A Generalized Framework for Self-Play Training in Reinforcement Learning

Daniel Hernandez^1**, Peter York**^{1}**, Yuan Gao**^{2}**, Sam Devlin**^{3}**, Spyros Samothrakis**^{4}**, James Walker**^{1}
^{1} (IGGI?) Department of Computer Science, University of York, United Kingdom
^{2} Department of Computational Science and Technology, Uppsala University, Sweeden
^{3} Microsoft Research, Cambridge
^{4} Department of Computer Science, Essex University, United Kingdom

Abstract

- Self-play has been used for ages without a rigurous definition.
- We present a formalized framework which encapsulates the mearning of self-play, and explain how it changes the MARL loop
- Examples drawn from the literature are presented under the proposed framework
- We explore a novel self-play idea to show the breadth of applicability of the framework.
- (Sentence about future experiments results)

1 Introduction

- Cover overfitting and catastrophic forgetting (unlearning) in multiagent systems.

Classical approaches to training a good performing policy on multi-agent environment suffer from the problem of overfitting. This is occurs when learning agents become good at operating with or against themselves, but considerably drop performance when matched against other agents that act differently to those they have previously encountered. This issue is exacerbated when one knows not of any existing good opponent strategies to test against, which is also the main reason why reinforcement learning is the only viable machine learning paradigm for these cases.

Self-play is a training scheme which arises in the context of multi-agent training. Training using self-play means that learning agents learn *purely* by simulating playing with themselves. The learning algorithm does not have access to any pre-existing dataset during the course of training. Using expert knowledge datasets [Silver *et al.*, 2016; **?**] to bias learning towards expert human play, or creating datasets from existing environment footage [Aytar *et al.*, 2018] is not an uncommon paradigm in reinforcement learning. However, a training scheme that relies on these cannot be considered self-play. This does not mean that the learning algorithm needs to start learning *tabula rasa*, as this constraint only takes effect during training, allowing for prior knowledge to be embedded into the agent before training begins.

Describe the notion of self-play as a training scheme. It merely decides which policies will define the behaviour of other agents in the environment (Lanctot here?). By choosing a set of fixed strategies to play against, we are esentially fixing a Markov Game for the agent under policy π to act on. The self-play module presents the learning agent π with a sequence of Markov Games throughout the duration of the training.

The rest of the paper is structured as follows. Section 2 briefly introduces Markov Games as an environment model and the difference between value based and policy based algorithms. Section 3 summarises the notion of self-play in the literature. In Section 4 we set up our self-play framework module, showcasing it's applicability to existing self-play usages and different environment models. Sections 6 present the set up behind the experiments we carried out to measure the significance of the self-play module. In Section 7 these results are presented. Finally, Section 8 concludes the paper.

1.1 Notation

A word on notation. Bold vectors $\boldsymbol{\pi}$ (vector of policies). Upper case for matrices A.

(If we talk about using KL divergence as a gating mechanism, discuss the benefit of computing KL divergence for a whole trajectory using matrix equations instead of aggregating over individual action distributions from the trajectory)

2 Preliminaries / Motivation

- Introduction to Markov games (or whatever model we are using)
- Value based methods. Talk about DQN.
- Policy gradient methods. Talk about PPO. (because both methods will be used)

3 Related Work

The notion of self-play has been present in the reinforcement learning world for over half a century. [Samuel, 1959] discusses the notion of learning a state value function to evaluate board positions, to inform a 1-ply tree search algorithm, by playing against itself in the game of checkers. The TD-Gammon algorithm [Tesauro, 1995] used self-play to learn a policy using TD(λ) [Sutton and Barto, 1998] to reach expert level backgammon play. This approach surpased previous work by the same authors which relied on learning from a dataset of expert human knowledge (cite?). More recently, AlphaGo [Silver *et al.*, 2017b] used a version of expert iteration [Anthony *et al.*, 2017], which is built on top of self-play. This approach was capable of beating the world champion of go, and it generalized to the games of shogi and chess [Silver *et al.*, 2017a].

