In *Proceedings of the 8th Brazilian Conference on Intelligent Systems* (BRACIS), Salvador, BA, Brazil. October 2019.

Building Self-Play Curricula Online by Playing with Expert Agents in Adversarial Games

Felipe Leno Da Silva, Anna Helena Reali Costa
University of São Paulo
São Paulo, Brazil
{f.leno,anna.reali}@usp.br

Peter Stone
The University of Texas at Austin
Austin, TX, USA
pstone@cs.utexas.edu

***Abstract*—Multiagent reinforcement learning algorithms are designed to enable an autonomous agent to adapt to an opponent's strategy based on experience. However, most such algorithms require a relatively large amount of experience to perform well. This requirement is problematic when opponent interactions are expensive, for example, when the agent has limited access to the opponent during training. In order to make good use of the opponent as a resource to support learning, we propose *SElf-PLay by Expert Modeling* (SEPLEM), an algorithm that models the opponent policy in a few episodes, and uses it to train in a simulated environment where it is cheaper to perform learning steps than in the real environment. Our empirical evaluation indicates that *SEPLEM*, by iteratively building a *Curriculum* of simulated tasks, achieves better performance than both only playing against the expert and using pure Self-Play techniques. SEPLEM is a promising technique to accelerate learning in multiagent adversarial tasks.**

I. Introduction

Reinforcement Learning (RL) [1] agents have the ability to learn through experience, which allows them to autonomously acquire behaviors and learn how to solve tasks under minimal supervision. However, classical model-free RL techniques are known to require a huge number of interactions with the environment for learning optimal task solutions [2], especially if other autonomous agents are in the environment, requiring learning how to adapt to another agent's actions [3].

For this reason, additional techniques are needed for accelerating learning. *Curriculum Learning* [4] has recently become a popular approach for enabling the solution of complex tasks. *Curriculum Learning* builds on the much older general idea of decomposing a complex task into several simpler ones [5]. The agent can then reuse knowledge from simpler tasks to learn faster overall when compared to directly training in the complex task or at least to reduce the time taken in the last task (often costlier or riskier than the simple tasks). In adversarial tasks, the agent might play against different versions of itself, building a *Curriculum* of *self-play* tasks aiming at learning how to play against a skillful agent [6].

However, playing against a copy of the agent might cause a suboptimal exploration of the task, as the agent only adapts against itself [7], failing to explore varied strategies. For this reason, more effective self-play techniques train against a different version of the agent, such as training more than one agent simultaneously [8]–[10], or against a previously successful version of the agent [6]. We here propose a self-play algorithm to learn how to adapt against *expert* agents in adversarial tasks faster than playing directly against them. When trying to learn how to beat an expert, the agent might take very long to find a good solution, as winning the game often requires a carefully selected sequence of actions that has to be initially observed randomly. However, the agent might benefit from learning how to beat easier opponents first, only facing the expert after achieving reasonable performance.

We here contribute a self-play approach, named *SElf-PLay by Expert Modeling* (SEPLEM), that uses the expert as a resource for building self-play tasks. Instead of playing against copies of the learning agent [7], we build a model of the expert by playing against it a few rounds. Then, we use the model for building a *Curriculum* task. Our contributions are:

1) We propose to use the expert agent that we intend to defeat as a resource for building a *Curriculum*;
2) We propose SEPLEM for building a *Curriculum* online, where tasks that do not require interactions with the expert are generated and used in the learning process;
3) We empirically show that SEPLEM enables the agent to learn how to adapt to an expert's policy while playing fewer games against the actual expert when compared to similar approaches.

