Knowing one's opponents: Self Modeling Advantage Actor Critic for the Iterated Prisoner's Dilemma

Eric van der Toorn[1] , Neil Yorke-Smith[1] ,
[1]TU Delft
e.a.vandertoorn@student.tudelft.nl
n.yorke-smith@tudelft.nl

Abstract

A recent advancement in Reinforcement Learning is the capability of modelling opponents. In this work, we are interested in going back to basics and testing this capability within the Iterated Prisoner's Dilemma, a simple method for modelling multi agent systems. Using the self modelling advantage actor critic model, we set up a single agent model that encodes its opponents, without requiring the opponents' actions directly. To verify that this technique is indeed capable of modelling opponents its capacity of encoding opponents is tested and the trained model is tested against several popular strategies. The embedding is found to not have a positive effect on the reward, only increasing the randomness of the model.

1 Introduction

The real world is rife with complex social situations which humans encounter and navigate through daily. One of the illustrious goals of Reinforcement Learning (RL) is to have machines that can interact with humans within their social web [1]. Understanding others is a crucial part of being able to interact, and is thus a topic of interest within RL [2].

One major drawback of normal Reinforcement Learning is that it is assumed the entire environment is observed, and unobserved parts of the state are not relevant for decisions. There are many real-world cases where this is not the case however, like traffic signal control[3], or when facing an opponent that takes into account more than just the last action. One technique used to understand such opponents is known as Opponent Modelling, where the agent attempts to derive the strategy of its opponents, thus acknowledging the partial environment and using that to achieve optimal results [4].

Usually, Opponent Modelling requires access to the opponent's observations, but Papoudakis and Albrecht recently developed a new model which does not require this information, using only local observations of the agent itself. They achieve this by using Variational Autoencoders (VAEs), as developed in 2014 by Kingma and Welling [6] to encode opponent information.

In their paper "Variational Autoencoders for Opponent Modelling in Multi-Agent Systems" (VAEOMMAS) [5], Papoudakis and Albrecht presented the Self Modelling Advantage Actor Critic (SMA2C), a model which allows the encoder part of the VAE to learn opponent models conditioned on only local information. Whilst testing the application in some different scenarios, ranging from the Speaker-Listener to Predator-Prey, there was only a brief investigation of the Iterated Prisoner's Dilemma, an extension of the famous Prisoner's Dilemma [7]. In this work, we explore this more by looking into using single-agent RL within the Iterated Prisoner's Dilemma (IPD) where we control a single agent while opponents have a fixed policy.

In doing this our major goal is to test the reproducibility of the VAEOMMAS work, then investigate how to determine when a representation of an opponent has been learned. Additionally, we investigate which information the model performs best with. As an additional expansion, we look whether this model can compete successfully in a round-robin tournament against several different types of opponents, taken from the first tournament hosted by Axelrod [8], to test its capability for generalization within the IPD game.

This paper will start by explaining the concepts that are involved in a background section, then follow with a detailed description of the methodology and model used. Then, we present the results of the experiments held, followed by an analysis and discussion. We conclude with a look at the reliability of this new model and potential future paths to follow.

2 Background

In this section, we will go into the definition of reinforcement learning and what problems it tries to solve, we will go into the policies that are required in multi-agent situations also known as strategies. Then we will explain the environment that we will be doing our experiments on, the iterated prisoner's dilemma, as well as the machine learning technique that will be essential for learning opponent strategies, the variational autoencoder.

2.1 Reinforcement Learning

Reinforcement Learning (RL) is a field of study and problem-solving technique focused on a sequential decision-making problem in which an agent interacts with an environment [9]. For ease of understanding, we will use the game of chess as a typical example hereof throughout the explanation.

2.1.1 Markov Decision Process

The first step before one can use RL is to model the problem as a Markov Decision Process (MDP). An MDP can be defined as a tuple $M = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle$, with $\mathcal{S}$ the set of states of the environment. For chess, this would be all possible configurations of the chessboard. $\mathcal{A}$ is the set of actions, the legal moves that the player can take. $\mathcal{P} : (\mathcal{S}, \mathcal{A}, \mathcal{S}) \rightarrow [0, 1]$ is the transition function or the chance that a certain move will lead to a certain next state. As the moves of the opponent are part of the system in chess, the next state in which the agent can take action is oft uncertain. Lastly, there is a certain numerical reward associated with each action taken at a given state, defined by $\mathcal{R} : (\mathcal{S}, \mathcal{A}) \rightarrow \mathbb{R}$. This could, for instance, be a high reward for the move that checkmates the opponent. One important thing to note about a Markov process is that it assumes the problem possesses the Markov property, meaning that all future states are conditionally dependent only on the current state, not on the past [10].

