Iterated Deep Reinforcement Learning in Games: History-Aware Training for Improved Stability

MASON WRIGHT, University of Michigan, USA
YONGZHAO WANG, University of Michigan, USA
MICHAEL P. WELLMAN, University of Michigan, USA

Deep reinforcement learning (RL) is a powerful method for generating policies in complex environments, and recent breakthroughs in game-playing have leveraged deep RL as part of an iterative multiagent search process. We build on such developments and present an approach that learns progressively better mixed strategies in complex dynamic games of imperfect information, through iterated use of empirical game-theoretic analysis (EGTA) with deep RL policies. We apply the approach to a challenging cybersecurity game defined over attack graphs. Iterating deep RL with EGTA to convergence over dozens of rounds, we generate mixed strategies far stronger than earlier published heuristic strategies for this game. We further refine the strategy-exploration process, by fine-tuning in a training environment that includes out-of-equilibrium but recently seen opponents. Experiments suggest this *history-aware* approach yields strategies with lower regret at each stage of training.

CCS Concepts: • **Theory of computation → Exact and approximate computation of equilibria**; **Quality of equilibria**; • **Computing methodologies** → *Multi-agent reinforcement learning*.

Additional Key Words and Phrases: deep reinforcement learning; double oracle; attack graphs; security games; multi-agent reinforcement learning

ACM Reference Format:
Mason Wright, Yongzhao Wang, and Michael P. Wellman. 2019. Iterated Deep Reinforcement Learning in Games: History-Aware Training for Improved Stability. In *The 20th ACM conference on Economics and Computation (EC '19), June 24–28, 2019, Phoenix, AZ, USA.* ACM, New York, NY, USA, 20 pages. https://doi.org/10.1145/3328526.3329634

EC '19, June 24–28, 2019, Phoenix, AZ, USA

ACM ISBN 978-1-4503-6792-9/19/06.
https://doi.org/10.1145/3328526.3329634

1 INTRODUCTION

The past few years have seen striking advances in computer game-playing, marked by milestones in computer defeat of the world Go champion [38] and reaching professional level performance in poker [5, 28]. These milestones were enabled by fundamental progress in algorithms for deep reinforcement learning (RL) and reasoning about games of imperfect information. The DeepMind research group followed up their AlphaGo success with particularly impressive developments of learning without data from human play, first in Go [40] and then—in amazingly short order—reaching championship levels in chess and shogi with AlphaZero [39].

Demonstrations in artificial games are invaluable for driving research, but the true promise of these developments lies in applying them to strategic decision-making domains of real intrinsic interest. Technically, translating the approaches to games of practical relevance requires nontrivial extensions and generalizations. For example, whereas chess and go are two-player, essentially symmetric, alternating-move zero-sum games with complete information, games of interest typically exhibit many players, non-symmetry, imperfect information (stochastic environments, partial observability, private information), and non-zero-sum utilities.

We seek to exploit the power of deep RL applied to simulated play, à la AlphaZero, in domains exhibiting the complexities mentioned above. The way has been paved by some prior work. The approach of *empirical game-theoretic analysis* (EGTA) [47] builds and reasons about game models induced from simulated play data. Schvartzman & Wellman [36] combine EGTA with RL (but not deep RL) in an iterative manner to learn strategies for a complex market game. The iterative technique of augmenting an enumerated strategy set by the best response to equilibrium is called (for the two-player case) the *double-oracle* (DO) method [24]. Lanctot et al. [21] generalized DO for combination with deep RL in a procedure called *policy space response oracle* (PSRO). Wang et al. [46] employ a version of PSRO to learn effective strategies in a class of green security games.

We build on this work, extending some elements of technique and demonstrating performance for a complex game in the domain of cybersecurity. The setting is a non-zero-sum interaction between attacker and defender over states defined by an *attack graph* [19, 33]. An attack-graph game models a computer system's attack surface as a directed acyclic graph, where each node is a capability that an attacker could have, and each edge is an exploit an attacker can attempt to achieve a new capability [6, 25, 29, 30]. Such games may exhibit significant complexities including imperfect information (stochastic action effects, partial observation of attack graph) and combinatorial action spaces. We specifically tackle the game defined in prior work by Nguyen et al. [30], which employed EGTA over hand-crafted heuristic strategies that use sophisticated optimization methods.

We start with a combination of double-oracle and EGTA (DO-EGTA), an instance of PSRO that uses deep RL as an (approximate) best-response oracle to iteratively extend the set of candidate strategies. Because the game has a combinatorial action space, we need to modify the standard encoding of deep neural network (DNN) policies for deep RL. We introduce a novel trick, called *greedy action set building*, which scales well to large action spaces. It also accounts for value dependencies among component actions, by learning the incremental value adding each candidate to the action set, given the set's current members.

DO-EGTA alternates between (a) solving for a Nash equilibrium over the strategies developed so far, and (b) using deep RL as a best-response oracle to add a beneficial deviation to each agent's strategy set, until no better deviation can be found. On convergence, DO-EGTA returns the equilibria computed from the game over the final strategy sets. Our results show that deep RL can successfully deviate from the best strategies identified by the prior work for this game [30].

In our explorations of DO-EGTA, we found successive training rounds sometimes appeared unstable, in that a strategy would appear with high weight in one round's mixed equilibrium, then

disappear from subsequent equilibria, only to reappear later. This instability represents a form of thrashing, which suggests inefficiency in the training process. This and other issues of training instability are known to occur in multiagent reinforcement learning (MARL) settings and DO in particular, due to the non-stationary training objective for each agent, and the susceptibility to vulnerabilities from strategies not considered.

One known way to address this issue in strategy exploration is to consider response to strategies beyond the current equilibrium [17, 21]. Accordingly, we introduce a *history-aware* extension, HADO-EGTA, which modifies the exploration process to produce more robust strategies. HADO-EGTA's RL stage starts by training against the current equilibrium opponent, then fine-tunes it against a mix of previous equilibrium opponents. The resulting DNN is recorded periodically during fine-tuning, and the best version is selected, based on mean performance against current and previous opponents, subject to being a beneficial deviation against the current one. We find HADO-EGTA tends to produce more consistent strategy improvements per iteration, with stronger final mixed strategies than DO-EGTA. Our (HA)DO-EGTA framework is illustrated in Figure 1.

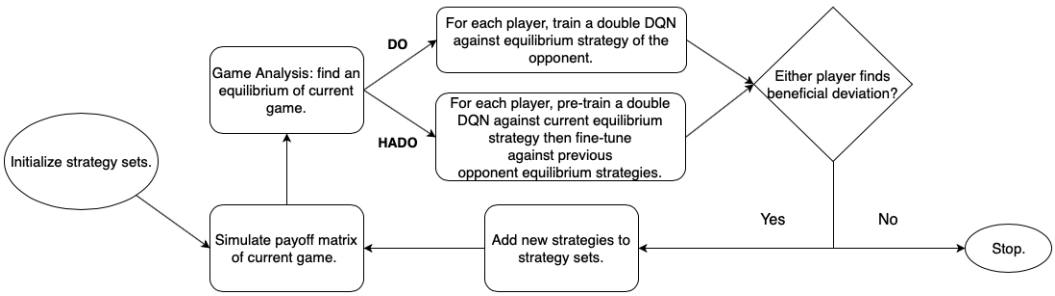

Fig. 1. Flowchart describing the DO-EGTA and HADO-EGTA process.

The key contributions of this work are:

- demonstration of the efficacy of iterated deep RL with game-theoretic analysis, in a realistic and strategically complex imperfect-information game, compared to sophisticated hand-designed strategies from prior literature on attack-graph games;
- greedy action set building: a trick for RL in combinatorial action spaces, which both scales to large action spaces and accounts for dependencies among sub-actions;
- a novel history-aware strategy exploration method that explicitly balances performance with respect to current and previously encountered equilibrium opponents;
- experimental evaluation of methods in this approach, including several innovations in measurement and visualization.