II. Background

Single-agent Reinforcement Learning problems can be modeled as *Markov Decision Processes* (MDP). An MDP is composed of a tuple $\langle \mathcal{S}, \mathcal{A}, T, R \rangle$, where $\mathcal{S}$ is a set of environment states, $\mathcal{A}$ is a set of available actions, T is a transition function, and R is a reward function. As the agent initially does not have information about T and R, the learning process is carried out by applying actions in the environment and observing samples of those functions. At every decision-making step, the agent observes the state s and chooses action a. The action will cause a state transition $s' = T(s, a)$, which on its turn generates a reward $r = R(s, a, s')$. This cycle is repeated until a termination condition is achieved and samples of $\langle s, a, s', r \rangle$ are the only information the agent has to learn how to solve a task. The goal

We gratefully acknowledge support from FAPESP (2015/16310-4 and 2018/00344-5), CNPq (425860/2016-7 and 307027/2017-1), CAPES (Finance Code 001), NSF (IIS-1637736, IIS-1651089, IIS-1724157), ONR (N00014-18-2243), FLI (RFP2-000), ARL, DARPA, Intel, Raytheon, and Lockheed Martin.

of the agent is to learn a policy π for dictating which action should be applied in the current state: $\pi : S \rightarrow A$. The optimal policy (the best solution for a particular MDP) π^* is the one that maximizes the sum of rewards in a predefined horizon. A possible way to learn such a policy is to first estimate the quality of each action in each state $Q : S \times A \rightarrow \mathbb{R}$. Some algorithms (e.g. Q-Learning [1]) provably converge to the true Q function: $Q^*(s,a) = E\left[\sum_{i=0}^{\infty} \gamma^i r_i\right]$, where r_i is the reward received after i steps from using action a on state s and following the optimal policy on all subsequent steps. After learning Q^*, it is straightforward to extract the optimal policy: $\pi^*(s) = \arg\max_{a \in A} Q^*(s,a)$.

When more than one agent is present in the environment, the state space is composed of local states of multiple agents $\mathcal{S} = \mathcal{S}_1 \times \mathcal{S}_2 \times \cdots \times \mathcal{S}_n$ and multiple action sets exist $\mathcal{U} = \mathcal{A}_1 \times \mathcal{A}_2 \times \ldots \mathcal{A}_n$. The transition function is now dependent on the *joint* action and every agent has their own reward function. Therefore, the learning agent has to coordinate with the other agents, for maximizing its sum of rewards according to joint actions. If the policy of the other agents can be assumed to be fixed, it is possible to learn in such a multiagent setting as in an MDP. However, if the other agents have non-stationary policies (i.e., they are learning or adapting to the learning agent's policy), then the task should be modeled as a Stochastic Game [11].

Even though the optimal policy will be eventually learned under certain conditions [12], learning Q can take a lot of data, as usually the state-action space to be explored is huge for all but the most simple tasks. For this reason, RL needs additional techniques for accelerating and scaling learning. An early but very effective idea is to first solve manually defined simplifications of the task [5]. Then, the agent can progressively solve harder tasks until eventually solving the full task. Recently, this idea has been revisited by *Curriculum Learning* approaches [13], [14] that automatically decompose a hard RL task into a sequence of easier tasks. If an appropriate task decomposition and sequencing is available, those techniques have been shown to accelerate learning in challenging domains. We here propose a *Curriculum Learning* approach based on switching the learning task between the real task with other agents and a simulated task used for accelerating learning. The problem formulation is described next.

III. Problem Statement

We are interested in tasks composed of two or more agents. In this paper, we describe the two-agent case, but the formulation can be easily extended to more agents. A learning agent l has to solve an adversarial RL task $\mathcal{T}$ in the presence of an expert agent e. Although e is expected to have a good policy π^e for solving $\mathcal{T}$, no information about π^e is initially available to l. Since $\mathcal{T}$ is an adversarial task (e.g., a zero-sum game), π^l should be specialized to achieve the best possible reward against π^e: $\pi^{l*} = argmax_{\pi^l} V^{\pi}$, where $\pi = \{\pi^l, \pi^e\}$.

In order to solve the task, we assume that l can challenge e for a round (episode) at any time, but playing with e is costly because it requires training in the real task. However, l is also equipped with a simulator $\mathcal{T}_s$ to practice the task whenever it wants without the presence of e. For that, l has to simulate the moves of the other player (e.g., playing against itself).

