Learning to Play against Any Mixture of Opponents

Max Olan Smith
University of Michigan
mxsmith@umich.edu

Thomas Anthony
DeepMind
twa@google.com

Yongzhao Wang
University of Michigan
wangyzh@umich.edu

Michael P. Wellman
University of Michigan
wellman@umich.edu

Abstract

Intuitively, experience playing against one mixture of opponents in a given domain should be relevant for a different mixture in the same domain. We propose a transfer learning method, *Q-Mixing*, that starts by learning Q-values against each pure-strategy opponent. Then a Q-value for *any* distribution of opponent strategies is approximated by appropriately averaging the separately learned Q-values. From these components, we construct policies against all opponent mixtures without any further training. We empirically validate Q-Mixing in two environments: a simple grid-world soccer environment, and a complicated cyber-security game. We find that Q-Mixing is able to successfully transfer knowledge across any mixture of opponents. We next consider the use of observations during play to update the believed distribution of opponents. We introduce an opponent classifier—trained in parallel to Q-learning, using the same data—and use the classifier results to refine the mixing of Q-values. We find that Q-Mixing augmented with the opponent classifier function performs comparably, and with lower variance, than training directly against a mixed-strategy opponent.

1 Introduction

Reinforcement learning (RL) agents commonly interact in environments with other agents, whose behavior may be uncertain. For any particular probabilistic belief over the behavior of another agent (henceforth, *opponent*), we can learn to play with respect to that opponent distribution (henceforth, *mixture*), for example by training in simulation against opponents sampled from the mixture. If the mixture changes, ideally we would not have to train from scratch, but rather could *transfer* what we have learned to construct a policy to play against the new mixture.

Traditional RL algorithms include no mechanism to explicitly prepare for variability in opponent mixtures. Instead, current solutions either learn a new behavior, or update a previously learned behavior. In lieu of that, researchers designed methods for learning a single behavior successfully across a set of strategies [52], or quickly adapting in response to new strategies [21]. In this work, we explicitly tackle the problem of responding to new opponent mixtures without requiring further simulations for learning.

We propose a new algorithm, *Q-Mixing*, that effectively transfers learning across opponent mixtures. The algorithm is designed within a population-based learning regime [21, 28], where training is conducted against a known distribution of opponent policies. Q-Mixing initially learns value-based best responses (BR), represented as Q-functions, with respect to each of the opponent's pure strategies. It then transfers this knowledge against any given opponent mixture by weighting the Q-functions according to the probability that the opponent plays each of its pure strategies. The idea is illustrated in Figure 1. This calculation is an approximation of the Q-values against the mixed-strategy opponent,

with error due to misrepresenting the future belief of the opponent by the current belief. The result is an approximate BR policy against any mixture, constructed without additional training.

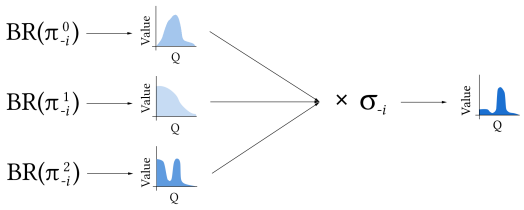

Figure 1: Q-Mixing concept. BR to each of the opponent's pure strategies π_{-i} are learned separately. The Q-value for a given opponent mixed strategy σ_{-i} is then derived by combining these components.

We experimentally validate our algorithm on: (1) a simple grid-world soccer game, and (2) a complicated cyber-security game. Our experiments show that Q-Mixing is effective in transferring learned responses across opponent mixtures.

We also address two potential issues with combining Q-functions according to a given mixture. The first is that the agent receives information *during play* that provides evidence about the opponent strategy. We address this by introducing an opponent classifier to predict which opponent we are playing for a particular state, and reweighting the Q-values to focus on the likely opponents. The second issue is that the complexity of the policy produced by Q-Mixing grows linearly with the support of the opponent's mixture. This can make simulation and decision making computationally expensive. We propose using policy distillation [40] as a method for compressing a complex Q-Mixing policy. Our experiments show that the compressed policy is able to recover the full performance of the Q-Mixing policy.

Key Contributions:

1. Theoretically relate (in idealized setting) the Q-value for an opponent mixture to Q-values for mixture components.
2. A new transfer learning algorithm, *Q-Mixing*, that uses this relationship to construct approximate policies against any given opponent mixture, without additional training.
3. Augmenting this algorithm with runtime opponent classification, to account for observations that can inform predictions of opponent policy during play.