2.1.2 Learning

Every turn t the agent observes $s_t \in \mathcal{S}$, with $\mathcal{S}$ all possible states that the chess board can be in. Then the agent takes an action $a \in \mathcal{A}(s_t)$, the set of all valid moves it can take. The environment executes the action (plays the move), and the state advances to s_{t+1}, more commonly known as s', and the agent receives a reward r. In our example, if the agent checkmates its opponent with move a, it receives full points. The goal of RL is to find a way to maximize the future reward. It does this by trying to discover the value a state has. For example, the state from which you can checkmate the opponent has a high value to the agent, so when the agent can finish the game from there, this state is given more value.

2.1.3 Policies

A particular strategy is also called a policy π. To find a policy, the agent predicts how much reward it would gain by following each one and chooses the one with the maximal value. The goal is for the agent to learn optimal policy π^*:

$$\pi^* = \text{argmax}_\pi \, R_t \tag{1}$$

With R_t the discounted future reward from time t, also know as the value of a state with respect to a policy:

$$R_t = V^\pi(s_t) = \sum_{k=0}^{T} \gamma^k r_{t+k} \tag{2}$$

The discount γ prioritizes rewards that are closer in the future.

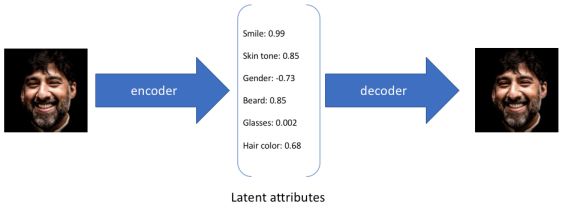


Figure 1: Example of the latent encoding of an autoencoder. Adapted from [13].

While one way of learning a policy is to optimize a value-based function Q in what is known as Q-learning [11], another is to directly parameterize the policy function with a neural network, in what is known as policy gradient [12]. This work will focus on the latter.

To predict this amount of reward for a particular policy, we define a value function which goes through each possible resulting state and assigns it a value, with the more steps a particular state is from the current one, the less value it holds to the agent currently.

2.2 Autoencoder

An autoencoder, as used in this work, is a neural network that has learned to compress its input data. It takes an input encodes that input into a smaller vector and decodes it back to something as similar as possible as its input. The error in the reconstruction can be propagated back to train the network. As an example see fig. 1.

2.3 Variational Autoencoder

One downside of autoencoders is that they do not have smooth transitions. That is, given two similar inputs that encode to similar latent encodings, they could have radically different reconstructions [14]. One way to retain the continuity is to use a variational autoencoder (VAE). This is a recently developed technique [6]. A VAE adds a step in between the encoding and decoding. The 'code', usually named z, is split into two, a μ_z and a σ_z. These variables are then used to sample from a distribution, usually a multivariate Gaussian, as follows:

$$z = \mu_z + \epsilon \sigma_z \tag{3}$$

Through recombining a standardized multivariate Gaussian sample ϵ with the parameters in what is known as the "reparameterization trick", backpropagation towards the encoder is still possible. The sample z is then passed to the decoder and used in the same manner as a normal autoencoder. This way, the autoencoder learns distributions, not discrete values, which forces a more smooth and continuous representation. For an example see fig. 2.

2.4 Iterated Prisoner's Dilemma

The Iterated Prisoner's Dilemma (IPD) an extension of the well-known extension of the general sum game Prisoner's Dilemma (PD). Both have been a rich source of

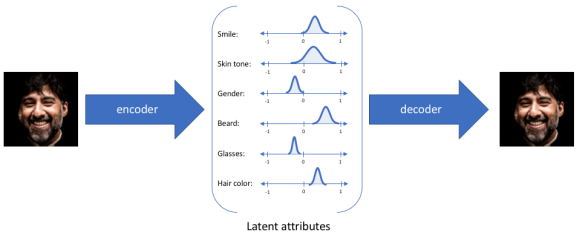


Figure 2: Example of how the learned encoding of a Variational Autoencoder could look like. The curves are representations of the mean and variance that the VAE encodes. Adapted from [13]

research material since the 1950's [7]. Definitions for PD are manifold, for clarity we define it here once again.

Prisoner's Dilemma

		Agent 1	
		Cooperate	Defect
Agent 2	Cooperate	(R,R)	(T,S)
	Defect	(S,T)	(P,P)

Table 1: Payout scheme PD

Two agents play a matrix game: each agent has to choose between Cooperation and Defection, without being able to communicate with or observe the opponent's choice beforehand. Payout is given according to the matrix shown in table 1, with Agents 1 and 2 receiving the first and second part of the tuple respectively, where (in strictly decreasing order of payout):

T is the Temptation payout, for the agent that defected while the other one cooperated. As it is tempting, it should be the largest.

R is the Reward for mutual cooperation.

P is the Penalty for mutual defection.

S is the Sucker payoff, for the agent that cooperated while the other one defected.

Furthermore, the following should also hold: $R > \frac{S+T}{2}$, it should not be a better tactic to interchangeably be the sucker and the tempter rather than consistently cooperating.

