained directly on Malmo. Figure 4 infirms their use for large batches. We made several attempts in weighting the auxiliary losses, but for our model so far nothing brought any improvement.

D. *Exploration and avoiding pseudo-deterministic policies*

In order to avoid pseudo-deterministic (almost one-hot encoded) policies, and to ensure some level of exploration, we first applied clamping on the policy values in interval $[.1, .9]$ before sampling during playing. We then investigated

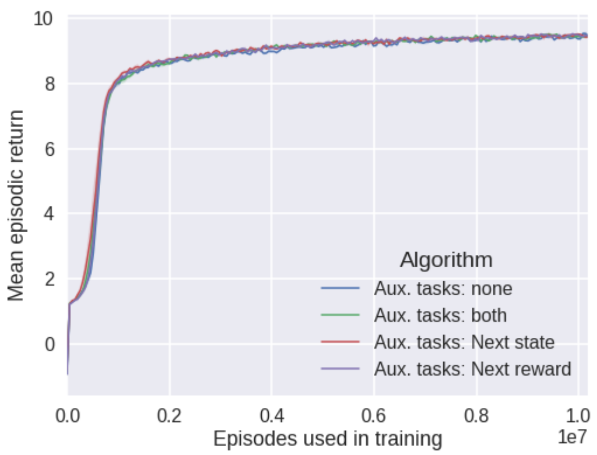

Figure 4. Learning averaged on five seeds with different combinations of auxiliary tasks. No improvement for large batches

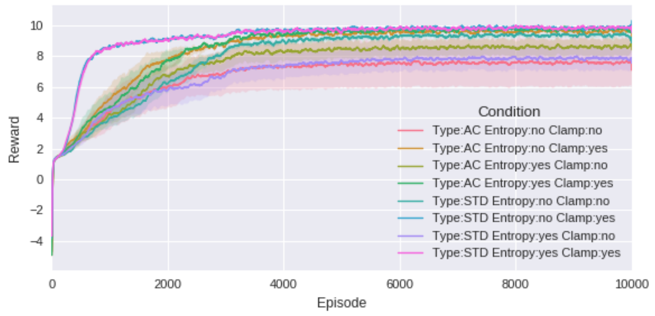

Figure 5. Learning averaged on five seeds with different combinations of exploration: clamping and entropy regularization

if other methods such as adding an entropy regularization term as in as in [9] for preventing the policy becoming deterministic early in the training. Although this proved to be beneficial for both Actor-Critic and our Reinforce algorithms as shown in Figure 5, it did not beat the more practical clamping method.

E. Using Batch Normalization on the last linear layer

In what follows we describe a practical problem encountered during training our agents. Using uni-dimensional batch normalization before the last linear transformation in our neural predictor proved to be extremely advantageous for learning. However, when we fine-tuned or evaluated our agents in single-instance mode on the original Malmo Pig Chase environment, the performance was severely degraded. There is a known problem in using neural models containing batch normalization and trained on large batches of examples. These models usually have performance issues when they are applied to single observations.

We investigated several approaches: (1) removing the last

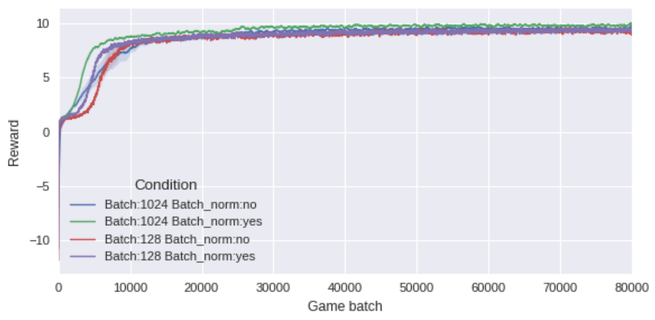

Figure 6. Learning averaged on five seeds with, and without batch normalization on the last linear layer. STD refers to our `standard` model.