It is often assumed that a training scheme can be defined as self-play if, and only if, the learning agent plays against the latest version of itself. Meaning that, when the agent's policy is updated, all the other agents in the environment mirror this policy update. [Bansal *et al.*, 2017] relaxes this assumption. Instead of keeping a single policy for both agents, they create a dataset of "historical" policies as training progresses. At the beginning of an episode, one of the agent's behaviour is defined by a policy sampled from this dataset of historical policies. By checkpointing[1] the latest policy every fixed interval during training. They allow some of the agents to sample uniformly from this policy dataset every fixed amount of episodes. Thus, the latest policy is presented with a wider variety of behaviours, increasing its robustness (i.e, successful against a variety of opponents without hyperparameter tuning). This is an attempt to solve the problem of catastrophic forgetting[2].

Similar ideas have also been independently discovered in other fields. In psychology [Treutwein, 1995] introduces the *adaptive staircase procedure*, where a learning agent is presented with a set of increasingly difficult sets of trials. Upon succeding enough trials inside a difficullty level, the agent is presented with harder trials, and if it fails continually, it is demoted to an easier set. The link with [Bansal *et al.*, 2017] and self-play is that sometimes the agent is presented with trials beloinging to easier levels of difficulty. Such procedure ensures that the agent does not forget how to solve trials outside its current level of difficulty. This procedure was proved to work for the deep reinforcement learning architecture UNREAL [Beattie *et al.*, 2016] in virtual visual acuity tests [Leibo *et al.*, 2018].

Self-play can be viewed as a natural curriculum learning problem, in which lessons are generated as training occurs. A dataset of historical policies which grows over the course of training can be regarded as a curriculum to which lessons are dynamically added. State of the art algorithms sample uniformly from this dataset of historical policies, but it could be interesting to experiment with other opponent sampling distributions. There is no research exploring which opponent sampling distribution works best for different types of environments. It may even be possible to *learn* this opponent sampling distribution during training using meta reinforcement learning. [Duan *et al.*, 2016] use meta reinforcement learning to learn a "learning algorithm" to perform well on a set of MDPs which is sampled from a given distribution over MDPs. Let's assume a dataset of policies that has been generated during training. Let's also assume that each of this policies is fixed. In a 2 agent environment, sampling a new opponent using some distribution over the set of avaialable policies is analogous to sampling a new MDP. Therefore it could be possible to translate the ideas of [Duan *et al.*, 2016] into self-play, with the addition that the set of available policies (MDPs) would grow over time.

Self-play has been empirically verified as a valid training scheme, but the theoretical proofs are lagging behind, as it is the case with many corners of the field of machine learning [Henderson *et al.*, 2017; **?**]. An underlying issue of self-play studies is the fact that multi-agent environments are non-stationary and non-Markovian [Laurent *et al.*, 2011], resulting in a loss of convergence guarantess for many algorithms. Beyond this, there are many open questions that arise regarding the nature of self-play. [Tesauro, 1992] observes that it is not clear a priori that such a learning system would converge to a sensible solution, and no convergence proofs exist to this date.

Although the lack of theoretical guarantess does not mean lack of experimental results. [Van Der Ree and Wiering, 2013] experimented with two different training strategies one being learning by classical self-play (only using the latest policy for all agents) and the other training against a fixed opponent. They have empirical evidence that shows that for some algorithms learning by self-play yields a higher quality policy than learning against a fixed opponent, but that this is algorithm dependant. Concretely, TD-learning [Sutton and Barto, 1998] learnt best from self play, but Q-learning performed better when learning against a fixed opponent. Similarly, [Firoiu *et al.*, 2017] found that DQN[Mnih *et al.*, 2013], a Q-learning based algorithm, did not perform well when trained against other policies which were themselves being updated simultaneously, but otherwise performed well when training against fixed opponents. However, the environments used for their experiments differ too much to draw parallel conclusions from their results.

4 Self-Play Framework

4.1 Self-play as an extension of the multi-agent reinforcement learning loop

The multi-agent reinforcement learning loop can be stated as follows. Let $\boldsymbol{\pi}$ denote the policy vector, containing the policies of every agent in a multi-agent environment. The environment presents all agents with state s_t. The vector containing the actions for all agents, $\boldsymbol{a_t}$, is sampled from the policy vector $\boldsymbol{a_t} \sim \boldsymbol{\pi}(s_t)$. The environment then executes the action vector $\boldsymbol{a_t}$, yielding a new environment state s_{t+1} and a reward vector $\boldsymbol{r_t}$ containing a reward for each agent. This loop ends when a terminal state is reached, signaling the beginning of a new iteration.