2 GAME-THEORETIC PRELIMINARIES

We model the strategic interaction between attacker and defender as a *two-player normal-form game* $G = (S, U)$, with $S = (S_a, S_d)$ representing the finite sets of *pure strategies* available to the attacker and defender, respectively. In our study, the attacker strategy set S_a and the defender set S_d are initialized with heuristic strategies. When the players simultaneously select a strategy *profile* $(s_a, s_d) \in S_a \times S_d$, they receive expected payoffs given by utility function $U = (U_a, U_d)$, with $U_d : S_a \times S_d \to \mathbb{R}$ for defender reward, and similarly with U_a for attacker reward. There is no explicit representation of U; rather we have a game simulator which generates stochastic payoff samples for a given profile.

More generally, attacker and defender can play *mixed strategies* σ_a and σ_d which are probability mass functions over the players' finite strategy sets. A *Nash equilibrium* is a mixed strategy profile $\sigma = (\sigma_a, \sigma_d)$ such that for any deviating strategy s_a or s_d, the player's expected payoff when deviating is no better than its expected payoff under σ. That is, σ is a Nash equilibrium if and only if $U_a(s_a, \sigma_d) \leq U_a(\sigma_a, \sigma_d)$, for all $s_a \in S_a$ (and similarly for the defender).

The *regret* of a mixed-strategy profile σ, or $\rho(\sigma)$, is the maximum any agent i can gain in expectation by unilaterally deviating from its mixed strategy in σ to an alternative strategy s_i. We define the regret $\rho(\sigma_i, \sigma^*)$ of a mixed strategy σ_i with respect to a Nash-equilibrium profile σ^* as the expected payoff that agent i would lose by unilaterally deviating to σ_i:

$$\rho(\sigma_i, \sigma^*) \equiv U_i(\sigma^*) - U_i(\sigma_i, \sigma^*_{-i}). \tag{1}$$

3 RELATED WORK

3.1 Attack-graph games

Attack graphs were introduced by Philips & Swiler [33] as a means to model vulnerabilities of computer systems. Graphical models of this form for cybersecurity have been widely studied, and automated attack-graph generation software has been produced, both academic [13, 37] and commercial [1, 14]. Examples of vulnerabilities that might be expressed by edges in the graph include bugs in FTP servers, buffer overflow attacks, and password guessing [19, 34].

Recent works [6, 29] searched for optimal attacker and defender strategies in attack-graph games, where agents compete for control of an attack graph. Miehling et al. introduce the attack graph form we study here [25], and present methods to solve for the best defense policy, assuming the attacker follows a fixed strategy. Nguyen et al. [30] extend the problem to a two-player game. The authors propose heuristic strategies, including some sophisticated ones based on particle filtering, for the attacker and defender. They employ the methods of EGTA [47] to estimate the payoffs over strategies in a specified set and solve for equilibria.

3.2 Deep reinforcement learning

Deep RL has famously succeeded in learning superhuman strategies for extensive-form strategy games like Go, chess, and shogi [39, 40]. It has also been applied with success to problems (like our security game) that involve some or all of: imperfect information, multiple players with asymmetric roles, and non-zero-sum payoffs. Examples include cooperative-competitive games [41], and video games like Super Smash Bros. [7], Dota 2 [32], and Doom [20].

In the security domain, Kamra et al. [18] trained a DNN to learn policies for a *green security game* (zero-sum between a ranger and poachers) with continuous actions. Wang et al. [46] used an iterated deep RL procedure, like ours a version of PSRO, to develop new strategies in a dynamic green security game. Venkatesan et al. [45] demonstrated a system using reinforcement learning (without DNNs) to determine the placement of intrusion detectors to defend a computer network.

A DNN and associated learning algorithm for deep RL known as a deep Q-network (DQN) [27] has been shown to learn strong policies for playing Atari games using only pixel input, based on the Q-learning method. Since the original work on DQN, various new deep RL algorithms have produced solid results, such as asynchronous advantage actor critic [26], trust region policy optimization [35], and Rainbow [12]. In DQN, the *max* operator uses the same values to both select and evaluate an action, which leads to overoptimistic value estimates [43]. van Hasselt et al. introduced the *double DQN* [44] to mitigate this problem. We compared this version of DQN with an actor-critic method and found it to be consistently competitive, and so adopt (double) DQN as our oracle in this study.

The issue of combinatorial action spaces in deep RL was previously addressed by He et al. [10], for the problem of recommending a subset of K news articles from a large candidate set. That work explored multiple methods, such as: (a) representing each news item as an embedding vector, concatenating K vectors, and predicting the value of the set; and (b) estimating the value of each news item independently (based on its embedding vector), and selecting the K highest-value items. The approach of predicting the value of an entire action set is exponential in K, and so does not scale well to large candidate sets. The approach of selecting the top K independently evaluated items by definition fails to account for dependencies among the items.

3.3 Double-oracle and empirical game-theoretic methods

A double-oracle method solves a two-player game iteratively, by first solving the game where only a finite set of strategies may be used, and then using an oracle for each player to derive the best-response strategy against the current equilibrium opponent. The double-oracle method was introduced by McMahan et al. [24], and was applied to security games by Jain et al. [15].

The general method of building game models from simulation payoffs has been termed *empirical game-theoretic analysis* (EGTA) [47]. Schvartzman & Wellman [36] combined RL with EGTA, using tabular Q-learning with tile coding to learn better-response policies in a dynamic trading game; a similar approach using (non-deep) Q-learning was taken in our own recent work [48].

Lanctot et al. [21] presented PSRO as a general procedure combining deep RL with EGTA, demonstrating the flexibility of their ideas on gridworld games. PSRO generalizes what we are terming DO-EGTA primarily by introducing *meta-strategy solvers* (discussed further below), which support a range of approaches for generating opponents to train against in strategy generation. Recently, Wang et al. [46] applied DQN as the oracle for a zero-sum dynamic form of green security game. They found that employing double oracle with DQN may take many rounds and substantial computation to converge, even for modestly sized environments.

3.4 History-aware training in multiagent RL

When iterating deep RL in a game, each agent's learning problem is non-stationary, which can produce thrashing or even cycles in training [2, 8]. EGTA (including DO-EGTA) avoids cyclic behavior by accumulating strategies and computing equilibria each round. However, as noted above and also observed by Lanctot et al. [21], double-oracle methods can produce DNNs that are overfit to the current opponent strategy and do not sufficiently exploit weaker opponents.

The idea of considering opponent history to avoid overfitting to the current opponent has been investigated in prior work. Lanctot et al. [21] proposed particular approaches as meta-strategy solvers within their PSRO framework. For example, their *projected replicator dynamics* method requires that every strategy trained so far be included with at least some minimum probability, ensuring that all opponent strategies are encountered during training. It also forces the mixed strategy of each agent to change by only a small step size from one round to the next. The goal is to inhibit thrashing, in which an agent trained in one round fails to learn what the previous agent did, due to drastic changes in the training opponent's mixed strategy.

Bansal et al. [4] propose opponent sampling, which trains a DNN against a randomly sampled old opponent instead of the current opponent, and appears to speed up training of competitive agents in simulated physics games (perhaps by providing a form of curriculum learning where neither training opponent can improve too quickly), as well as improving the robustness of the trained agents to unseen opponents. Wang et al. [46] promote generalization by training against a weighted average of the current equilibrium opponent mixed strategy and the uniform distribution over opponent pure strategies. Foerster et al. [8] introduce Learning with Opponent Learning Awareness (LOLA), a deep RL learning rule intended to stabilize MARL training, that models the opponent's

policy, predicts the opponent's policy update, and updates one's own policy in anticipation of the opponent's update. Grnarova et al. [9] introduce a method for training the two competing policies in a generative adversarial network (GAN), which is designed to improve training stability, resulting in a mixed strategy over DNNs.

In earlier work on stabilizing EGTA, Jordan et al. [17] found augmenting the set of available pure strategies with even arbitrary beneficial deviations from the current Nash equilibrium can lead to faster convergence compared to adding best responses.

3.5 Evaluation of strategies and training methods

Evaluating strategy quality in games is inherently relative to choice of opponent strategies [3, 42]. We focus our comparisons between training procedures on the performance of the resulting strategies at Nash equilibrium, an approach termed NE-response ranking [16] or Nash averaging [3].

We report results on tournaments between the final mixed strategies from distinct training methods, as suggested for evaluating GANs by Olsson et al. [31]. The performance of a strategy against unseen opponents is an indicator of generalizability, which can be poor even for strategies learned from distinct runs of the same algorithm [21].