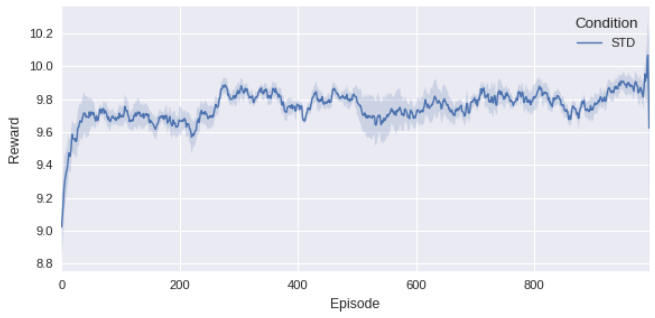

Figure 7. Fine tuning the last Batch Normalization layer with batches of size 1.

batch normalization layer which resulted in a performance loss and a larger training time needed to achieve the best policy, (2) training a network with batch normalization and then post-tune it on batches of one, and (3) training a network with batch normalization, removing the batch normalization layer and fine tuning it for a few epochs. Results in Figure 6 show the difference in training with and without the batch normalization layer.

The most time efficient method from the three above was to train with batch normalization and then remove this layer and fine tune the network for a short time taking advantage again of large batches. The average evolution of this fine tuning process can be visualized in Figure 7.

F. Evaluating agents against the two different types of adversaries

We trained our agents on batches of adversaries sampled at random based on a Bernoulli distribution in accordance with the Microsoft AI Challenge rules. Specifically, the other agent was an A* focused player with probability $p = 0.7$, or a random player with probability $1 - p = 0.3$. We trained our agents on batches of 1024 episodes performing a policy update after every batch.

We wanted to see if our trained agents learned some policy situated in between optimal ones for playing with each

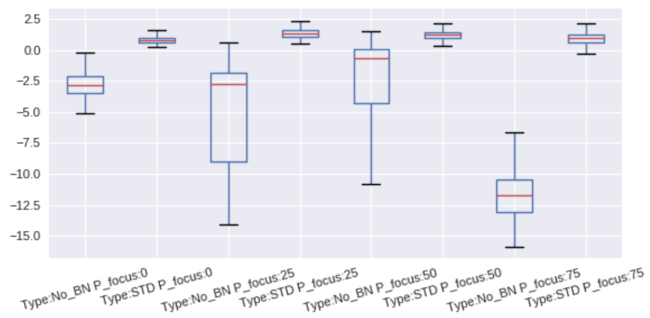

Figure 8. Results against random players

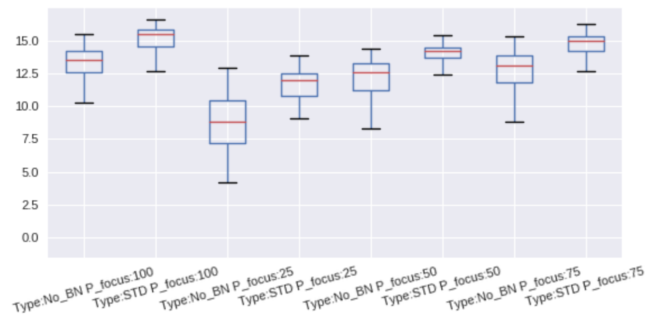

Figure 9. Results against A* (focused) players

type of adversaries, or if they really did learned to identify the strategy of their adversaries and chose a near-optimal sequence of decisions. Therefore we trained identical agents with different distributions ($p \in \{.0, .25, .5, .75, 1.\}$) and tested the agents with each type of player. We compared the results between agents trained on $p \in \{.25, .5, .75, \}$ with agents that had seen only one type of agent during optimization.

As Figures 8 and 9 show, agents that faced both players during training achieve near optimal scores. An expected negative difference is expected since the agent needs to spend some time steps figuring out who is he playing with. An interesting aspect here is that models without batch normalization on the last linear layer perform worse when the distribution changes.

G. Evaluation on Malmo

The last thing in the pipeline was to make sure the performance of our agent transfers from our Pig Chase Replica environment to Malmo Pig Chase. For the competition we performed fine-tuning for six hours with several players collecting experiences and collecting gradients in a similar setup as in [9]. This process brought no significant improvement in the average episodic return, but reduced the variance of the scores stabilizing the policy in the new environment.

Method	Overall	vs A*	vs Random
`standard` + fine tuning	9.527	13.43	0.42

Table I
MODELS EVALUATED ON MALMO PIG CHASE

We report in Table I scores from evaluation on Malmo Pig Chase.