When we repeat the Prisoner's Dilemma any number of times, we get the Iterated Prisoner's Dilemma (IPD). The question of IPD is often whether cooperation can evolve with repetition [8].

3 Method

This section contains the methodology used to answer the hypothesis. It starts by defining the reinforcement learning problem that the model will be optimizing for and motivating why this method was chosen in particular. Following that is a description of the environment, an adaptation of IPD. Then we explain the experiments that we have held, from replicating the SMA2C original paper to hosting an Axelrod tournament.

3.1 Problem overview

When tackling a dynamic environment with Reinforcement Learning, we model it as a Markov Decision Process, with an inherent assumption that the problem obeys the Markov property as explained in section 2.1. The situation this work looks at is when the environment contains an opponent whose state is unknown and unobservable to the agent. The task of the agent is then to find out what the opponent is going to do. It encodes the information that it does possess to attempt to model what the opponent is thinking, allowing it to capture that as part of the environment.

3.1.1 Environment and Observations

To model the game as a reinforcement learning task, we describe it as a modified MDP: $M = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle$. The environment in which this investigation takes place is that of the Iterated Prisoner's Dilemma, as explained in section 2.4. This is a matrix game without a state, leaving us with a choice of what our agent observes.

As we wish to investigate whether it is possible to learn what the opponent is thinking without looking into his mind, the state that is observed was decided to be the joint previous action. As the action space $\mathcal{A} = \{C, D\}$, Cooperate (C) or Defect (D), the joint action has 4 possibilities. Additionally, we need a state to represent the start of an episode, so as not to introduce any bias. Thus, $\mathcal{S} = \{CC, CD, DC, DD, d\}$, where d is only active the first state after the game has been reset.

The choice of rewards is arbitrary as long as the rules for IPD are followed. In this work, we chose to stick with the same values as the authors of SMA2C, with $(T, R, P, S) = (3, 2, 0, -1)$, for the respective Temptation, Reward, Punishment, and Sucker payouts. Using these values we define a reward function $\mathcal{R}$ that maps state-actions to rewards. Because the environment is not completely observed, this mapping function and the transition function $\mathcal{P}$ are stochastic, with a given action resulting in one of two states with different rewards, regardless of what the previous state was. For instance, if the action chosen is to Cooperate, the resulting state and reward could be either CC and 2 or CD and -1, depending on what the opponent does.

3.1.2 Multi-Agent learning

This work is focused on the case of single-agent learning with the state of the opponent hidden from the agent. The problem modelled in this work was chosen because it is simpler than the multi-agent setup. However, this system can be naturally extended to a multi-agent environment by allowing other agents to take the place of the hard-coded opponents.

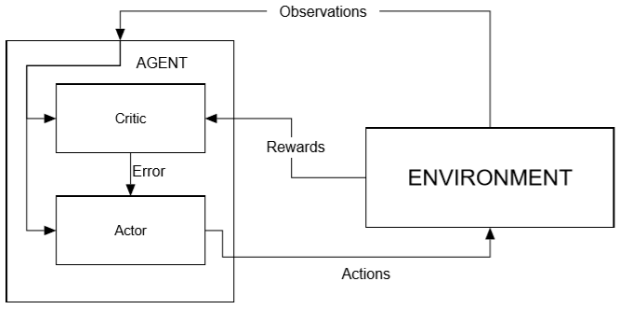


Figure 3: Flowchart of the A2C model, adapted from [15].

3.2 Approach

As a methodology for learning the best policy, we use an adaptation of the Advantage Actor Critic (A2C) gradient optimization method, known as the Self Modeling A2C (SMA2C). What follows is a brief explanation of the original A2C method, after which the changes made by SMA2C are discussed.

3.2.1 Advantage Actor Critic

Advantage actor critic is the synchronous version of Asynchronous Advantage Actor Critic, a variation of the actor critic method [16]. In this method one uses two networks to decide on a policy, the actor and the critic. The actor is responsible for taking actions while the critic perceives the results in the form of rewards and 'criticizes' the actor's actions. This allows the critic network to estimate the value function of the system, while the actor decides the policy. An illustration of the flow of data is shown in fig. 3. In A2C the critic optimizes the advantage using the difference between the return R and the state value $V(s, \theta_v)$, with θ_v the parameters to be optimized, usually with a loss of Mean Squared Error (MSE)

$$A(s, \theta_v) = R - V(s, \theta_v) \tag{4}$$

Then, the value gradient update for the critic becomes

$$\Delta V(\theta_v) = \frac{\delta(A(s, \theta_v)^2)}{\delta \theta_v} \tag{5}$$

And the performance gradient update is:

$$\Delta J(\theta) = \nabla_\theta \ln \pi(a_t|s_t, \theta)(A(s, \theta_v) \tag{6}$$