Therefore, the goal of the agent is to **learn** $\mathcal{T}$, while taking advantage of simulated episodes in $\mathcal{T}_s$, to **minimize** the number of episodes against e. We assume the following:

- *Rules of the game*: In order to build $\mathcal{T}_s$, the learner needs to be able to simulate the transition and reward functions, as well as identify initial and goal states. This information can be given to l as a black box simulator.
- *Observable actions*: We assume that l is able to accurately observe expert actions when interacting with it, even without knowing π^e.
- *Fully-Observable States*: We assume that the agents are solving a task in which the state (or observations) of one agent is observable to the other. Those observations will be used in our method to model the expert.

Those assumptions are similar to or less restrictive than the ones found in contemporary related works [7], [14], [15].

IV. Building Self-Play Curricula

We propose a novel algorithm, hereafter named *SElf-PLay by Expert Modeling* (SEPLEM), as a solution to the challenge presented in the previous section. Algorithm 1 is a high-level description of our proposal. At first, the agent has no knowledge about the policy π^e and, for this reason, n_e preliminary rounds are played against e (line 3). After each step against the expert, the tuple $\langle s_k, a_k^e \rangle$ (state and action executed by e) is recorded in a set $\mathcal{I}$. This set is then used for estimating a policy π^s that resembles the observed expert behavior (line 4). This model is later used for playing in the simulator $\mathcal{T}_s(\pi^s)$ (line 5). In order to avoid adapting only to π^s, $\mathcal{T}_s$ will be played for n_s rounds only, and the whole process will be iteratively repeated for building better models of e.

Algorithm 1 SEPLEM task generation

Require: task $\mathcal{T}$, simulator $\mathcal{T}_s$, number of rounds against expert n_e, number of self-play rounds n_s.

1: $\mathcal{I} \leftarrow \emptyset$
2: **while** learning **do**
3: play n_e rounds in $\mathcal{T}$, $\forall$ step $k : \mathcal{I} \leftarrow \mathcal{I} \cup \langle s_k, a_k^e \rangle$
4: $\pi^s \leftarrow MODEL(\mathcal{I})$
5: play n_s rounds in $\mathcal{T}_s(\pi^s)$

Ideally, the modeling function should be able to easily adapt to new instances, avoiding the need of recomputing all the set $\mathcal{I}$ each time line 4 is executed (how expensive recomputing π^s is depends on the used modeling function). Any supervised learning algorithm could be used in the general case, though (the current expert state s_k is the feature set and the chosen action a_k^e is the class). If the expert is reactive (i.e., changes its strategy according to the previous actions applied by the agent), a subset of the previous agent's actions can be included as additional features [16]. The simplest way to compute the policy π^s is by calculating a probability for each action as:

$\forall a \in A : \pi^s(s,a) \leftarrow \frac{n(s,a,\mathcal{I})}{\sum_{a_i \in A} n(s,a_i,\mathcal{I})}$, where $n(s,a,\mathcal{I})$ is the number of tuples in $\mathcal{I}$ with state s and action a.

However, this model might be inapplicable when the exact same state is not expected to be observed frequently (e.g., continuous domains). In this case, a more sophisticated model should be used. Algorithm 2 shows how a model can be built by using a Supervised Learning algorithm. First, the model is initialized or retrieved from a previous iteration (line 1). Then, the observed instances are used to train the model, where the state variables are the features for the classifier and the chosen action the class (line 2). Finally, the trained model is used to define the simulated policy, where the action with highest probability for the current state is chosen. Alternatively, $\mathcal{I}$ might be saved each time the algorithm is executed and $\mathcal{M}$ might be trained from scratch on line 2 using all the data. Notice that every time the agent improves the expert model, a new task is implicitly created. The objective of this task is beating the model in the simulator. Therefore, by switching between playing against the expert and different expert models, the agent is building a *Curriculum* of adversarial tasks and reusing the gathered knowledge across them.