4 GAME DESCRIPTION

4.1 Attack-graph model

The game takes place on a *Bayesian attack graph*, of the type defined by Miehling et al. [25]. It is a directed acyclic graph $\mathcal{G} = (V, E)$, where vertices $v \in V$ represent possible attacker capabilities, and edges $e \in E$ are exploits the attacker can use to activate nodes.

An *attack-graph game* is defined by a Bayesian attack graph endowed with additional specifications. The game evolves over a finite number of time steps $\mathcal{T}$. At each time step τ, the state of the graph is simply which nodes are active (i.e., attacker-controlled), indicated by $s_\tau(v) \in \{0, 1\}$. The defender receives only a noisy observation $O_\tau(v) \in \{0, 1\}$ of whether each node is active, based on publicly known probabilities $P_v(o = 1 \mid s = 1)$ (detection probability) and $P_v(o = 1 \mid s = 0)$ (false alarm rate). Positive observations are known as *alerts*.

Attacker and defender act simultaneously at each time step τ. The defender's action space is the power set of nodes V, meaning the defender can choose to defend any subset of the nodes.

The attacker can attack any of the graph's $\wedge$-nodes (*and* nodes), if all the node's parents are active. The attacker can attack any edge to an $\vee$-node (*or* node), if the edge's source is active. Nodes without parents (root nodes) may be attacked in any state. The action space is the power set of eligible $\wedge$-nodes and edges to $\vee$-nodes.

Defender actions override attacker actions, such that any node v that is defended becomes inactive. Otherwise, active nodes remain active; an $\wedge$-node v that is attacked becomes active with probability $P(v)$, and any $\vee$-node becomes active with probability based on the success probabilities $P(e)$ of attacked edges.

Each *goal node*, v, has a value for attacker reward $r_a(v)$ and defender penalty $r_d(v)$. Any item an agent can act on has a cost: $c_d(v)$ for nodes defended, $c_a(v)$ for $\wedge$-nodes attacked, and $c_a(e)$ for edges to $\vee$-nodes attacked. There is a discount factor $\eta \in (0, 1]$ for future rewards.

The defender's loss at a time step is the cost of its action (i.e., total cost of nodes defended), plus the penalty for active goal nodes after that step. The defender's total payoff is the negated, exponentially discounted sum of losses over time. The attacker's total payoff is likewise the discounted sum of rewards, in this case the value for active goal nodes, minus cost of attacks pursued.

4.2 Heuristic strategies

Prior work by Nguyen et al. [30] proposed sophisticated heuristic strategies for the attack-graph game. Specifically, the *random walk attacker* strategy assigns a value to each node or edge by sampling a random process moving from currently active nodes toward goal nodes; the strategy uses a heuristic to select items to attack based on the resulting values. Defender heuristics use a Bayesian belief about which nodes are currently active; the belief is updated using a particle filter. The *random walk defender* strategy simulates a series of future attacker actions by sampling a random process; the defender greedily adds nodes to a set to be defended, until adding another node would not increase the expected payoff.

5 METHODS

5.1 Attack-graph game instances

We study attack-graph games with two kinds of graph topology: *random graphs* and *separate-layers graphs*. Random graphs are Erdős-Rényi graphs, where edges have been added uniformly randomly. Separate-layers graphs are built in layers, where edges are randomly added only from nodes of one layer to the next. Our games are built on two randomly-generated graph instances, one of them separate-layers with 29 nodes (s_{29}), the other random with 30 nodes (r_{30}). In game s_{29}, there are 3 layers, 7 goal nodes with values in $[15, 45]$, and 89 edges. In game r_{30}, there are 6 goal nodes with values in $[10, 20]$, and 100 edges. We present the topology of the graph in game r_{30} in Figure 2. Each game proceeds over $\mathcal{T} = 10$ time steps, with payoff discount factor $\eta = 0.99$.

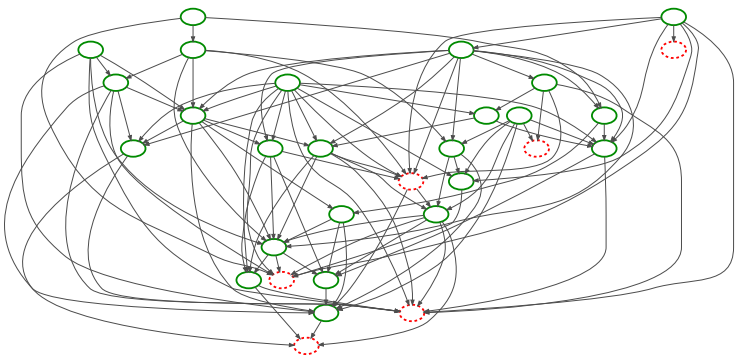

Fig. 2. Topology of random graph r_{30}. Goal nodes have dashed red outlines, non-goal nodes solid green.

5.2 Deep Q-networks

We employ DQN and modify it to tune the number of episodes used to determine if the current DNN is better than the previous best, and to implement (HA)DO-EGTA pre-training and fine-tuning. The DNN takes as input a vector representing the agent's current observation of the state of the environment, filters this vector through multiple convolutional or fully-connected layers and rectifier nonlinearities (ReLUs), and finally yields a regression estimate of the value of taking each action in the current state. Each DNN represents a strategy, mapping from observations to action values based on which actions can be chosen.

We experimented with various architectures for the DNN, before settling on a multilayer perceptron with two hidden, fully-connected layers of 256 neurons each. Pilot experiments also evaluated two fully-connected layers of 128 neurons each; one hidden layer of 256 neurons; and a convolutional

network with two layers, 32 or 64 filters per layer, convolved along one dimension corresponding to the nodes of the attack graph. There were large differences in performance between different network architectures. The structure selected, in the example of the attacker strategy for game r_{30}, is shown in Figure 3. Besides the network architecture, the only hyperparameters we tuned experimentally were the learning rate and the number of training steps. For each of which we tried 3 options and then fixed; other hyperparameters were set based on values suggested in previous works or by intuition, and are likely not highly sensitive.

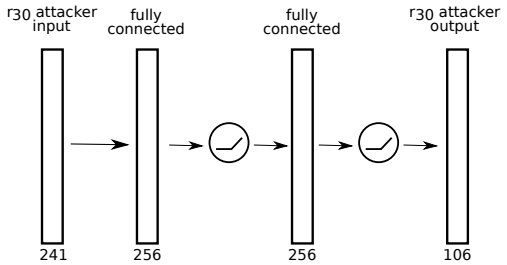

Fig. 3. DNN architecture for r_{30} attacker.

5.3 Game representation for deep RL

The action space in the attack-graph game is huge for attacker and defender, because the defender can choose to defend any subset of the nodes, leading to 2^{30} possible actions in game r_{30}, and the attacker can choose to attack any subset of the $\wedge$-nodes and edges to $\vee$-nodes for which the parents are active, resulting in a tremendously high dimension for the output layer of the DNN.

Our key technique to make the exponential action space tractable is to let each deep RL agent add items to be attacked or defended, one at a time, to an *attack set* or *defense set*. Taking the defender DNN for example, initially in each time step the defense set is empty, and the defender DNN can either add one node to the set or *pass*. Eventually, when the defender DNN passes, the game proceeds with the defender acting on the nodes in the defense set. Thus, the DNN may be called multiple times during one time step, to determine the set of items to be acted on, depending on the DNN's outputs. (Note that the heuristic attacker and defender, unlike the deep RL agents, select sets to act on all at once, instead of one item at a time.)

To encourage deep RL agents' DNNs to add appropriate items to their sets, we impose rules that cause undesirable actions to be treated as a pass. The defender DNN's action is counted as a pass if it selects a node that is already present in the defense set. The attacker DNN's action is counted as a pass if it selects a node or edge already in the attack set, or whose preconditions are not satisfied. And to encourage agents' DNNs to add items to their sets greedily, beginning with the most valuable, we randomly force the DNN to pass with probability 0.1 in each update where at least one item is already in the action set. (The parameter value of 0.1 was selected based on intuition, without experiments testing alternative values; it might be possible to achieve better performance by tuning this value better, or by decaying it toward 0 during training.)