3.2.2 Self Modeling Advantage Actor Critic

To allow the agent to model its opponent an additional network is added which acts as a form of preprocessor on some additional factors in the input. The changes can be seen in fig. 4. This network is the encoder part of a variational autoencoder (see section 2.3 for details), and is henceforth referred to as the encoder. The encoder receives as its input the following: The state of the environment s_t, the previous action a_{t-1}, and the reward just obtained r_t. The encoder internally uses a recurrent layer to learn historical behaviour, requiring the addition

Algorithm 1: Training SMA2C

```
Require: Set of opponents 𝕋, episode length l
Require: encoder L : (𝒮, 𝒜, ℛ, d) → z⃗
Require: actor network P : (𝒮, z⃗) → 𝒜
Require: critic network C : (𝒮, z⃗) → ℝ
Require: environment E_𝕋 : 𝕋 → (𝒮, ℛ)
Pick opponent A_{-1} ∈ 𝕋
o_0, r_0 ← E_{A_{-1}}                    // reset the environment
a_0 ← 0                                  // initial action
o_1, r_0 ← E(a_0)
O ← array of size l
for t = 1 to l do
    e_t ← L(o_t, a_{t-1}, r_{t-1}, d_{t-1})   // encode
    a_t ← P(e_t, o_t)                    // decide action
    o_{t+1}, r_t ← E(a_t)                // perform action
    store (o_t, a_t, r_t) in O
end
Replay(O)                                // Gradient ascent
```

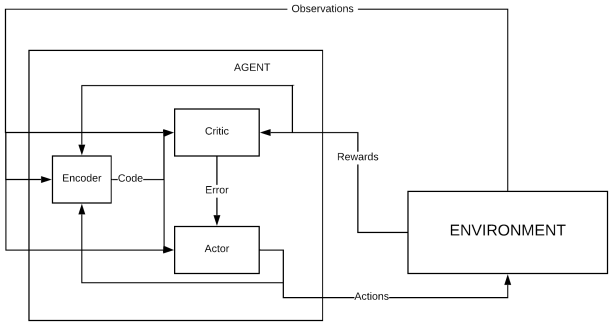


Figure 4: Flow diagram of the SMA2C network

of a reset parameter d, only true if the environment has been reset. The final encoding z is generated using the reparameterization trick, explained in section 2.3.

3.3 Training

Running the algorithm is done as described in algorithm 1. The major difference, when compared to standard A2C, is that the encoding is first generated from the inputs and added to the input that the standard A2C network receives.

3.3.1 Updating the gradients

In SMA2C we back-propagate the loss from the A2C network into the encoder network, allowing it to optimize for that loss as well. As such, we do not need to add a custom loss on top of the typical A2C loss [5].

3.4 Experiments

This section contains an overview and an in-depth explanation of the experiments done for this work. Starting with the replication experiment and followed by the expansions on that model, a third experiment holds the

created model and tests its performance against that of several other strategies.

3.4.1 Opponent Strategies

When testing a model's learning to understand opponents, it is important to control what the model faces, both during training and testing. For these experiments, we will be testing the model against the following strategies, as defined in the Axelrod package [17]. While far from the only ones (the package itself contains more than 100), these are easy to understand, and the Tit for Tat strategy has been shown to perform outstandingly regardless of its simplicity [8].

- The Cooperator, always cooperates, regardless of what its opponent does.
- The Defector, always defects, regardless of what its opponent does.
- The Tit for Tat strategy starts by cooperating with its opponent and then does whatever its opponent did last.

3.4.2 Replication

This work is largely focused on the work by Papoudakis and Albrecht [5], and as such it is our primary goal to test whether we can replicate the results they achieved. In section 5.1 of their work, they set up their experiment as an agent facing two possible policies: The defector, who always defects, and the Tit-For-Tat, a policy that starts by cooperating and then does whatever its opponent did previously. For our replication, we use the same payout matrix and the same number of time steps per episode (25). We compare our results by comparing our mean score graph to the one displayed in their work.

3.5 Expansions

To expand on the model described in 3.4.2, we experiment with changes to the model as well as the environment. An ablation study is done by removing inputs from the encoder model. We remove the reward and observation from the inputs to the encoder and test its impact on performance. Then, we test other strategies for the agent to learn, firstly a simple cooperator stratagem, and then the original players of the Axelrod tournament [17].

3.6 Evaluation

Testing the accuracy of the model is not as simple as taking the reward and seeing if it increases, as the randomly picked opponents have a very different maximal score that can be attained. With a timestep of 25, against the Defector, which always defects, the maximal score is -1, obtained by always defecting after the first turn. Contrarily, against Tit For Tat, the maximal score is 50, obtained by always cooperating. To simplify this, we allow the optimal policy to be constant, at the expected maximal value of 24.5.