Algorithm 2 Classification $MODEL$

Require: Set of observed instances $\mathcal{I}$
1: $\mathcal{M} \leftarrow$ initialize classifier.
2: $train(\mathcal{M}, \mathcal{I})$
3: **return** $\mathcal{M}$

V. Experimental Evaluation

In order to evaluate SEPLEM, we empirically compare the performance achieved by both our algorithm and alternative similar approaches for solving the same problem. The following approaches were considered:

- **Q-Learning**: The expert is continually challenged for new rounds and the regular Q-Learning algorithm is applied as if the expert were part of the environment.
- **Self-Play**: The agent trains exclusively from simulated rounds against itself. Instead of using the expert, only self-play rounds are used to apply the RL algorithm, similarly to the *AlphaZero* algorithm [6].
- **SEPLEM-Prob**: Here, our approach is implemented with a probabilistic model (counting samples as described in Sec. IV). Even though this model is only expected to work in discrete state spaces,its simplicity and the speed with which this model is able to mimic the expert moves makes it appealing when applicable.
- **SEPLEM-NN**: Here, the expert policy is modeled through a Neural Network. Even though this model takes longer to converge, it has better generalization capabilities and is applicable to a wide range of domains, including continuous state spaces.

A. Evaluation Domain

Tic-Tac-Toe is a simple and well-known competitive two-player game. The players (represented by either a *X* or *O* mark) alternate by choosing an open position in a 3×3 board to write down its mark. The objective is to get three marks in a row, horizontally, vertically, or diagonally, after which the game ends. In case the nine positions are filled and no player won, the game finishes as a draw. Despite its simplicity, this domain has interesting properties for serving as a proof-of-concept to our algorithm. First, it is easy to come with a manually defined optimal policy that never loses the game, as well as to progressively degrade this policy to study the effect of using experts of different levels. Second, random policies perform poorly (most likely losing all rounds) when playing against high-level agents, which means that in this domain the agents are expected to benefit from smarter ways to improve initial performance. Third, due to the relatively simple representation of the environment, full observability, and ease of observing opponent's actions, we can study the performance on learning and adapting to game strategies without additional noise introduced by partial observability or learning by using more complicated state representations.

A player playing the optimal policy never loses, which means that two players playing perfectly would always draw the game. The *expert* was programmed with a policy that never loses, while the learning agent has to learn from scratch. This domain is modeled as an RL problem by considering each of the possible positions as a state variable. Each of the 9 state variables might be *empty*, *occupied by the player*, or *occupied by the opponent*, hence three values are possible for each of them. The available actions are the positions that can be marked. A reward of $+1$ is awarded to the winner, while a reward of -1 is given to the loser. In case of a draw, both players receive -0.1 reward. In order to evaluate the algorithms, we designed different scenarios to evaluate different aspects of the algorithm, as described in the next subsections. In all cases, the extracted metric was the average of the sum of rewards observed in 16 rounds against the expert. Those evaluations are executed every 150 learning steps and no exploration or updates are executed during evaluation rounds. After each game, the players switch sides. During training, the agents might freely use the simulator or "challenge" the expert according to their learning algorithms.

The environment and algorithms were implemented in *Python* and *TensorFlow* for Neural Network training[1]. The parameters below were set based on preliminary experiments. The algorithms are not very sensitive to them: Similar results are observed for similar parameters.

- **SEPLEM-Prob**: A probabilistic model as described in Section IV was used. For this algorithm, we set the number of expert interactions $n_e = 200$ and the number of simulated steps $n_s = 5,000$.
- **SEPLEM-NN**: The model has a single fully-connected hidden layer of 20 neurons. The input layer receives the state variables (9 neurons in total) and the output layer consists of all possible actions (9 possible positions to mark). We minimized the *cross-entropy* cost function by

[1] Available at https://github.com/f-leno/Online_SelfPlaying_Curriculum

using the *Adam* optimizer [17]. A softmax function is used to stochastically select actions for the simulated expert. For this algorithm, $n_e = 200$ and $n_s = 5,000$.

For all algorithms, Q-Learning was used as the base RL algorithm, configured with $\alpha = 0.2$ and $\gamma = 0.99$. For all the graphs in the next section, the shaded areas correspond to the 95% confidence interval over 500 repetitions.

B. Learning against an expert

We first evaluate our algorithm in the most straightforward setting. The agent has to learn how to play against an expert by playing as few games against it as possible. The policy followed by the expert is unbeatable. Therefore, an optimal policy against this agent would always draw the game. Notice that we do not count learning steps taken in simulation for *SEPLEM*, as we are interested in problems where only playing against the real expert is costly (e.g., any task in which the real expert is a human).