Algorithm 1 presents the procedure for generating the deep RL attacker's action. The attacker DNN ϕ_{att} selects one choice at a time x based on the current attacker DNN input vector *attObs*. This selection can represent *pass*, or an $\wedge$-node or edge to $\vee$-node, to add to the *attackSet*. The choice is legal only if it is to *pass*, or if any parent nodes sufficient for the attack, called $pre(x)$, are in the active node set A (i.e., the set of nodes controlled by attacker).

Algorithm 1 Deep RL attacker's greedy action set building

Require: *attObs*
1: $attackSet \leftarrow \emptyset$
2: **do**
3: $x \leftarrow \phi_{att}(attObs, attackSet)$
4: $isDup \leftarrow x \in attackSet$
5: $isLegal \leftarrow x = pass \vee pre(x) \subseteq A$
6: **if** $\neg isDup \wedge isLegal \wedge x \neq pass$ **then**
7: $attackSet \leftarrow attackSet + \{x\}$
8: **while** $\neg isDup \wedge isLegal \wedge x \neq pass \wedge rand() > 0.1$
9: **return** *attackSet*

The attacker DNN has one action output unit for each $\wedge$-node and each edge to an $\vee$-node, plus one representing pass. This sums to 103 action units in game s_{29} (which has 13 $\wedge$-nodes and 89 edges to $\vee$-nodes) and 106 in game r_{30} (which has 5 $\wedge$-nodes and 100 edges to $\vee$-nodes). The defender DNN has one action output unit per attack-graph node, plus one to pass. This leads to 30 action units in game s_{29} and 31 in game r_{30}.

In the attacker DNN's input vector, we include only data from the attacker's current observation, because the attacker can see the true game state. For each node, an observation bit indicates whether the node is active. For each $\wedge$-node and edge to $\vee$-node, a bit indicates whether the item is reasonable to attack, meaning its preconditions are active but the target is not. One bit indicates whether that item is currently in the attack set. Finally, one vector element indicates how many time steps are left. The attacker DNN input vector has 234 elements in game s_{29} and 241 in game r_{30}. We summarize the attacker DNN's input vector in Table 1, where N is the node count, $N_\wedge$ is the $\wedge$-node count, and $E_\vee$ is the edge to $\vee$-node count.

Attacker		Defender	
Feature	Entry count	Feature	Entry count
isActive	N	*hadAlert*	hN
canAttack	$N_\wedge + E_\vee$	*wasDefended*	hN
inAttackSet	$N_\wedge + E_\vee$	*inDefenseSet*	N
timeStepsLeft	1	*timeStepsLeft*	N

Table 1. DNN input vectors, with entry counts.

The defender DNN's input vector includes data from the defender's previous 3 observations to mitigate the problem that the defender has only noisy data on the game state; we fill in the history with zeros if fewer than 3 past observations exist. (We chose to use a fixed input depth of 3 observations, based on pilot experiments showing that this was sufficient to consistently learn beneficially deviating strategies. It is possible that a different observation depth might yield similar or better performance, or that still better outcomes could be produced by a recurrent neural network that represents the full history of the game.)

For each node and each time step, one bit shows if an alert was observed, and another bit shows whether the node was defended. One bit indicates whether each node is in the defense set. Finally, one vector element per node indicates how many time steps are left (repeating for symmetry, in

case a convolutional layer is used). The input vector has length 232 in game s_{29}, 240 in game r_{30}. Table 1 summarizes the defender DNN's input, where $h = 3$ is the history length.

5.4 DO-EGTA

5.4.1 Definition of DO-EGTA. Let the set of initial heuristic strategies for attacker and defender be $S^H = (S_a^H, S_d^H)$. In each round t of the iterative procedure, we have current strategy sets $S_{a,t}$ and $S_{d,t}$. Let $\sigma_t = (\sigma_{a,t}, \sigma_{d,t})$ be any mixed-Nash equilibrium over these strategy sets under the game's utility function $U = (U_a, U_d)$. Let $g()$ be any deep RL algorithm, such as DQN, that can optimize over an objective function like U_a against an opponent mixed strategy like $\sigma_{d,t}$, returning a new pure strategy $\delta_{a,t}$. Let $\nu()$ be a Nash equilibrium solver, which returns any mixed-Nash equilibrium, given a utility function and finite strategy sets. The DO-EGTA procedure is shown in Algorithm 2.

Algorithm 2 DO-EGTA iterated deep RL method

Require: $U, S^H, g(), \nu()$
$t \leftarrow 0$; $S_{a,t} \leftarrow S_a^H$; $S_{d,t} \leftarrow S_d^H$
$\sigma_t \leftarrow \nu(U, S_{a,t}, S_{d,t})$
do
  $t \leftarrow t + 1$
  $\delta_{a,t} \leftarrow g(U_a, \sigma_{d,t-1})$; $\delta_{d,t} \leftarrow g(U_d, \sigma_{a,t-1})$
  if $U_a(\delta_{a,t}, \sigma_{d,t-1}) > U_a(\sigma_{a,t-1}, \sigma_{d,t-1})$ **then**
    $S_{a,t} \leftarrow S_{a,t-1} + \{\delta_{a,t}\}$
  if $U_d(\delta_{d,t}, \sigma_{a,t-1}) > U_a(\sigma_{d,t-1}, \sigma_{a,t-1})$ **then**
    $S_{d,t} \leftarrow S_{d,t-1} + \{\delta_{d,t}\}$
  $\sigma_t \leftarrow \nu(U, S_{a,t}, S_{d,t})$
while $S_{a,t} \neq S_{a,t-1} \vee S_{d,t} \neq S_{d,t-1}$
return $\sigma_t, S_{a,t}, S_{d,t}$

Note that our Algorithm 2 is a variation on PSROs [21] and the double-oracle method, which begins with a set of heuristics and adds only beneficial deviations to the game.

We are aware that the final payoff achieved after deep RL training is sensitive to the choice of random seeds [11]. One could reduce the likelihood of spurious convergence of DO-EGTA, by requiring multiple trials of DQN to fail for attacker and defender in the same round, before considering the method converged. In other words, instead of stopping after a training round in which both attacker and defender training produce strategies that are not beneficial deviations, one could repeat both attacker and defender training in such cases, up to some maximum count such as 3 attempts each, stopping only if these repeated attempts all fail to yield a beneficial deviation.

5.4.2 DO-EGTA implementation. We use the Gambit library [23] of game solvers to search for Nash equilibria, specifically the *linear complementarity method* known as *gambit-lcp*. When more than one Nash equilibrium is produced, we simply use the first.

We train our DNNs in DO-EGTA for 700,000 time steps (but 1,000,000 for the defender in game r_{30}), with a learning rate of 5×10^{-5} and an artificial discount factor on future rewards of 0.99. During training, we anneal the exploration rate ϵ linearly from 1 to 0.03 over the first half of DO-EGTA training steps, holding it constant at 0.03 afterward.

5.5 HADO-EGTA

5.5.1 Definition of HADO-EGTA. History-aware double-oracle EGTA (HADO-EGTA) extends DO-EGTA with a new strategy exploration procedure. Recall that DO-EGTA strategy exploration simply seeks a best response for each agent i to the current equilibrium opponent mixed strategy $\sigma_{-i,t-1}$:

$$\delta_{i,t} \equiv \arg\max_{s_i} U_i(s_i, \sigma_{-i,t-1}). \tag{2}$$

Instead, HADO-EGTA balances between deviating beneficially against the current equilibrium opponent, and consideration of previous opponents.

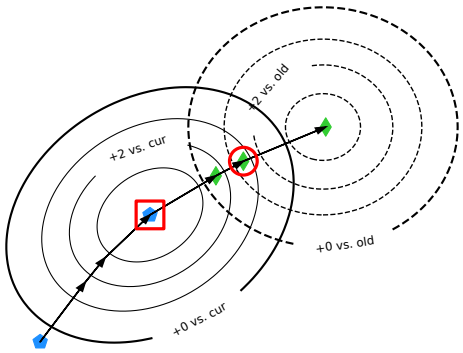

Fig. 4. HADO-EGTA conceptual diagram. Solid contours show payoff vs. current opponent, dashed ones vs. old opponents. Blue pentagons: pre-training, result in red square. Green diamonds: fine-tuning outputs, result in red circle.