Self-play can be conceived as a module that extends this loop by introducing two new steps. One before the beginning of an episode and another on episode termination. A self-play module envelops the RL loop by first deciding which policies $\boldsymbol{\pi}$ will define the agent's behaviour before an episode begins. On episode termination, it decides whether or not the

[1]If a neural network is used to represent the policy, creating a checkpoint means storing the weight values of the neural network at a given timestep.

[2]In a multi-agent reinforcement learning context, catastrophic forgetting refers to the event of a policy dropping in performance against policies for which it used to perform favourably.

(possibly updated) policy π will be introduced in the pool of available policies $\boldsymbol{\pi}^{\boldsymbol{o}}_{\boldsymbol{t+1}}$. This intuition is formally captured in Algorithm 1. The functionality of the self-play module has bees highlighted in red, the black text represents the vanilla multi-agent RL loop.

A self-play module is fully specified by the tuple $< \Omega(\cdot|\cdot), G(\cdot,\cdot) >$:

- $\Omega(\boldsymbol{\pi'}|\boldsymbol{\pi^o}) \in [0,1]$ where $\boldsymbol{\pi'} \subseteq \boldsymbol{\pi^o}$ is the opponent sampling distribution. A probability distribution over the set of available opponent policies $\boldsymbol{\pi^o}$, the menagerie. Given the current policy π being learnt and the menagerie, it chooses an opponent policy to be used.
- $\boldsymbol{\pi}^{\boldsymbol{o}}_{\boldsymbol{t+1}} = G(\boldsymbol{\pi^o})$ is the gating function. Where π is the latest version of the current policy being learnt and $\boldsymbol{\pi}^{\boldsymbol{o}}_{\boldsymbol{t+1}}$ is the menagerie at the begining of episode $t+1$. It is a function (distribution) conditioned on a menagerie which determines whether to add policy π into the menagerie, in which case $\boldsymbol{\pi}^{\boldsymbol{o}}_{\boldsymbol{t+1}} = \{\pi\} \cup \boldsymbol{\pi}^{\boldsymbol{o}}_{\boldsymbol{t}}$, or the policy is not added to the menagerie, thus $\boldsymbol{\pi}^{\boldsymbol{o}}_{\boldsymbol{t+1}} = \boldsymbol{\pi}^{\boldsymbol{o}}_{\boldsymbol{t}}$. This function could be deterministic of stochastic, it is not clear if having a stochastic gating function would yield benefits.

(The gating function saves every episode. It could also potentially be used everytime that the policy receives an update, or after X episodes. The opponent sampling distribution samples opponents on every episode, should we have an opponent policy change interval?)

Algorithm 1: Multi-Agent RL Loop with Self-Play.

```
Input: Environment: $(\mathcal{S}, \mathcal{A}, \mathcal{P}(\cdot|\cdot,\cdot), \mathcal{R}(\cdot,\cdot), \gamma, \rho_0(\cdot))$
Input: (random) Single training policy: $\pi(\cdot)$
1  $e \leftarrow 0$;
2  $\boldsymbol{\pi}^{\boldsymbol{o}}_{\boldsymbol{0}} = \{\pi\}$
3  for $e = 0, 1, 2, \ldots$ do
4      $\boldsymbol{\pi'} \sim \Omega(\boldsymbol{\pi}^{\boldsymbol{o}}_{\boldsymbol{e}})$;
5      $\boldsymbol{\pi_e} = \boldsymbol{\pi'} \cup \{\pi\}$;
6      $s_0 \sim \rho_0$;
7      $t \leftarrow 0$;
8      $\tau \leftarrow \{\}$;
9      repeat
10         $\boldsymbol{a_t} \sim \boldsymbol{\pi_e}(s_t)$;
11         $s_{t+1} \sim P(s_t, \boldsymbol{a_t})$;
12         $\boldsymbol{r_t} \sim \boldsymbol{R}(s_t, \boldsymbol{a_t})$;
13         $t \leftarrow t + 1$;
14         $\tau \leftarrow \tau \cup \{s_t, \boldsymbol{a_t}, \boldsymbol{r_t}, s_{t+1}\}$;
15     until Episode termination;
16     $\boldsymbol{\pi}^{\boldsymbol{o}}_{\boldsymbol{e+1}} = G(\boldsymbol{\pi}^{\boldsymbol{o}}_{\boldsymbol{e}}, \tau)$;
17     $e \leftarrow e + 1$;
18 end
```