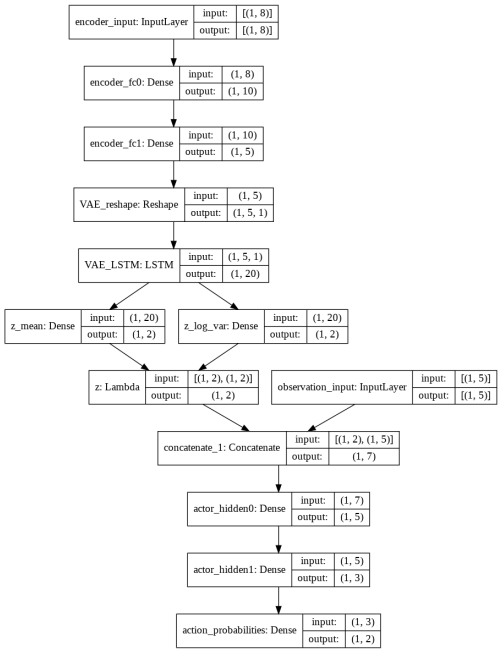


Figure 5: Model of the actor network, critic only uses a different output layer

4 Results

This section contains the details for running the experiments, the results obtained through running the experiments, as well as a brief analysis of these results.

4.1 Training details

For the sake of reproducibility, it is important to clarify the many variables that have been set and used, especially the hyperparameters.

4.1.1 Environment details

For matrix games like the iterated prisoner's dilemma, the state can be quite ambiguous. In this work, we chose to treat the joint action as the state, which was passed to the model as a one-hot encoding. The rewards for the actions were kept to $3, 2, 0,$ and -1 for Temptation, Reward, Penalty, and Sucker respectively, and scaled to between $[-1, 1]$ using a MinMaxScaler [18], to effectively standardize the scores while allowing for reversal. Standardization is applied to speed up gradient descent and prevent floating-point errors [19].

4.1.2 Network details

As displayed in fig. 5, the input of the model is transformed into the encoding and that is then combined with the original input as the input to the A2C network. For the pure A2C network, the embedding is not added to the input of the A2C network and is thus skipped. In terms of variables, we set the discount factor γ to be 0.999, a fairly typical value, and set the learning rates of the actor and critic to be 0.0015 and 0.005, or half of that, respectively. Exact configurations for each run

used within this work and many more can be found in the run files in appendix A.1.

4.2 Replication

For the replication of the VAEOMMAS study [5], we use the same episode length and 4000 episodes, or 100,000 timesteps. The results from the original study are shown in fig. 6. Using several different configurations[1], and running them at least 10 times with a random seed, we visualized the results in fig. 7. We see that both A2C and SMA2C are learning and result in better policies than a purely random policy. On average, the A2C policy performs better, although it has a wider spread of its eventual policy rewards, as can be seen in fig. 8. To find out why the score of the SMA2C agent is lower, we also visualize the scores for a single training episode in fig. 9 and see that the A2C network is more concentrated, indicating that the SMA2C network has more randomness.

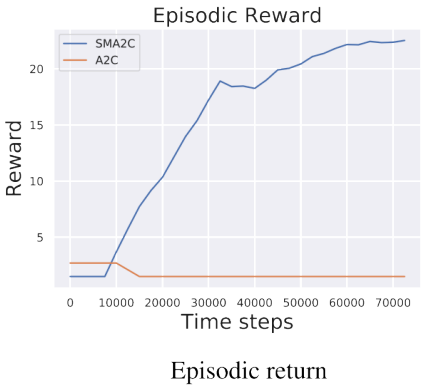


Figure 6: Original results from [5]. Optimal average return is 24.5

4.2.1 Embeddings

The embeddings created in the original paper, as seen in fig. 10 is vastly different from the one our trained model creates, shown in fig. 11. The trained embeddings from our replication, found by running the trained model 100 times against both the Defector and Tit For Tat policies, converge around the same area for all three tested agents, contrary to how they have no overlap in fig. 10. This seems to indicate that the input is not used. Adding to this, a pure A2C agent shows the same or better performance, as can be seen in fig. 7

4.3 Expansions

4.3.1 Cooperator

To test whether a trained agent is capable of facing different opponents, we first see how its embeddings change when facing an unseen strategy. On fig. 10, one can see

[1]Run configurations and results can be found in appendix A

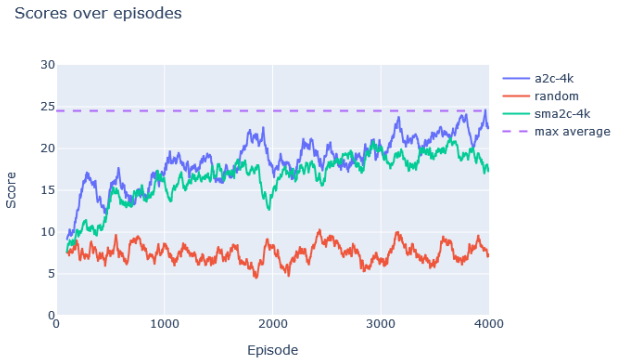
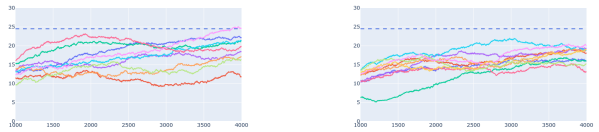
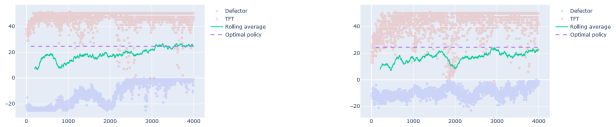


Figure 7: Results from this replication, rolling average of mean scores with window size of 500. Every line represents 10 experiments. Dashed line represents optimum policy.