Figure 1 shows that both *SEPLEM-Prob* and *SEPLEM-NN* achieve a significantly faster learning speed between around 800 and 3000 learning steps, converging to the same final performance as using *Q-Learning* directly against the expert. Since *Self-Play* never plays against the expert, the figure shows its performance in a way that the x-axis position corresponds to the same number of learning steps in both simulated or real rounds for both *Self-Play* and *SEPLEM*. Initially (when its policy resembles the random policy), *Self-Play* shows a huge improvement over Q-Learning, while *SEPLEM* is still acting as *Q-Learning* and gathering samples for opponent modeling. However, *Self-Play* quickly overfits to its own policy, deviating from the real objective of beating the expert. As a consequence, *Self-Play* finally converges to a lower performance than the other algorithms.

In order to understand the difference between using the probabilistic and Neural Network models with *SEPLEM*, we also show the results when *counting* steps in simulation as if they were as costly as steps against the expert in Figure 2. Notice that *SEPLEM-Prob* learns faster than Q-Learning even when steps in simulation are counted. Since the probabilistic model is very simple to learn, after only a few steps playing against the expert *SEPLEM-Prob* already has a good simplified model of the expert, enabling a faster learning speed. *SEPLEM-NN* however, uses a model that requires a higher number of samples to build a good representation of the expert. *Self-Play* takes much longer to improve its performance than the other algorithms, which is not surprising since this algorithm never plays against the expert.

C. Transfer over different experts

For our last setting, imagine a different scenario. If the final goal is to play against an expert that is unavailable during learning, but another expert is available to play, it might be possible to transfer the learned policy across the experts. However, since the experts might be following different policies, transferring this knowledge might be challenging, as the agent needs to be able to cope with the differences in their policies. Nevertheless, *SEPLEM* might be useful if playing against the available expert is still costly, and for this reason we built a setting to evaluate this scenario. We executed experiments in two scenarios, degrading the optimal policy of the expert according to a parameter p, where the expert follows the best policy with probability p and a random action otherwise:

1) *Transfer from weaker to stronger*: We evaluate here the most common transfer scenario. A lower-level expert is available to practice against, while the higher-level expert is the agent's true objective. We train the *SEPLEM* agent against a $p = 0.7$ expert, and evaluate it against an *optimal* expert.
2) *Transfer from stronger to weaker*: The available expert might be stronger than the one the agent is trying to beat, even though this is a less common scenario. Here, we train the agent against an *optimal* expert, and evaluate the agent performance against a $p = 0.7$ expert.

In both scenarios, we compare the performance of *SEPLEM* with the performance achieved by applying *Q-Learning* directly against the expert that is the agent's objective.

1) Results: Transfer from weaker to stronger: Figure 3 shows the results for this scenario. Clearly, using an expert as a resource to accelerate learning is advantageous even if the policy of the available expert does not match precisely the expert the agent is trying to beat. The major improvement is during the beginning of the learning process, where the use of simulated plays enables the agent to learn very quickly how to avoid losing the game. Both *SEPLEM-NN* and *SEPLEM-Prob* have similarly positive results in this scenario, showing that both simple and complex models can be used in this setting.

2) Results: Transfer from stronger to weaker: Figure 4 shows the experimental results for this scenario. Even though *SEPLEM-Prob* improves the learning speed until around step 2000, it is not able to improve over the performance achieved against the *optimal* expert. Since the $p = 0.7$ expert allows the agent to perform better, *Q-Learning* surpasses *SEPLEM-Prob* after around 2000 learning steps. However, *SEPLEM-NN* is able to learn a policy that achieves optimal performance against the $p = 0.7$ expert as well. This result shows that the stochastic selection of actions allowed the agent to explore behaviors that were not demonstrated by the expert, resulting in a better performance for this transfer scenario.

D. Summary of experimental results

The main findings in the experimental evaluation were:

- *SEPLEM* achieves a better learning performance than both Q-Learning and playing only self-play matches.
- Both simple (e.g. probabilistic) and complex (e.g. *Neural Network*) models can be effectively used with *SEPLEM*. The choice of model to mimic expert behaviors should be taken according to the complexity of the desired task.
- *SEPLEM* can also be used to transfer across experts, where a complex model is more effective when transferring from a more competent to a less competent expert.