2 Preliminaries

At any particular time $t \in \mathcal{T}$, an agent receives the environment's state $s^t \in \mathcal{S}$, or an observation $o^t \in \mathcal{O}$, a partial state. (Even if an environment's state is fully observable, the inclusion of other agents with uncertain behavior makes the overall system partially observable.) The agent then takes an action $a^t \in \mathcal{A}$ based on that observation, and receives a reward $r^t \in \mathbb{R}$. The agent's *policy* describes its behavior given each observation $\pi : \mathcal{O} \to \Delta(\mathcal{A})$. Actions are received by the environment, and a next observation is determined following the environment's transition dynamics $p : \mathcal{O} \times \mathcal{A} \to \mathcal{O}$.

The agent's goal is to maximize its reward over time. This quantity is called the *return*: $G^t = \sum_l^\infty \gamma^l r^{t+l}$, where γ is the discount factor weighting the importance of immediate rewards. Return is used to define the values of being in a given observation state:

$$V(o^t) = \mathbb{E}_\pi \left[\sum_{l=0}^{\infty} \gamma^l r(o^{t+l}, a^{t+l}) \right],$$

and taking an action given an observation:

$$Q(o^t, a^t) = r(o^t, a^t) + \gamma \mathbb{E}_{o^{t+1} \in \mathcal{O}} \left[V(o^{t+1}) \right].$$

For multiagent settings, we index the agents and distinguish the agent-specific components with subscripts. Agent i's policy is $\pi_i : \mathcal{O}_i \to \Delta(\mathcal{A}_i)$, and the policy for the opponent[1] is the negated index, $\pi_{-i} : \mathcal{O}_{-i} \to \Delta(\mathcal{A}_{-i})$. Boldface denotes elements that are joint across agents (e.g., joint-action $\mathbf{a}$).

[1] Our methods are defined here for environments with two agents. Extension to greater numbers while maintaining computational tractability is a topic for future work.

Agent i has a strategy set Π_i comprising the possible policies it can employ. Agent i may choose a single policy from Π_i to play as a *pure strategy*, or randomize with a *mixed strategy* $\sigma_i \in \Delta(\Pi_i)$. Note that the pure strategies π_i may themselves choose actions stochastically. For a mixed strategy, we denote the probability the agent plays a particular policy π_i as $\sigma_i(\pi_i)$. A *best response* (BR) to an opponent's strategy σ_{-i} is a policy with maximum return against σ_{-i}.

Agent i's prior belief about its opponent is represented by an opponent mixed-strategy, $\sigma^0_{-i} \equiv \sigma_{-i}$. The opponent plays the mixture by sampling a policy according to σ_{-i} at the start of the episode. The agent's updated belief at time t about the opponent policy faced is denoted σ^t_{-i}.

We introduce the term *Strategy Response Value* (SRV) to refer to the observation-value against a particular opponent's strategy.

Definition 1 (Strategic Response Value)**.** *An agent's* π_i strategic response value *is its expected return given an observation, when playing* π_i *against a specified opponent strategy:*

$$V_{\pi_i}(o_i^t \,|\, \sigma_{-i}^t) = \mathbb{E}_{\sigma^t_{-i}}\left[\sum_{\mathbf{a}} \boldsymbol{\pi}(a_i \,|\, o_i) \sum_{o_i', r_i} p(o_i', r_i \,|\, o_i, \mathbf{a}) \cdot \delta\right],$$

where $\delta \equiv r_i + \gamma V_{\pi_i}(o_i' \,|\, \sigma_{-i}^{t+1})$.

Let the optimal SRV *be*

$$V_i^*(o_i^t \,|\, \sigma_{-i}^t) = \max_{\pi_i} V_{\pi_i}(o_i^t \,|\, \sigma_{-i}^t).$$

From the SRV, we define the Strategic Response Q-Value (SRQV) for a particular opponent strategy.

Definition 2 (Strategic Response Q-Value)**.** *An agent's* π_i strategic response Q-value *is its expected return for an action given an observation, when playing* π_i *against a specified opponent strategy:*

$$Q_{\pi_i}(o_i^t, a_i^t \,|\, \sigma_{-i}^t) = \mathbb{E}_{\sigma^t_{-i}}\left[r_i^t\right] + \gamma \mathbb{E}_{o_i^{t+1}}\left[V_{\pi_i}(o_i^{t+1} \,|\, \sigma_{-i}^{t+1})\right],$$

where $r_i^t \equiv r_i(o_i^t, a_i^t, a_{-i}^t)$.

Let the optimal SRQV *be*

$$Q_i^*(o_i^t, a_i^t \,|\, \sigma_{-i}^t) = \max_{\pi_i} Q_{\pi_i}(o_i^t, a_i^t \,|\, \sigma_{-i}^t).$$

3 Q-Mixing

Our goal is to transfer the Q-learning effort across different opponent mixtures. We consider the scenario where we first learn against each opponent's pure strategy. From this, we construct a Q-function for a given distribution of opponents from the policies trained against each opponent's pure strategy.