(a) A2C. (b) SMA2C.

Figure 8: Individual rolling averages from experiments training for both A2C and SMA2C. Same axes for easy comparison.



(a) A2C. (b) SMA2C.

Figure 9: Single experiment scores, with rolling averages (window size 200). Dashed line represents optimal policy.

that the average encoding of the cooperator falls in the same area as the TFT and Defector strategies, so the embedding doesn't seem to be changing depending on the opponent policy. When a trained agent faces the cooperator, they always cooperate, which is the suboptimal strategy. This is likely because it has learned to cooperate when the opponent cooperates from the Tit For Tat policy.

4.3.2 Performance against different strategies

To test the results of our algorithm on these different strategies, we created a wrapper around the best per-

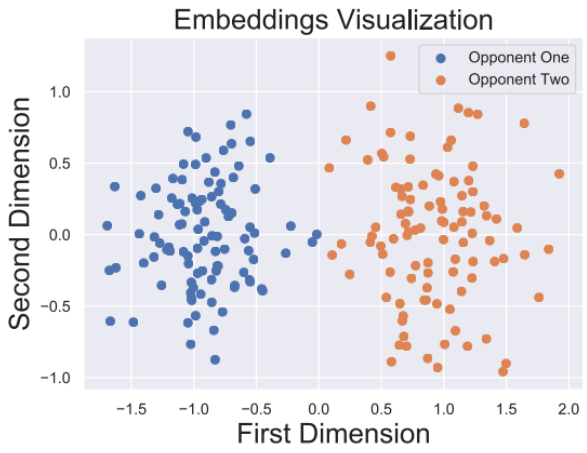


Figure 10: Original embeddings from [5].

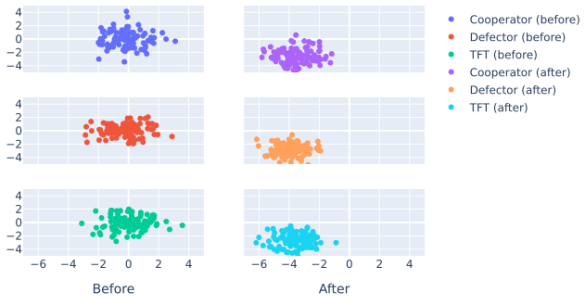


Figure 11: Embeddings obtained in this work. Single run, x and y axes are the respective dimensions of the embedding. Left column shows embeddings before training, right column after training. From top to bottom the rows indicate the Cooperator, Defector, and Tit For Tat strategy.

forming model[2] to function as a strategy within the Axelrod package [17]. Then, we hosted a tournament with the same strategies as Axelrod's first tournament and added our model to the pool. The results of one of these tournaments are visualized in fig. 12, showing that the model had comparatively small variance in its mean reward throughout the tournament. Our agent consistently got a mean of at least 1.6 points per turn, allowing it to win 5 out of 10 tournaments and remain in the top half the rest of the tournaments. It was capable of consistently cooperating while also retaliating when 'attacked' with Defection.

4.3.3 Ablation study on SMA2C inputs

Using several subsets of the inputs, we tested whether the model required all its diverse inputs. Using just the joint action as input would result in purely random decisions without improvement, or quick convergence to either constant defection or cooperation, which were undesired results.

[2]Run identifier: 1592552683-sma2c-noseed-10000-smaller-mid

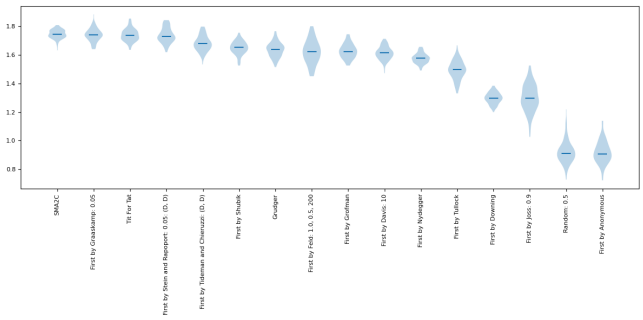


Figure 12: Box plot of mean score per turn, using same payoff matrix. Tournament consisted of 100 repetitions of a round robin, each game was 25 turns.