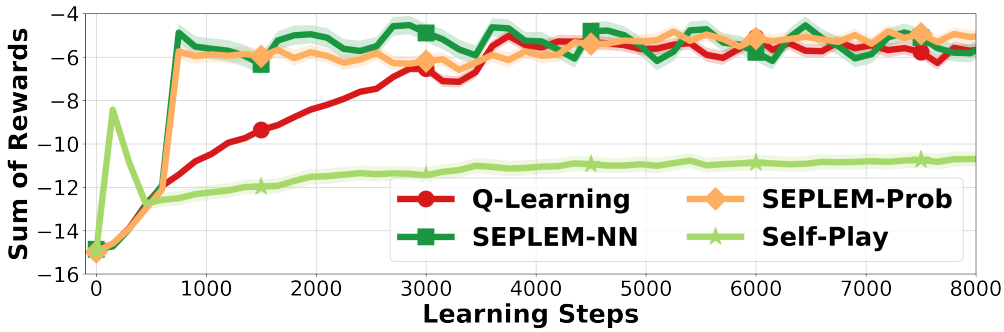

Fig. 1: Average sum of rewards observed in evaluation steps against the expert for each algorithm. Here, steps taken against simulated experts are not counted for *SEPLEM-Prob* and *SEPLEM-NN*.

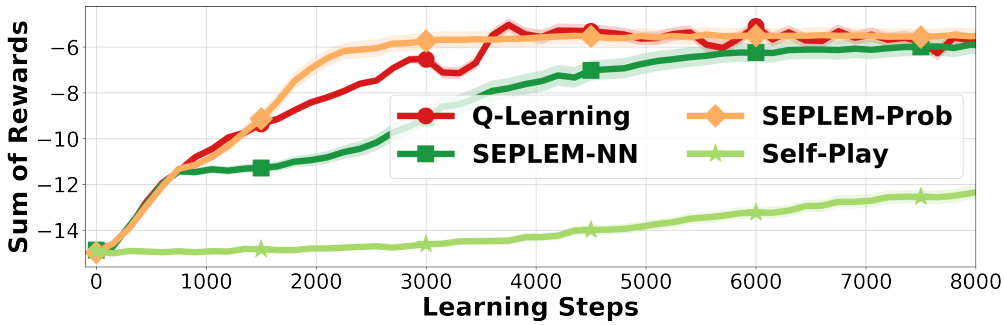

Fig. 2: Average sum of rewards observed in evaluation steps against the expert for each algorithm. Here, steps taken against simulated experts are counted for all Algorithms.

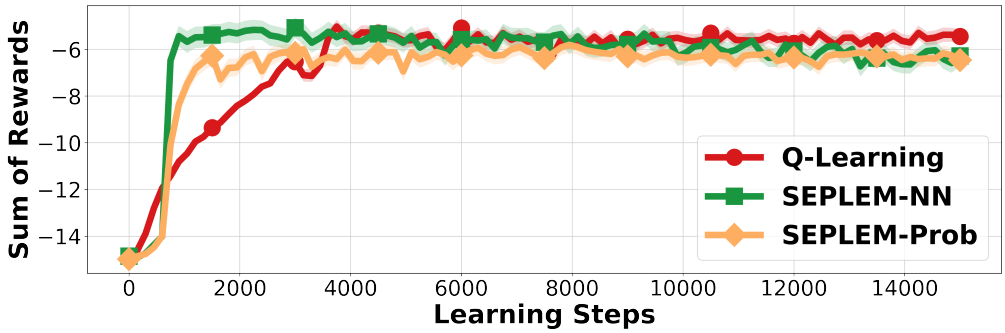

Fig. 3: Evaluation of the performance against an *optimal* expert while training against a $p = 0.7$ expert.