4.2 Generalizability to different environment models

The self-play RL loop presented in Algorithm 1 assummes that the environment is modeled using a (Markov Game?). However, due to the fact that the self-play module only interacts directly with the agent policies, it remains agnostic on the underlying environment model being used.

The self-play module described at the beginning of this section makes three assumptions:

- The environment model features multiple agents.
- During training, the self-play module has access to the policy representations[3].
- These policies can be modified during episodes. More concretely, the ability to swap them for other stored policies.

For completeness, the Appendix.

4.3 Examples from literature

As discussed in Section 1, the literature already uses varios self-play module. This framework generalises to existing self-play module. To illustrate this point, we present two self-play module, a *naive*-self-play module and a δ-uniform-self-play module introduced by (Bansal 2017).

Naive Self-Play

A naive-self-play training scheme always uses the same version of the latest policy to populate the behaviour of all present agents. To capture this, the opponent sampling distribution Ω put all probability weight to the latest policy being learnt, denoted by π_{latest}.

$$\Omega(\pi|\boldsymbol{\pi}^{\boldsymbol{o}}_{\boldsymbol{t}}) = \begin{cases} 1 & \pi == \pi_{latest} \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

In such degenerate scenario the choice of gating function G becomes irrelevant, as any policies added to the menagerie will be ignored. However, it is also possible to capture the full essence of naive-self-play by choosing a gating function G that never adds a candidate policy to the menagerie. In such scenario the size of the menagerie remains always constant: $\forall t \in \mathbb{R}\ |\boldsymbol{\pi^o}| = 1$. A vector of policies of size one, containing only the policy being learnt. Formally:

$$\forall\pi\forall\boldsymbol{\pi}^{\boldsymbol{o}}_{\boldsymbol{t}}\forall t \in \mathbb{R}\ \boldsymbol{\pi}^{\boldsymbol{o}}_{\boldsymbol{t+1}} = G(\pi|\boldsymbol{\pi}^{\boldsymbol{o}}_{\boldsymbol{t}}) \tag{2}$$

It requires no extra computation to be used for both Ω and G.

δ-Uniform Self-Play

Let v denote the size of the menagerie $v = |\pi^o|$, and let $\delta \in [0,1]$ denote the threshold on the oldest policy that will be considered as a sample. A $\delta = 0$ Thus, $\delta = 1$ corresponds to the latest available opponent and $\delta = 0$ corresponds to uniform sampling over the entire history.

$$\Omega(\pi|\pi^o) = Uniform(\delta v, v) \tag{3}$$

The gating function G used in δ-uniform-self-play is fully inclusive, after every episode it always includes the latest policy π_{latest} into the menagerie π^o. This is captured by the equation:

[3]If the policies are being represented by a neural network. Access to the policy representation means access to the neural network topology and weights.

$$\forall \pi \forall \boldsymbol{\pi^o_t} \forall \boldsymbol{\pi^o_{t+1}} = \{\pi^0\} \cup \boldsymbol{\pi^o_t} = G(\boldsymbol{\pi^o_t}) \tag{4}$$

The self-play modules presented above are not adaptive. Both feature a fixed opponent sampling distribution $\Omega(\pi'|\pi_o)$ and gating function $\pi^o_{t+1} = G(\pi^o_t)$ which ignores the menagerie to express its functionality.

5 Experimental Environments

- Describe environments used. Transition probability function, reward function.
- Peculiarities of environments (zero-sum games or not?). Environments with various possible "good" strategies.