5 Conclusion

To conclude this work, a recently developed technique for training a Reinforcement Learning agent, the SMA2C model, was replicated and expanded using the Iterated Prisoner's Dilemma. It was shown that using variational autoencoders to create embeddings for the A2C model does not improve results for simple IPD strategies over the basic A2C model, because the A2C model can perform better even without the embedding being given.

This is likely because the embeddings are not optimized to represent the opponent model correctly, as they are shown to not have significant differences when facing different policies. Instead of helping the A2C network decide, the embedding then only adds a random factor, which reduces its consistency. The goal of this work was to test the new network under several circumstances and see how it would hold itself together. While it does not seem to be the best model for the Iterated Prisoner's Dilemma, this could also be due to the lack of complexity and state involved in the network.

5.1 Discussion

The results of this work were unfortunately not as conclusive as we would have hoped. Most results for SMA2C showed a positive increase, but with different learning rates and layer sizes, we also got no convergence or convergence on defection only. In machine learning and especially RL, one must never be surprised about failure given the number of hyperparameters that need to be adjusted and the stochastic nature of learning, especially when there is so little time to tune. An effort was initially made to allow this to be done using the Distributed RL library RLlib [20], which failed due to the complexities involved in adding the VAE and its stateful components.

5.1.1 Why not a Multi-Armed Bandit problem?

One decision that had to be made was whether to model this as a multi-armed bandit problem (MAB), a simpler

case of reinforcement learning where there is no state for the environment [21]. In other words, the MAB assumes that the next action is not dependent in any way on the previous action. Within IPD all but the most basic of tactics use the history of the opponent's actions or their actions to determine what to do. This makes it more than just a MAB as there is not necessarily an observable state but there is a state that opponent has which you do not have access to.

5.1.2 Future work

The most exciting future study for this work would be pitting it against other learning agents and thus introducing true non-stationarity [3]. Additionally, it would be very interesting to see the result of running the SMA2C on Sequential Social Dilemmas to test whether it can understand the opponent in more complicated situations [22].

One metric that would be of great use for continuing this study is the Mutual Information Neural Estimation (MINE) metric, which estimates how accurately an embedding holds information [23]. Using this metric to estimate whether the embeddings are useful could strengthen our hypothesis that they are currently not, and help us find a way to train them to be more so.

In closing, we would like to thank the author of the VAEOMMAS paper, Georgios Papoudakis, for helping us set up the model and granting aid when requested.

6 Responsible Computer Science

Machine learning and reinforcement learning are a field of study that is central to many reproducibility issues [24]. Because of a continuous drive for innovation and many competing research institutions, few papers focus on reproducing results and verifying them and there are also lacking descriptions of mythology for methodologies and the lack of data in many papers [25]. This paper centres around a reproduction of the paper by Papoudakis and Albrecht while adding new experiments and tests for one of the examples that they used. We have shown that instead of much better performance than A2C, SMA2C could perform even worse. Contrary results are not always desired, but they allow us to realise flaws in our theories, which open up the pathway to improved scientific study. To aid in that effort, code for these experiments is made open-source and written in a way that is hopefully reproducible, along with all the data used for graphing our results. The methodology is explained in detail such that readers can recreate the experiments if so desired.

In terms of ethical issues, this work in itself does not pose much risk, centring around the reproduction and extension of another paper as it did. However, the general research topic of being able to recognize what your 'opponent' is thinking, of course, could have many implications when taken to its limit. Knowing how someone will react to our actions could prevent many dramatic miscommunications with disastrous consequences.

Research into the prisoner's dilemma has led to some interesting insights into human trust as well as perception. This game has been put to the test with real humans which did not create as much cooperation as was expected and further research on this topic could lead us to understand in which circumstances human cooperation is stimulated.

References

[1] Applying AI for social good | McKinsey. URL: https : / / www . mckinsey . com / featured - insights/artificial- intelligence/applying- artificial- intelligence-for-social-good.

[2] Finnegan Southey et al. "Bayes' Bluff: Opponent Modelling in Poker". In: Proceedings of the 21st Conference on Uncertainty in Artificial Intelligence, UAI 2005 (July 2012), pp. 550–558. URL: http://arxiv.org/abs/1207.1411.

[3] Sindhu Padakandla, Prabuchandran K. J, and Shalabh Bhatnagar. "Reinforcement Learning in Non-Stationary Environments". In: (May 2019). DOI: 10.1007/s10489-020-01758-5. URL: http://arxiv.org/abs/1905.03970%20http://dx.doi.org/10.1007/s10489-020-01758-5.

[4] Stefano V. Albrecht and Peter Stone. "Autonomous Agents Modelling Other Agents: A Comprehensive Survey and Open Problems". In: Artificial Intelligence 258 (Sept. 2017), pp. 66–95. DOI: 10.1016/j.artint.2018.01.002. URL: http://arxiv.org/abs/1709.08071%20http://dx.doi.org/10.1016/j.artint.2018.01.002.