3.1 Single-State Setting

Let us first consider a simplified setting with a single state. This is essentially a problem of bandit learning, where our opponent's strategy sets the rewards of each arm for an episode. Intuitively, our expected reward against a mixture of opponents is proportional to the payoff against each opponent weighted by their respective likelihood.

As shown in Theorem 1, weighting the component SRQV by the opponent's distribution supports a BR to that mixture. We call this relationship *Q-Mixing: Prior* and define it as follows:

$$Q_i^*(a_i \,|\, \sigma_{-i}) = \sum_{\pi_{-i} \in \Pi_{-i}} \sigma_{-i}(\pi_{-i}) Q_i^*(a_i \,|\, \pi_{-i}).$$

Theorem 1 (Single-State Q-Mixing)**.** *Let* $Q_i^*(\cdot \mid \pi_{-i})$*,* $\pi_{-i} \in \Pi_{-i}$*, denote the optimal strategic response Q-value against opponent policy* π_{-i}*. Then for any opponent mixture* $\sigma_{-i} \in \Delta(\Pi_{-i})$*, the optimal strategic response Q-value is given by*

$$Q_i^*(a_i \,|\, \sigma_{-i}) = \sum_{\pi_{-i} \in \Pi_{-i}} \sigma_{-i}(\pi_{-i}) Q_i^*(a_i \,|\, \pi_{-i}).$$

Proof. The definition of Q-value is as follows [46]:

$$Q_i^*(a_i) = \sum_{r_i} p(r_i \,|\, a_i) \cdot r_i.$$

In a multiagent system, the dynamics model p suppresses the complexity introduced by the other agents. We can unpack the dynamics model to account for the other agents as follows:

$$p(r_i \,|\, a_i) = \sum_{\pi_{-i}} \sum_{s_{-i}} \sum_{a_{-i}} \pi_{-i}(a_{-i}) \cdot p(r_i \,|\, \mathbf{a}).$$

We can then unpack the strategic response value as follows:

$$Q_i^*(a_i \,|\, \pi_{-i}) = \sum_{a_{-i}} \pi_{-i}(a_{-i}) \sum_{r_i} p(r_i \,|\, \mathbf{a}) \cdot r_i.$$

Now we can rearrange the expanded Q-value to explicitly account for the opponent's strategy. The independence assumption enables the following re-writing by letting us treat the opponent's mixed strategy as a constant condition.

$$\begin{aligned} Q_i^*(a_i \,|\, \sigma_{-i}) &= \sum_{r_i} \sum_{\pi_{-i}} \sigma_{-i}(\pi_{-i}) \sum_{a_{-i}} \pi_{-i}(a_{-i}) p(r_i \,|\, \mathbf{a}) \cdot r_i \\ &= \sum_{\pi_{-i}} \sigma_{-i}(\pi_{-i}) \sum_{a_{-i}} \pi_{-i}(a_{-i}) \sum_{r_i} p(r_i \,|\, \mathbf{a}) \cdot r_i \\ &= \sum_{\pi_{-i}} \sigma_{-i}(\pi_{-i}) Q_i^*(a_i \,|\, \pi_{-i}). \end{aligned}$$

□

3.2 Leveraging Information from the Past

Next we consider the RL setting where both agents are able to influence an evolving observation distribution. As a result of the joint effect of agents' actions on observations, the agents have an opportunity to gather information about their opponent during an episode. Methods in this setting need to (1) use information from the past to update its belief about the opponent, and (2) grapple with uncertainty about the future. To deal with the past, the agent maintains a belief σ_{-i}^t about the opponent's policy, $\psi_i : \mathcal{O}_i^{0:t} \to \Delta(\Pi_{-i})$. To bring Q-Mixing into this setting we need to quantify the agent's current belief about their opponent and their future uncertainty. Let ψ represent the agent's current belief about the opponent's policy. This quantity replaces the prior σ_{-i} that appears in the Q-Mixing: Prior definition.

During a run with an opponent's pure strategy drawn from a distribution, the actual observations experienced generally depend on the identity of this pure strategy. With this idea in mind, we propose an approximate version of Q-Mixing that accounts for past information. The approximation works by first predicting the relative likelihood of each opponent given the current observation. From this prediction, we weight the Q-value-based BRs against each opponent by their relative likelihood. This approximation does not consider any future uncertainty about the opponent (blue area in Figure 2).