Figure 4 diagrams the intuition behind HADO-EGTA strategy exploration. In HADO-EGTA pre-training, a DNN is trained to maximize payoff against the current equilibrium opponent, shown as training the blue pentagon to find the optimum of the solid contour lines. Note that it may be possible to increase payoffs against previous opponents (indicated by dashed contour lines), while maintaining a beneficial deviation against the current opponent (i.e., staying in the bold, solid contour). Fine-tuning maximizes payoff against old opponents (shown as green diamonds), returning a strategy that deviates beneficially against the current opponent while also considering performance against old ones (i.e., the highlighted diamond). This procedure improves the robustness of the equilibrium strategies against both old and unexplored opponent's strategies.

HADO-EGTA strategy exploration, $h()$, takes as input: an agent's utility function; a count $\kappa \geq 2$ of DNNs to record; a decay factor $\gamma \in (0, 1]$ for weighting old opponents; a weighting $\alpha \in [0, 1]$ between current and old opponents; and a history of opponent equilibrium strategies $(\sigma_{d,0}, \ldots, \sigma_{d,t-1})$. It requires a deep RL method for pre-training like the $g()$ of Algorithm 2, and a deep RL fine-tuning method, $g'()$, that proceeds for a limited step count from a pre-trained DNN. HADO-EGTA exploration $h()$ can replace $g()$ as the strategy-exploration routine in DO-EGTA.

We present HADO-EGTA from the attacker's perspective as Algorithm 3. For each agent i, HADO-EGTA strategy exploration seeks a beneficial deviation that also exploits old opponents:

$$\delta_{i,t} \equiv \arg\max_{s_i} \alpha U_i(s_i, \sigma_{-i,t-1}) + (1-\alpha) U_i(s_i, \overline{\sigma}_{-i}), \tag{3}$$

$$\text{s.t. } U_i(s_i, \sigma_{-i,t-1}) > U_i(\sigma_{t-1}). \tag{4}$$

where $\overline{\sigma}_{-i}$ is the weighted sum of history of opponent equilibrium strategies. The defender's strategy exploration procedure is analogous to Algorithm 3. Note that if no interim DNN is a beneficial deviation, the procedure returns *null*, and no new strategy will be added to the agent's strategy set in the current iteration.

Algorithm 3 HADO-EGTA attacker strategy exploration rule

Require: $U_a, \kappa, \gamma, \alpha, g(), g'(), (\sigma_{d,0}, \ldots, \sigma_{d,t-1})$

$\overline{\sigma}_d \leftarrow \left(\sum_{\psi=0}^{t-1} \gamma^{t-1-\psi}\right)^{-1} \sum_{\psi=0}^{t-1} \gamma^{t-1-\psi} \sigma_{d,\psi}$

$k \leftarrow 0$

$\delta^k_{a,t} \leftarrow g(U_a, \sigma_{d,t-1})$

while $k < \kappa$ **do**

  $k \leftarrow k + 1$

  $\delta^k_{a,t} \leftarrow g'(U_a, \overline{\sigma}_d, \delta^{k-1}_{a,t})$

$\delta^*_{a,t} \leftarrow \arg\max_{\delta^{k'}_{a,t}} \alpha U_a(\delta^{k'}_{a,t}, \sigma_{d,t-1}) + (1-\alpha) U_a(\delta^{k'}_{a,t}, \overline{\sigma}_d)$,

s.t. $U_a(\delta^{k'}_{a,t}, \sigma_{d,t-1}) > U_a(\sigma_{a,t-1}, \sigma_{d,t-1})$ [or $\emptyset$ if none].

return $\delta^*_{a,t}$

Depending on its parameters, HADO-EGTA allows considerable choice of fine-tuning opponents. HADO-EGTA can fine-tune against a uniform mixed strategy over previous opponent equilibrium strategies, as in fictitious play, if $\gamma = 1$. Or HADO-EGTA can fine-tune against only the current equilibrium opponent, as in double oracle, if $\gamma \approx 0$. Adjusting α allows a user to make HADO-EGTA favor beneficial deviations against the current opponent, or exploitation of previous opponents. HADO-EGTA can select for maximal payoff against the current opponent if $\alpha = 1$, still reaping the benefits of multiple interim strategy options $\delta^k_{a,t}$. Or HADO-EGTA can be focused on exploiting previous opponents, subject to producing a beneficial deviation against the current one, if $\alpha = 0$.

5.5.2 HADO-EGTA implementation. We pre-train DNNs for 700,000 steps against the current equilibrium opponent (1,000,000 for the defender in r_{30}), and fine-tune for 400,000, recording $\kappa = 4$ interim strategies to select from. More specifically, the interim strategies are recorded at the end of pre-training, and after $\frac{1}{3}$, $\frac{2}{3}$, and all fine-tuning training steps are complete. (Mean performance is measured for the interim networks via independent sampling of payoffs.) We linearly anneal ϵ from 1.0 to 0.03 over the first half of pre-training steps, then hold it at 0.03. Similarly, ϵ is annealed from 0.3 to 0.03 over the first half of fine-tuning steps and then is held at 0.03.

6 RESULTS

6.1 Effectiveness of deep RL oracles

We find that DQN consistently learns beneficial deviations from the current Nash equilibrium over many rounds of DO-EGTA, only at times failing to generate beneficial deviations after several rounds of training have already been completed. In all, we conducted 2 runs of HADO-EGTA and 3 of DO-EGTA in each environment (s_{29} or r_{30}). All of these runs have converged.

Figure 5 shows the expected payoffs for attacker and defender equilibrium strategies and learned deviations, in an example run of DO-EGTA in game r_{30}. The defender fails to deviate beneficially in rounds (16, 19, 23), while the attacker fails in seven of the later rounds. The largest deviation gains tend to occur in earlier training rounds. In this run, the process converges after round 23, when neither training process learns a beneficial deviation.

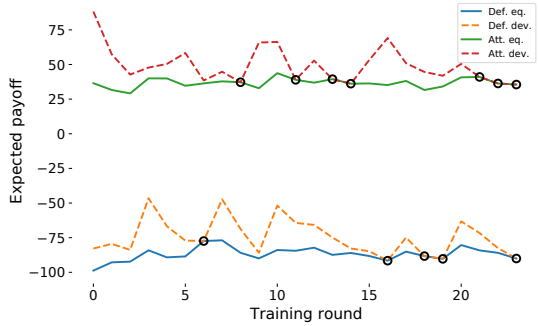

Fig. 5. Expected payoff of attacker and defender, before and after learning a new DNN, in each training round. Circles indicate rounds with no beneficial deviation. Results are for game r_{30}, in a single example run of DO-EGTA.

Trends in deviation gains are easier to see when we plot the difference in each round between the equilibrium payoff and the payoff of the deviating strategy, as shown for game r_{30} in Figure 6(a). (Note that where no beneficial deviation was learned, the gain is shown as zero.) As shown, deviation gains tend to become smaller as more training rounds are completed, but with large variations from one round to the next. One clear result in Figure 6(a) is that the trend in deviation gains from deep RL is not smooth or monotonic. Deviation gains are modest in most rounds, but with occasional very large gains, even (but infrequently) in later rounds of training.

Figure 6(b) shows the defender learning curves of an example run of DO-EGTA, over 21 rounds in game r_{30}. Note that most curves with large gains are from early rounds, which are mapped to purple colors. Even in later training rounds (which are mapped to yellow), the learner steadily improves in each curve, especially early in training, before leveling off at a payoff above the equilibrium payoff, except in the final round.

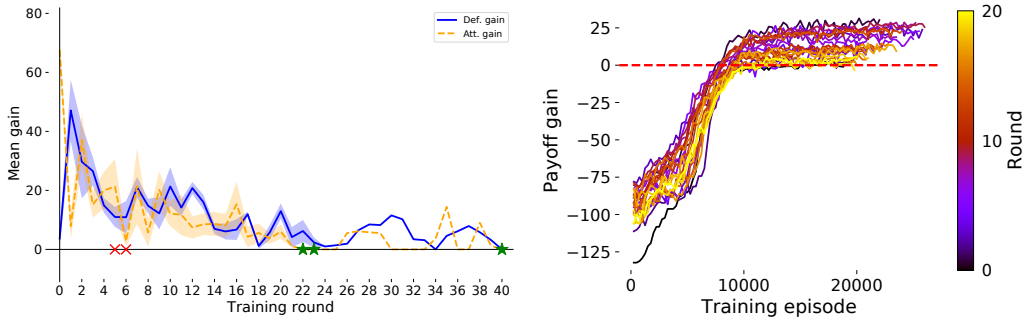

Fig. 6. (a) Payoff gain of new DNNs w.r.t. to current equilibrium. Results are means over all DO-EGTA runs for r_{30}, with standard error of the mean (SEM). (SEM equals zero for training rounds with only one run remaining.) Green stars show when runs converged, red crosses when they were stopped early. (b) Defender's learning curve for each round of DO-EGTA training, in an example run for game r_{30}. Curves are shown as gain relative to previous equilibrium. Color map goes from purple in early rounds to yellow in later ones.