[5] Georgios Papoudakis and Stefano V. Albrecht. "Variational Autoencoders for Opponent Modeling in Multi-Agent Systems". In: arXiv e-prints (Jan. 2020). URL: http://arxiv.org/abs/2001.10829.

[6] Diederik P. Kingma and Max Welling. "Auto-encoding variational bayes". In: 2nd International Conference on Learning Representations, ICLR 2014 - Conference Track Proceedings. International Conference on Learning Representations, ICLR, Dec. 2014.

[7] Graham Kendall, Xin Yao, and Siang Yew Chong. The Iterated Prisoners' Dilemma. Vol. Volume 4. World Scientific, May 2007, p. 272. ISBN: 978-981-270-697-3. DOI: doi:10.1142/6461. URL: https://doi.org/10.1142/6461.

[8] R Axelrod and W. Hamilton. "The evolution of cooperation". In: Science (1981). ISSN: 0036-8075. DOI: 10.1126/science.7466396.

[9] Afshin OroojlooyJadid and Davood Hajinezhad. "A Review of Cooperative Multi-Agent Deep Reinforcement Learning". In: arXiv e-prints (Aug. 2019). URL: http://arxiv.org/abs/1908.03963.

[10] A Markov. Theory of algorithms. Moscow: Academy of Sciences of the USSR, 1954.

[11] Christopher J. C. H. Watkins and Peter Dayan. "Q-learning". In: Machine Learning 8.3-4 (May 1992), pp. 279–292. ISSN: 0885-6125. DOI: 10.1007/bf00992698.

[12] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: Nature 518.7540 (Feb. 2015), pp. 529–533. ISSN: 14764687. DOI: 10.1038/nature14236.

[13] Jeremy Jordan. Variational autoencoders. 2018. URL: https://www.jeremyjordan.me/variational-autoencoders/.

[14] Jesse Engel, Matthew Hoffman, and Adam Roberts. "Latent Constraints: Learning to Generate Conditionally from Unconditional Generative Models". In: 6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings (Nov. 2017). URL: http://arxiv.org/abs/1711.05772.

[15] Asynchronous actor – critic training - Learn Unity ML-Agents - Fundamentals of Unity Machine Learning. URL: https://subscription.packtpub.com/book/game_development/9781789138139/4/ch04lvl1sec34/asynchronous-actor-critic-training.

[16] Volodymyr Mnih et al. "Asynchronous Methods for Deep Reinforcement Learning". In: 33rd International Conference on Machine Learning, ICML 2016 4 (Feb. 2016), pp. 2850–2869. URL: http://arxiv.org/abs/1602.01783.

[17] Vince Knight et al. Axelrod-Python/Axelrod: v4.9.1. Apr. 2020. DOI: 10.5281/ZENODO.3744465.

[18] F Pedregosa et al. "Scikit-learn: Machine Learning in {P}ython". In: Journal of Machine Learning Research 12 (2011), pp. 2825–2830.

[19] R Reed and RJ MarksII. Neural smithing: supervised learning in feedforward artificial neural networks. 1999. URL: https://books.google.com/books?hl=en&lr=&id=7C4TDgAAQBAJ&oi=fnd&pg=PR5&dq=Neural+Smithing:+Supervised+Learning+in+Feedforward+Artificial+Neural+Networks&ots=8i2gYWZ-gp&sig=9QOdyAEkUmcxraIwO77y1_SyFKg.

[20] Eric Liang et al. "RLlib: Abstractions for Distributed Reinforcement Learning". In: 35th International Conference on Machine Learning, ICML 2018 7 (Dec. 2017), pp. 4768–4780. URL: http://arxiv.org/abs/1712.09381.

[21] Michael N. Katehakis and Arthur F. Veinott. "MULTI-ARMED BANDIT PROBLEM: DECOMPOSITION AND COMPUTATION." In: Mathematics of Operations Research 12.2 (May 1987), pp. 262–268. ISSN: 0364765X. DOI: 10.1287/moor.12.2.262.

[22] Joel Z. Leibo et al. "Multi-agent Reinforcement Learning in Sequential Social Dilemmas". In: Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS 1 (Feb. 2017), pp. 464–473. URL: http://arxiv.org/abs/1702.03037.

[23] Mohamed Ishmael Belghazi et al. "MINE: Mutual Information Neural Estimation". In: (Jan. 2018), pp. 1–44. URL: http://arxiv.org/abs/1801.04062.

[24] Matthew Hutson. Artificial intelligence faces reproducibility crisis. Feb. 2018. DOI: 10.1126/science.359.6377.725.

[25] Adriano Rivolli et al. "Characterizing classification datasets: a study of meta-features for meta-learning". In: arXiv e-prints (2018). URL: http://arxiv.org/abs/1808.10406.

A Code

Code for this project can be found at https://github.com/Pluriscient/sma2c-ipd

A.1 Runfiles

Runfiles for each respective run can be found on the github repository under the outputs folder.