VI. Related Works

Even though the general idea of solving "easy missions" first dates back to the 90s [5], the modern approaches of *Curriculum Learning* and *Self-Play* have become popular in the past few years. *AlphaGo* and its newer versions are related to our work, as they intended to win matches against a "human expert" [6], [7]. However, they execute numerous self-play matches, virtually learning how to play against *any* player before facing the expert. We here focus on learning how to adapt to a specific agent, making use of preliminary matches as a resource for learning faster. Some *Curriculum Learning* approaches focus on building sequences of subtasks to learn a more complex task faster [18]–[20], as we also do. However, they usually focus on transferring knowledge through single-agent tasks, while we focus on building a *Curriculum* to learn faster how to adapt against another agent's strategy. Rosin and Belew [9] introduced an earlier approach in which two agents evolve their strategy by direct competition. Although we also focus on learning through competition, our paper is concerned with learning faster by modeling an expert and playing in a simulator. The ability of modeling another agent has been studied by a vast literature before [21]. Some approaches are specialized to compute the best strategy against a specific opponent as we do [22]. However, those works usually require extensive domain knowledge to quickly compute the best strategy once the opponent model is known, while here we also learn how to play the game from scratch. Nonetheless, many

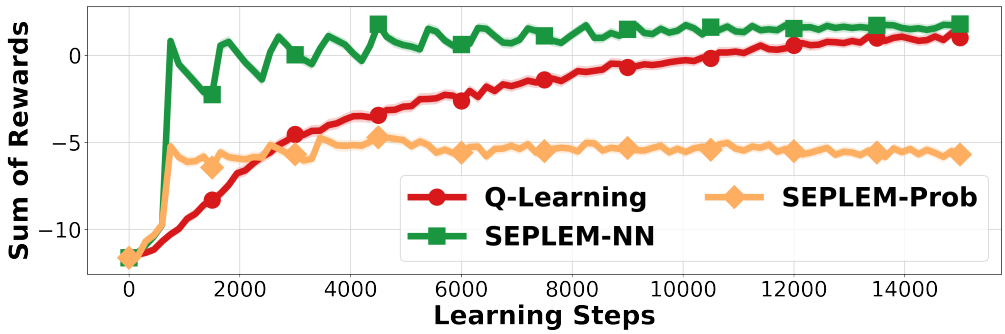

Fig. 4: Evaluation of the performance against a $p = 0.7$ expert while training against an *optimal* expert.

of those methods and some from other paradigms of *Transfer Learning* [20] could be integrated into SEPLEM. Bansal *et al.* [10] has a similar purpose of building a multiagent *Curriculum*, but they aim at showing emergent competition behaviors when multiple agents are trained together, instead of training one agent to face another.

VII. Conclusion and Further Work

Learning adversarial tasks requires the ability to quickly adapt to an opponent's policy, often in a setting where gathering samples of interactions is expensive. Inspired by the recent successes of *self-play* and *Curriculum Learning* procedures, we here propose *SElf-PLay by Expert Modeling* (SEPLEM), an algorithm that builds a model of the opponent in a few episodes and then trains the agent in simulated tasks that are much cheaper than playing against the real opponent. We performed empirical evaluations that showed that *SEPLEM* performs better than pure self-play and regular RL. Our experiments also showed that *SEPLEM* is effective for transferring knowledge across different opponents.

SEPLEM opens several avenues for possible future work. The algorithm could be adapted to not only model opponents in adversarial tasks, but also to model other agents in cooperative or self-interested scenarios. A current limitation of our algorithm is that *SEPLEM* assumes the opponent follows a fixed or slowly-changing policy. In order to cope with opponents that change their strategies, *SEPLEM* could be combined with *Concept Drift* [23], [24] techniques, which would allow the learning agent to detect changes of strategy and to adapt accordingly. Finally, some works have successfully mimicked demonstrated policies even when it is not possible to accurately observe the actions taken by the demonstrator [25], [26]. *SEPLEM* could be combined with those approaches for modeling expert policies under partial observability.

References

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 1998.

[2] M. L. Littman, "Reinforcement Learning Improves Behaviour from Evaluative Feedback," *Nature*, vol. 521, no. 7553, pp. 445–451, 2015.

[3] ——, "Markov Games as a Framework for Multi-Agent Reinforcement Learning," in *ICML*, 1994, pp. 157–163.