VI. CONCLUSIONS AND FUTURE WORK

In this paper we presented an empirical approach for solving a complex time-extended instance of a social dilemma game. Our solution involved building a replica of the target environment, learning agents on this environment, and then fine tuning them in the original one. Even if there are discrepancies between the two environments, resemblance in the dynamics of the two are enough to transfer high-return policies from the simpler one to the more complex one.

The main objectives for building a replica were: playing simultaneously in a large batch of instances of this game, and doing this very fast by taking advantage of modern GPUs' computing capabilities in tensor algebra.

We tackled the problem with state-of-the-art actor-critic with auxiliary tasks and entropy regularization, but in the end a very careful trained REINFORCE with value prediction as a supplementary cost (but not used in bootstrapping) proved to be enough. Several practical tricks such as clamping the policy, and the gradients significantly improved the training stability and provided a boost in performance.

The success of REINFORCE algorithms depends on large on the baseline used. Our baseline is the mean discounted return for that particular time step in the game.

Although real reinforcement learning problems are usually limited to a single game instance at a time, and playing 1024 trials at once is impossible (e.g. problems in robotics), our method is equivalent to paying a series of games in sequence. The only difference stems from our use of Batch Normalization in the neural predictor, but there are solutions such as building *fake* batches of inputs from the current observations and some old ones (this comes with a cost as prediction for these old observations is not needed, hence computation time is wasted).

There are still further investigations to be done: seeing if transfer learning is possible by evaluating our agents in a new stag hunt problem with different environment dynamics, and trying to train hierarchical models such as [16] in order to abstract the decisions on the payoff matrix of expected episodic returns from the sequence of micro-actions an agent would take to accomplish them.

REFERENCES

[1] Robert M Axelrod. *The evolution of cooperation: revised edition*. Basic books, 2006.

[2] Mohammad Babaeizadeh, Iuri Frosio, Stephen Tyree, Jason Clemons, and Jan Kautz. Reinforcement learning through asynchronous advantage actor-critic on a gpu. 2016.

[3] Enrique Munoz de Cote, Alessandro Lazaric, and Marcello Restelli. Learning to cooperate in multi-agent social dilemmas. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 783–785. ACM, 2006.

[4] Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z Leibo, David Silver, and Koray Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. *arXiv preprint arXiv:1611.05397*, 2016.

[5] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*, 2016.

[6] Guillaume Lample and Devendra Singh Chaplot. Playing fps games with deep reinforcement learning. In *AAAI*, pages 2140–2146, 2017.

[7] Joel Z Leibo, Vinicius Zambaldi, Marc Lanctot, Janusz Marecki, and Thore Graepel. Multi-agent reinforcement learning in sequential social dilemmas. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*, pages 464–473. International Foundation for Autonomous Agents and Multiagent Systems, 2017.

[8] Piotr Mirowski, Razvan Pascanu, Fabio Viola, Hubert Soyer, Andy Ballard, Andrea Banino, Misha Denil, Ross Goroshin, Laurent Sifre, Koray Kavukcuoglu, et al. Learning to navigate in complex environments. *arXiv preprint arXiv:1611.03673*, 2016.

[9] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy P Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *arXiv preprint arXiv:1602.01783*, 2016.

[10] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[11] Anatol Rapoport. Prisoners dilemma: recollections and observations. *Game theory as a theory of conflict resolution. Dordrecht: Reidel*, pages 17–34, 1974.

[12] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

[13] Richard S. Sutton, David A. McAllester, Satinder P. Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *NIPS*, 1999.

[14] Chen Tessler, Shahar Givony, Tom Zahavy, Daniel J Mankowitz, and Shie Mannor. A deep hierarchical approach to lifelong learning in minecraft. In *AAAI*, pages 1553–1561, 2017.

[15] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. *arXiv preprint arXiv:1509.06461*, 2015.

[16] Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. Feudal networks for hierarchical reinforcement learning. *arXiv preprint arXiv:1703.01161*, 2017.

[17] Ziyu Wang, Nando de Freitas, and Marc Lanctot. Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581*, 2015.

[18] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

[19] Michael Wunder, Michael L Littman, and Monica Babes. Classes of multiagent q-learning dynamics with epsilon-greedy exploration. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 1167–1174, 2010.