6.2 HADO-EGTA improvement over DO-EGTA

We compare the performance of HADO-EGTA and DO-EGTA by evaluating the strategic stability of each beneficially deviating DNN produced by a run of either method. Strategy quality is always relative to other-agent strategy choices; we adopt the approach of evaluation with respect to play against a Nash equilibrium [3, 16]. Specifically, we take the union over all beneficially deviating DNNs produced by all runs of HADO-EGTA and DO-EGTA in an environment (e.g., r_{30} or s_{29}), along with all the heuristic strategies, which we term the *combined game*. We then measure the quality of any strategy as its regret with respect to the Nash equilibria of this combined game. For example, to evaluate an attacker DNN strategy ϕ_{att} against equilibrium profile σ^*, we compute $\rho(\phi_{att}, \sigma^*)$. The combined games contain many more strategies than the final games from individual runs: in setting s_{29}, 182 attacker and 348 defender strategies, vs. 30–50 and 82–122 in individual runs' final games; in setting r_{30}, 109 attacker and 182 defender strategies, vs. 23–35 and 67–89.

As shown in Figure 7, HADO-EGTA produces DNNs with lower regret on average, with respect to Nash equilibria of the combined game, compared to DO-EGTA. Figure 7 shows results from the 2 runs of HADO-EGTA and 3 of DO-EGTA carried out in each setting, r_{30} or s_{29}. In the combined games, we found 4 Nash equilibria in setting r_{30} and 1 in s_{29}. Each HADO-EGTA column has 2 scatter plot points per equilibrium, each DO-EGTA column 3 per equilibrium, corresponding to the distinct runs. Across all equilibria, almost all HADO-EGTA runs produced mean regret below any corresponding DO-EGTA run. The magnitude of the HADO-EGTA improvement ranges from roughly a factor of 0.25 to 0.5 for a given condition, taking the mean over results for attacker or defender, in setting r_{30} or s_{29}. Note that the combined games we analyze here for r_{30} and s_{29} contain all DNN strategies. All runs included in this analysis have converged, over up to 109 rounds.

As shown in Figure 8, it appears that DNNs trained by both DO-EGTA and HADO-EGTA tend to have lower regret, with respect to the combined game, in later training rounds. However, HADO-EGTA produces DNNs whose regrets decrease more rapidly during the initial training rounds, and remain lower late in training.

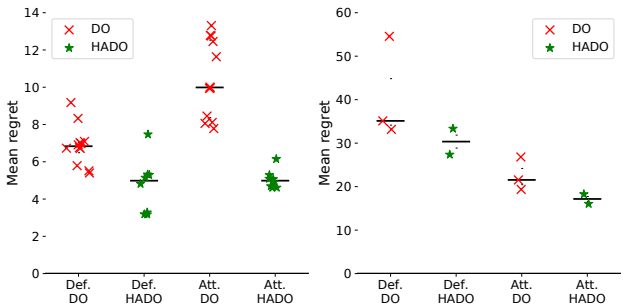

Fig. 7. Mean regret of DNNs trained, w.r.t. Nash equilibria of the combined game. Left: r_{30}, right: s_{29}. Each marker is based on one run, taking regret w.r.t. to one Nash equilibrium. Bars show median result for each condition.

A distinct approach we take to comparing HADO-EGTA and DO-EGTA performance is to conduct a tournament between the final equilibrium mixed strategies from all converged runs. In the tournament, for a given environment (e.g., r_{30} or s_{29}), the final equilibrium mixed strategy of each training run of HADO-EGTA (DO-EGTA) will meet the final equilibrium mixed strategy of the opposing agent type from every run of DO-EGTA (HADO-EGTA). For any pair of converged runs, where one is HADO-EGTA and the other DO-EGTA, a run is considered the clear winner if it

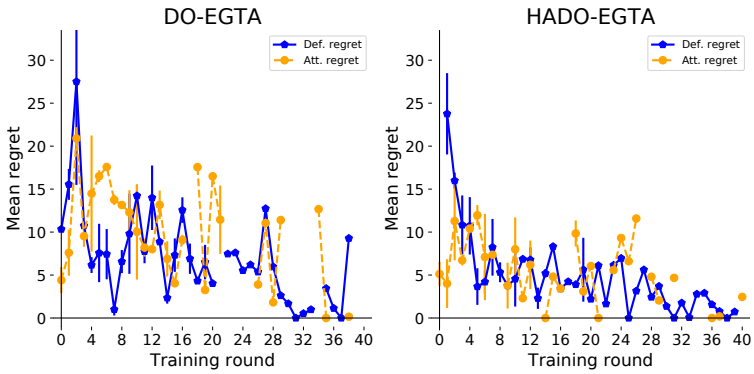

Fig. 8. Mean over training runs of DO-EGTA or HADO-EGTA, of the regret of each round's DNNs, with respect to a Nash equilibrium of the combined game for r_{30}. Gaps appear where no beneficially deviating network was produced. Error bars show standard error of the mean. (SEM equals zero for training rounds with only one run producing a beneficial deviation.)

outperforms the other run in each of 8 payoff comparisons. Specifically, for some run A to win over run B: both attacker and defender of each run must achieve lower payoff against A than against B; and both attacker and defender from A must get higher payoff than B against either opposing run.

For the tournament approach to analysis, we have six pairs of converged runs, comprising one run each of DO-EGTA and HADO-EGTA in setting s_{29}, as well as all three DO-EGTA and two HADO-EGTA runs in setting r_{30}. The HADO-EGTA run is the clear winner in setting s_{29} and in one of the six pairs from setting r_{30}, with all 8 payoff comparisons being in favor of the HADO-EGTA mixed strategies. In the other three matched pairs from setting r_{30}, HADO-EGTA is superior in only $(3, 5, 6, 7, 7)$ of 8 payoff comparisons respectively, meaning neither equilibrium is clearly stronger. In all, the tournament results constitute modest evidence that HADO-EGTA produces stronger mixed strategies upon convergence than DO-EGTA.

6.2.1 Analyzing causes of HADO-EGTA gains. Multiple methodological differences between HADO-EGTA and DO-EGTA could potentially explain the gains of HADO-EGTA:

(1) Each round of HADO-EGTA in our study adds 400,000 training steps in the fine-tuning phase.
(2) Each round of HADO-EGTA in these experiments selects the best DNN among 4 intermediate training outputs, while DO-EGTA must use the DNN from the end of training.
(3) HADO-EGTA fine-tunes each DNN against opponents from equilibria of previous rounds, while DO-EGTA trains only against the current round's equilibrium opponent.

It is important to consider all possible explanations to elucidate why HADO-EGTA is successful [22].

To evaluate Item (1), the effect of added training steps, we check if the expected payoff of the DNN being trained increases substantially in the final steps of DO-EGTA training. We found that during the last 0.1 fraction of training steps in DO-EGTA, only modest payoff gains of about 0.5 to 3.6 were produced on average, with only 0.57 to 0.64 fraction of changes being positive, depending on the setting.

To investigate the impact of Item (2), selection among interim DNNs by HADO-EGTA, we measure how much worse each DNN produced by DO-EGTA performs, relative to the best interim payoff achieved during its training. We find that the mean difference between the final DNN payoff of a DO-EGTA training run and the highest payoff during training tends to be small, between 3.4 and 8.7. We also check how often HADO-EGTA selects DNNs other than the final one produced

during a training round. HADO-EGTA usually selects the final fine-tuned DNN, and when it does not, the gain over the final DNN in payoff against the current equilibrium opponent is modest on average, between 1.5 and 4.6.