[4] S. Narvekar, J. Sinapov, M. Leonetti, and P. Stone, "Source Task Creation for Curriculum Learning," in *AAMAS*, 2016, pp. 566–574.

[5] M. Asada, S. Noda, S. Tawaratsumida, and K. Hosoda, "Vision-based Behavior Acquisition for a Shooting Robot by Using a Reinforcement Learning," in *IEEE Workshop on Visual Behaviors*, 1994, pp. 112–118.

[6] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez *et al.*, "Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm," *CoRR*, vol. abs/1712.01815, 2017.

[7] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche *et al.*, "Mastering the Game of Go with Deep Neural Networks and Tree Search," *Nature*, vol. 529, pp. 484–503, 2016.

[8] G. Tesauro, "Temporal Difference Learning and TD-Gammon," *Commun. ACM*, vol. 38, no. 3, pp. 58–68, Mar. 1995.

[9] C. D. Rosin and R. K. Belew, "New Methods for Competitive Coevolution," *Evolutionary computation*, vol. 5, no. 1, pp. 1–29, 1997.

[10] T. Bansal, J. Pachocki, S. Sidor, I. Sutskever, and I. Mordatch, "Emergent Complexity via Multi-Agent Competition," in *ICLR*, 2018.

[11] M. Bowling and M. Veloso, "An Analysis of Stochastic Game Theory for Multiagent Reinforcement Learning," Tech. Rep., 2000.

[12] C. J. Watkins and P. Dayan, "Q-Learning," *Machine Learning*, vol. 8, no. 3, pp. 279–292, 1992.

[13] M. Svetlik, M. Leonetti, J. Sinapov, R. Shah, N. Walker, and P. Stone, "Automatic Curriculum Graph Generation for Reinforcement Learning Agents," in *AAAI*, 2017, pp. 2590–2596.

[14] F. L. D. Silva and A. H. R. Costa, "Object-Oriented Curriculum Generation for Reinforcement Learning," in *AAMAS*, 2018.

[15] S. Sukhbaatar *et al.*, "Intrinsic Motivation and Automatic Curricula via Asymmetric Self-Play," in *ICLR*, 2018.

[16] D. Chakraborty and P. Stone, "Multiagent Learning in the Presence of Memory-Bounded Agents," *Autonomous Agents and Multi-Agent Systems*, vol. 28, no. 2, pp. 182–213, Mar 2014.

[17] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," in *ICLR*, 2015.

[18] S. Narvekar, J. Sinapov, and P. Stone, "Autonomous Task Sequencing for Customized Curriculum Design in Reinforcement Learning," in *IJCAI*, 2017, pp. 2536–2542.

[19] C. Florensa, D. Held, M. Wulfmeier, M. Zhang, and P. Abbeel, "Reverse Curriculum Generation for Reinforcement Learning," in *CoRL*, 2017.

[20] F. L. D. Silva and A. H. R. Costa, "A Survey on Transfer Learning for Multiagent Reinforcement Learning Systems," *Journal of Artificial Intelligence Research (JAIR)*, vol. 64, pp. 645–703, 2019.

[21] S. V. Albrecht and P. Stone, "Autonomous Agents Modelling Other Agents: A Comprehensive Survey and Open Problems," *Artificial Intelligence*, vol. 258, pp. 66 – 95, 2018.

[22] S. Ganzfried and T. Sandholm, "Game Theory-based Opponent Modeling in Large Imperfect-information Games," in *AAMAS*, 2011.

[23] Y. Sun, K. Tang, Z. Zhu, and X. Yao, "Concept Drift Adaptation by Exploiting Historical Knowledge," *IEEE Trans. on Neural Networks and Learning Systems*, vol. , in press, 2018.

[24] E. Liebman, E. Zavesky, and P. Stone, "Autonomous Model Management via Reinforcement Learning," in *AAAI Workshops*, 2018.

[25] B. Price and C. Boutilier, "Accelerating Reinforcement Learning through Implicit Imitation," *Journal of Artificial Intelligence Research (JAIR)*, vol. 19, pp. 569–629, 2003.

[26] F. Torabi, G. Warnell, and P. Stone, "Behavioral Cloning from Observation," in *IJCAI*, 2018, pp. 4950–4957.