6 Experimental Details

Anything that helps with experiment replicability of experiments and results:

- Algorithm choices,
- Neural network topologies, activation function used, optimizer (SGD, ADAM).
- Hyper parameters: [PPO: GAE λ, clipping ϵ, batchsize, number of epochs, MDP: discount factor γ]

7 Results

8 Conclusion

- We've presented a general framework for self-play.
- This framework generalises to existing algorithms which make use of self-play
- A novel self-play algorithm is presented (Results)
- Future work will study other possibilities presented within the expressive capabilities of the framework

9 Acknowledgements

References

[Anthony *et al.*, 2017] Thomas Anthony, Zheng Tian, and David Barber. Thinking Fast and Slow with Deep Learning and Tree Search. (Il):1–19, 2017.

[Aytar *et al.*, 2018] Yusuf Aytar, Tobias Pfaff, David Budden, Tom Le Paine, Ziyu Wang, and Nando de Freitas. Playing hard exploration games by watching YouTube. 2018.

[Bansal *et al.*, 2017] Trapit Bansal, Jakub Pachocki, Szymon Sidor, Ilya Sutskever, and Igor Mordatch. Emergent Complexity via Multi-Agent Competition. 2:1–12, 2017.

[Beattie *et al.*, 2016] Charles Beattie, Joel Z. Leibo, Denis Teplyashin, Tom Ward, Marcus Wainwright, Heinrich Küttler, Andrew Lefrancq, Simon Green, Víctor Valdés, Amir Sadik, Julian Schrittwieser, Keith Anderson, Sarah York, Max Cant, Adam Cain, Adrian Bolton, Stephen Gaffney, Helen King, Demis Hassabis, Shane Legg, and Stig Petersen. DeepMind Lab. pages 1–11, 2016.

[Duan *et al.*, 2016] Yan Duan, John Schulman, Xi Chen, Peter L. Bartlett, Ilya Sutskever, and Pieter Abbeel. RL\$^2\$: Fast Reinforcement Learning via Slow Reinforcement Learning. pages 1–14, 2016.

[Firoiu *et al.*, 2017] Vlad Firoiu, William F. Whitney, and Joshua B. Tenenbaum. Beating the World's Best at Super Smash Bros. with Deep Reinforcement Learning. 2017.

[Henderson *et al.*, 2017] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep Reinforcement Learning that Matters. 2017.

[Laurent *et al.*, 2011] Guillaume J. Laurent, Laëtitia Matignon, and N. Le Fort-Piat. The world of independent learners is not markovian. *International Journal of Knowledge-Based and Intelligent Engineering Systems*, 15(1):55–64, 2011.

[Leibo *et al.*, 2018] Joel Z. Leibo, Cyprien de Masson D'Autume, Daniel Zoran, David Amos, Charles Beattie, Keith Anderson, Antonio García Castañeda, Manuel Sanchez, Simon Green, Audrunas Gruslys, Shane Legg, Demis Hassabis, and Matthew M. Botvinick. Psychlab: A Psychology Laboratory for Deep Reinforcement Learning Agents. pages 1–28, 2018.

[Mnih *et al.*, 2013] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing Atari with Deep Reinforcement Learning. pages 1–9, 2013.

[Samuel, 1959] A.L. Samuel. Some studies in machine learning using the game of checkers. *Ibm Journal*, 3(3):210, 1959.

[Silver *et al.*, 2016] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of {Go} with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[Silver *et al.*, 2017a] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm. pages 1–19, 2017.

[Silver *et al.*, 2017b] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George Van Den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of Go without human knowledge. *Nature*, 550(7676):354–359, 2017.

[Sutton and Barto, 1998] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. 1998.

[Tesauro, 1992] Gerald Tesauro. Practical Issues in Temporal Difference Learning. *Machine Learning*, 8(3-4):257–277, 1992.

[Tesauro, 1995] Gerald Tesauro. Temporal Difference Learning and TD-Gammon. *Commun. ACM*, 38:58–68, 1995.

[Treutwein, 1995] Bernhard Treutwein. Adaptive Psychophysical Procedures. 1995.

[Van Der Ree and Wiering, 2013] Michiel Van Der Ree and Marco Wiering. Reinforcement learning in the game of Othello: Learning against a fixed opponent and learning from self-play. *IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning, ADPRL*, pages 108–115, 2013.

Appendix

Add algorithms for the following environment models:

- Multi-Agent Markov Decision Process (MMDP)
- dec-POMDP

Explain how they differ from Markov Games and briefly talk about their tuple representation, don't go too deep into them.