These analyses provide some evidence that neither Item (1), the extended training period, nor Item (2), the DNN selection component, can fully account for the improved performance of HADO-EGTA relative to DO-EGTA. This leaves the remaining explanation, Item (3), that it is fine-tuning against previous rounds' equilibrium opponents that largely accounts for the improved performance of HADO-EGTA relative to DO-EGTA.

6.3 Evolution of equilibrium mixed strategies

We can see a more detailed picture of how a player's equilibrium behavior changes from round to round in a heat map, plotting the weight of each round's DNN in each equilibrium (which might be 0). In Figure 9, for each of 22 training rounds of an example run of DO-EGTA in game r_{30}, we plot the weight in our Nash equilibrium of each defender DNN strategy learned so far, as well as of all heuristic strategies combined (shown as column 0). By definition, the full weight of 1 will be on heuristic strategies in round 0. After a later round n, weight of 1 can be spread across any DNN up to n.

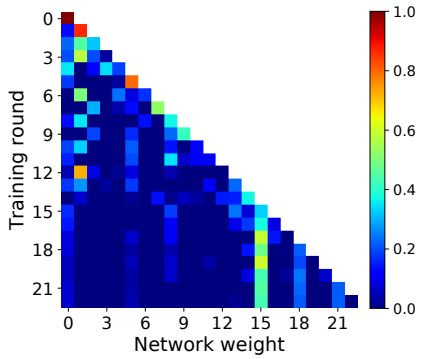

Fig. 9. Heat map of defender mixed strategies by training round, for an example DO-EGTA run in game r_{30}.

If the agent consistently learned better strategies in each round, almost all weight would appear along the main diagonal of the heat map, where the newest DQN strategies are shown. But if learning plateaued, there would be little weight on the main diagonal beyond a certain vertical line, after which the weight would remain on the most successful strategies from early rounds. Figure 9 seems to show the defender in game r_{30} learned successfully before round 15, placing much weight on the newest DNNs, although with some on heuristic strategies as well. After round 16, the defender began using the epoch 15 DNN heavily at equilibrium, and not placing as much weight on new DNNs. This pattern is typical of DO-EGTA training runs.

6.4 Performance with uninformed initial strategies

Starting with effective heuristic strategies for a complex game, our method has demonstrated an ability to learn significantly superior strategies. One natural question is whether the initialization with those strategies was pivotal to the success. To answer this question, we re-ran HADO-EGTA

on r_{30}, initialized with attacker and defender strategies that select actions uniformly at random. Preliminary results suggest that our method actually performs well even when started with random strategies. Specifically, we found that after a handful of rounds, the quality of learned attacker and defender reached a roughly comparable level with those learned in the same round of the heuristic-driven HADO-EGTA runs. This result contrasts with the experience of Wang et al. [46], who found iterative deep RL to be ineffective for their game when starting from random strategies, but successful when seeded with hand-coded heuristics. This disparity could be due to differences in the game setting, the methods, or a combination. We note that in our game, the random attacker performs relevant actions, and while distinctly suboptimal the performance is not terrible. The random defender, however, is quite inept.

6.5 Time requirements of DO-EGTA

We ran our experiments on Intel Xeon CPUs with 2.0–3.7 GHz clock speed. Our DNNs were trained with CPU only, using TensorFlow version 1.5. We believe training on GPU might have sped up the process only slightly, as the game simulator is CPU-bound.

Each round of DO-EGTA takes about 21–32 hours, and a run of 20–30 rounds requires about 25–40 days. HADO-EGTA requires more time per iteration than DO-EGTA, because it adds a fine-tuning step, and another step for evaluating the payoff of several learned DNNs to select the best. Each iteration of HADO-EGTA took from 21–51 hours. In addition, HADO-EGTA tends to require more rounds before converging.

7 CONCLUSION

The primary contribution of this study is the demonstration that EGTA iterated with deep RL can produce superior strategies for a complex dynamic game, compared to sophisticated heuristic strategies carefully designed and tuned (also with EGTA) in prior work. The game, a two-player non-zero-sum multi-stage interaction between attacker and defender on a Bayesian attack graph, features imperfect information and combinatorial action spaces. To deal with combinatorial actions, we introduced a greedy action-building approach that accounts for interdependencies among actions without blowing up the size of the DNN representing agent strategies. In two reasonably sized instances of the attack-graph game, we find out baseline method DO-EGTA consistently learns DNNs for both attacker and defender with stronger performance than the initial heuristics, both against equilibria of these heuristics and against the cumulative equilibria of generated strategies. These results were achieved in a rather generic DNN architecture, not specially structured or highly tuned to the action-graph domain.

To deal with learning instability due to DO's tendency to overfit to current equilibrium, we proposed a specific approach to balance responsiveness to current and historically encountered strategies. Experimental evidence suggests that our history-aware variant, HADO-EGTA, tends to produce better strategies earlier compared to DO-EGTA. Moreover, the DNNs ultimately returned on convergence by the history-aware method have lower regret, with respect to the combined game over all DNN and heuristic strategies generated by both methods. This, along with results from a tournament among final mixed strategies of converged runs, supports a positive conclusion about the benefits of this approach.

Our methods are computationally intensive, and as reported above the round count needed for convergence is unpredictable and may have high variance. Fortunately, the anytime performance of (HA)DO-EGTA appears reasonable after roughly the first half of the expected run count has completed. Further work will refine our methods and tackle some computational bottlenecks to improve the practicality of this iterated deep RL approach to complex game solving.

ACKNOWLEDGMENTS

This work was supported by funding from the US Army Research Office (MURI grant W911NF-13-1-0421) and from DARPA SI3-CMD.

REFERENCES

[1] Amenaza. 2018. SecurITree. http://www.amenaza.com/
[2] David Balduzzi, Sébastien Racanière, James Martens, Jakob Foerster, Karl Tuyls, and Thore Graepel. 2018. The mechanics of *n*-player differentiable games. In *35th International Conference on Machine Learning*. 363–372.
[3] David Balduzzi, Karl Tuyls, Julien Pérolat, and Thore Graepel. 2018. Re-evaluating evaluation. In *32nd Conference on Neural Information Processing Systems*. 3272–3283.
[4] Trapit Bansal, Jakub Pachocki, Szymon Sidor, Ilya Sutskever, and Igor Mordatch. 2018. Emergent complexity via multi-agent competition. In *6th International Conference on Learning Representations*.
[5] Noam Brown and Tuomas Sandholm. 2018. Superhuman AI for heads-up no-limit poker: Libratus beats top professionals. *Science* 359 (2018), 418–424.
[6] Karel Durkota, Viliam Lisỳ, Branislav Bošanskỳ, and Christopher Kiekintveld. 2015. Approximate solutions for attack graph games with imperfect information. *6th Int'l Conference on Decision and Game Theory for Security*, 228–249.
[7] Vlad Firoiu, William F. Whitney, and Joshua B. Tenenbaum. 2017. Beating the world's best at Super Smash Bros. with deep reinforcement learning. *arXiv:1702.06230* (2017).
[8] Jakob Foerster, Richard Y. Chen, Maruan Al-Shedivat, Shimon Whiteson, Pieter Abbeel, and Igor Mordatch. 2018. Learning with opponent-learning awareness. In *17th International Conference on Autonomous Agents and Multiagent Systems*. 122–130.
[9] Paulina Grnarova, Kfir Y. Levy, Aurelien Lucchi, Thomas Hofmann, and Andreas Krause. 2018. An online learning approach to generative adversarial networks. In *6th International Conference on Learning Representations*.
[10] Ji He, Mari Ostendorf, Xiaodong He, Jianshu Chen, Jianfeng Gao, Lihong Li, and Li Deng. 2016. Deep reinforcement learning with a combinatorial action space for predicting popular Reddit threads. In *Conference on Empirical Methods in Natural Language Processing*. 1838–1848.
[11] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. 2018. Deep reinforcement learning that matters. *32nd AAAI Conference on Artificial Intelligence*, 3207–3214.
[12] Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. 2018. Rainbow: Combining improvements in deep reinforcement learning. In *32nd AAAI Conference on Artificial Intelligence*. 3215–3222.
[13] Kyle Ingols, Richard Lippmann, and Keith Piwowarski. 2006. Practical attack graph generation for network defense. In *Computer Security Applications Conference*. IEEE, 121–130.
[14] Isograph. 2018. AttackTree+. http://www.isograph-software.com/atpover.htm
[15] Manish Jain, Dmytro Korzhyk, Ondřej Vaněk, Vincent Conitzer, Michal Pěchouček, and Milind Tambe. 2011. A double oracle algorithm for zero-sum security games on graphs. *10th International Conference on Autonomous Agents and Multiagent Systems*, 327–334.
[16] Patrick R. Jordan, Christopher Kiekintveld, and Michael P. Wellman. 2007. Empirical game-theoretic analysis of the TAC supply chain game. In *6th Int'l Joint Conference on Autonomous Agents and Multiagent Systems*. 1188–1195.
[17] Patrick R. Jordan, L. Julian Schvartzman, and Michael P. Wellman. 2010. Strategy exploration in empirical games. In *9th International Conference on Autonomous Agents and Multiagent Systems*. 1131–1138.
[18] Nitin Kamra, Umang Gupta, Fei Fang, Yan Liu, and Milind Tambe. 2018. Policy learning for continuous space security games using neural networks. In *32nd AAAI Conference on Artificial Intelligence*. 1103–1112.
[19] Barbara Kordy, Ludovic Piètre-Cambacédès, and Patrick Schweitzer. 2014. DAG-based attack and defense modeling: Don't miss the forest for the attack trees. *Computer Science Review* 13 (2014), 1–38.
[20] Guillaume Lample and Devendra Singh Chaplot. 2017. Playing FPS games with deep reinforcement learning. In *31st AAAI Conference on Artificial Intelligence*. 2140–2146.
[21] Marc Lanctot, Vinicius Zambaldi, Audrunas Gruslys, Angeliki Lazaridou, Julien Perolat, David Silver, Thore Graepel, et al. 2017. A unified game-theoretic approach to multiagent reinforcement learning. *Advances in Neural Information Processing Systems*, 4190–4203.
[22] Zachary C. Lipton and Jacob Steinhardt. 2018. Troubling trends in machine learning scholarship. *arXiv:1807.03341* (2018).
[23] Richard D. McKelvey, Andrew M. McLennan, and Theodore L. Turocy. 2006. Gambit: Software tools for game theory.
[24] H. Brendan McMahan, Geoffrey J. Gordon, and Avrim Blum. 2003. Planning in the presence of cost functions controlled by an adversary. In *20th International Conference on Machine Learning*. 536–543.

[25] Erik Miehling, Mohammad Rasouli, and Demosthenis Teneketzis. 2015. Optimal defense policies for partially observable spreading processes on Bayesian attack graphs. In *Second ACM Workshop on Moving Target Defense*. 67–76.

[26] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous methods for deep reinforcement learning. In *33rd International Conference on Machine Learning*. 1928–1937.

[27] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. 2015. Human-level control through deep reinforcement learning. *Nature* 518 (2015), 529–533.

[28] Matej Moravčík, Martin Schmid, Neil Burch, Viliam Lisỳ, Dustin Morrill, Nolan Bard, Trevor Davis, Kevin Waugh, Michael Johanson, and Michael Bowling. 2017. DeepStack: Expert-level artificial intelligence in heads-up no-limit poker. *Science* 356, 6337 (2017), 508–513.

[29] Apurba K. Nandi, Hugh R. Medal, and Satish Vadlamani. 2016. Interdicting attack graphs to protect organizations from cyber attacks: A bi-level defender–attacker model. *Computers & Operations Research* 75 (2016), 118–131.

[30] Thanh H. Nguyen, Mason Wright, Michael P. Wellman, and Satinder Singh. 2018. Multi-stage attack graph security games: Heuristic strategies, with empirical game-theoretic analysis. *Security and Communication Networks* Article ID 2864873 (2018).

[31] Catherine Olsson, Surya Bhupatiraju, Tom Brown, Augustus Odena, and Ian Goodfellow. 2018. Skill rating for generative models. *arXiv:1808.04888* (2018).

[32] OpenAI. 2018. OpenAI Five. https://blog.openai.com/openai-five/

[33] Cynthia Phillips and Laura Painton Swiler. 1998. A graph-based system for network-vulnerability analysis. *Workshop on New Security Paradigms*, 71–79.

[34] Nayot Poolsappasit, Rinku Dewri, and Indrajit Ray. 2012. Dynamic security risk management using Bayesian attack graphs. *IEEE Transactions on Dependable and Secure Computing* 9, 1 (2012), 61–74.

[35] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. 2015. Trust region policy optimization. In *32nd International Conference on Machine Learning*. 1889–1897.

[36] L. Julian Schvartzman and Michael P. Wellman. 2009. Stronger CDA strategies through empirical game-theoretic analysis and reinforcement learning. In *8th Int'l Conference on Autonomous Agents and Multiagent Systems*. 249–256.

[37] Oleg Sheyner, Joshua Haines, Somesh Jha, Richard Lippmann, and Jeannette M Wing. 2002. Automated generation and analysis of attack graphs. In *IEEE Symposium on Security and Privacy*. IEEE, 273.

[38] 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. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature* 529 (2016), 484–489.

[39] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. 2018. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science* 362, 6419 (2018), 1140–1144.

[40] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. 2017. Mastering the game of Go without human knowledge. *Nature* 550, 7676 (2017), 354–359.

[41] Ardi Tampuu, Tambet Matiisen, Dorian Kodelja, Ilya Kuzovkin, Kristjan Korjus, Juhan Aru, Jaan Aru, and Raul Vicente. 2017. Multiagent cooperation and competition with deep reinforcement learning. *PLOS One* 12, 4 (2017).

[42] Anderson Tavares, Hector Azpurua, Amanda Santos, and Luiz Chaimowicz. 2016. Rock, paper, StarCraft: Strategy selection in real-time strategy games. In *12th AAAI Conf. Artificial Intelligence and Interactive Digital Entertainment*.

[43] Hado van Hasselt. 2010. Double Q-learning. In *Advances in Neural Information Processing Systems*. 2613–2621.

[44] Hado van Hasselt, Arthur Guez, and David Silver. 2016. Deep reinforcement learning with double Q-learning. In *30th AAAI Conference on Artificial Intelligence*. 2094–2100.

[45] Sridhar Venkatesan, Massimiliano Albanese, Ankit Shah, Rajesh Ganesan, and Sushil Jajodia. 2017. Detecting stealthy botnets in a resource-constrained environment using reinforcement learning. In *Fourth ACM Workshop on Moving Target Defense*. 75–85.

[46] Yufei Wang, Zheyuan Ryan Shi, Lantao Yu, Yi Wu, Rohit Singh, Lucas Joppa, and Fei Fang. 2019. Deep reinforcement learning for green security games with real-time information. In *33rd AAAI Conference on Artificial Intelligence*.

[47] Michael P. Wellman. 2016. Putting the agent in agent-based modeling. *Autonomous Agents and Multi-Agent Systems* 30 (2016), 1175–1189.

[48] Mason Wright and Michael P. Wellman. 2018. Evaluating the stability of non-adaptive trading in continuous double auctions. In *17th International Conference on Autonomous Agents and Multiagent Systems*. 614–622.

A DEEP RL HYPERPARAMETERS

r_{30} attacker inputs	241
r_{30} defender inputs	240
s_{29} attacker inputs	234
s_{29} defender inputs	232
hidden layers	2 FC layers, size 256
activations	ReLU per hidden layer
r_{30} attacker outputs	106
r_{30} defender outputs	31
s_{29} attacker outputs	103
s_{29} defender outputs	30
learning rate	5×10^{-5}
training discount factor	0.99
batch size	32
ϵ in DO-EGTA	linear 1.0 to 0.03 in first half, then 0.03
ϵ in HADO-EGTA pre.	linear 1.0 to 0.03 in first half, then 0.03
ϵ in HADO-EGTA fine.	linear 0.3 to 0.03 in first half, then 0.03
train steps in DO-EGTA	700k (1m for r_{30} defender)
train steps in HADO-EGTA pre.	700k (1m for r_{30} defender)
train steps in HADO-EGTA fine.	400k
κ in HADO-EGTA	4
γ in HADO-EGTA	0.7
α in HADO-EGTA	$\frac{2}{7}$
DDQN options	no prioritized replay; no param noise

Table 2. Deep RL hyperparameters