Let the previously defined ψ be the *opponent policy classifier* (OPC), which predicts the opponent's policy. In this work we consider a simplified version of this function, that operates only on the agent's current observation $\psi_i : \mathcal{O}_i \to \Delta(\Pi_{-i})$. We then augment Q-Mixing to weight the importance of each BR as follows:

$$Q_{\pi_i}(o_i, a_i \,|\, \sigma_{-i}) = \sum_{\pi_{-i}} \psi_i(\pi_{-i} \,|\, o_i, \sigma_{-i}) Q_{\pi_i}(o_i, a_i \,|\, \pi_{-i}).$$

We refer to this quantity as *Q-Mixing*, or *Q-Mixing: X* where X describes ψ. By continually updating the opponent distribution during play, the adjusted Q-Mixing result better responds to the actual opponent.

An ancillary benefit of the opponent classifier is that poorly estimated Q-values tend to have their impact minimized. For example, if an observation occurs only against the second pure strategy, then the Q-value against the first pure strategy would not be trained well, and thus could distort the policy from Q-Mixing. These poorly trained cases correspond to unlikely opponents, and get reduced weighting in the version of Q-Mixing augmented by the classifier.

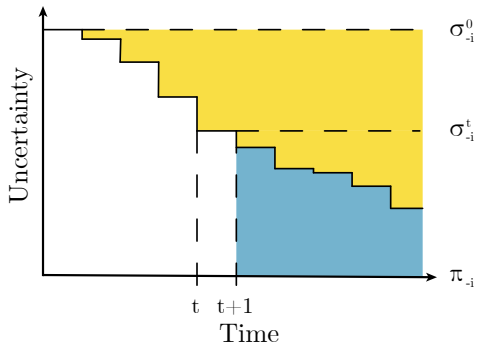

Figure 2: Opponent uncertainty over time. The yellow area represents uncertainty reduction as a result of updating belief about the distribution of the opponent. The blue area represents approximation error incurred by Q-Mixing.

3.3 Accounting for Future Uncertainty

To account for future uncertainty we must replace the Q-values against individual opponents. This can be done by expanding the Q-value definition into two parts: expected reward under the current belief in the opponent's policy, and our expected next observation value. The second term, successor observation value, recursively references a new opponent belief, accounting for changing uncertainty in the future. The extended formulation, Value Iteration with Q-Mixing (VIQM), is given by:

$$\delta = r_i(o_i^t, a_i^t \,|\, \pi_{-i}) + \gamma \mathbb{E}_{o_i^{t+1}}\left[V^*(o_i^{t+1} \,|\, \sigma_{-i})\right],$$
$$Q_i^*(o_i^t, a_i^t \,|\, \sigma_{-i}) = \sum_{\pi_{-i} \in \Pi_{-i}} \psi_i(\pi_{-i} \,|\, o_i^t, \sigma_{-i}) \cdot \delta.$$

If we assume that we have access to both a dynamics model and the observation distribution dependent on the opponent, then we can directly solve for this quantity through Value Iteration (Algorithm 1). These requirements are quite strong, essentially requiring perfect knowledge of the system with regards to all opponents. The additional step of Value Iteration also carries computational burden, as it requires iterating over the full observation and action spaces. Though these costs may render Value Iteration infeasible in practice, we provide the algorithm as way to ensure correctness in Q-values.

Algorithm 1: Value Iteration: Q-Mixing

Input: $\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \epsilon, \gamma$
$V_0(s \,|\, \sigma_{-i}) \leftarrow \sum_{\pi_{-i}} \sigma_{-i}(\pi_{-i}) Q(s, a \,|\, \pi_{-i})$
do
  $Q_t(s, a \,|\, \sigma_{-i}) \leftarrow \sum_{\pi_{-i}} \psi(\pi_{-i} \,|\, s, \sigma_{-i}) \sum_{s', r} \mathcal{T}(s', r \,|\, s, a, \pi_{-i})[r + \gamma V_{t-1}(s' \,|\, \sigma_{-i})]$
  $V_t(s \,|\, \sigma_{-i}) \leftarrow \max_a Q_t(s, a \,|\, \sigma_{-i})$
  $\pi_t(s \,|\, \sigma_{-i}) \leftarrow \text{argmax}_a Q_t(s, a \,|\, \sigma_{-i})$
while $\exists_{s \in \mathcal{S}} \; |V_t(s) - V_{t-1}(s)| > \epsilon$
Output: V_t, Q_t, π_t

4 Experiments

4.1 Grid-World Soccer

We first evaluate Q-Mixing on a simple grid-world soccer environment [29, 15]. This environment has small state and action spaces, allowing for inexpensive simulation. With this environment we pose the following questions:

1. Can VIQM obtain Q-values for mixed-strategy opponents?

2. Can Q-Mixing transfer Q-values across all of the opponent's mixed strategies?

The soccer environment is composed of a soccer field, two players, one ball, and two goals. The player's objective is to acquire the ball and score a goal while preventing the opponent from scoring on their own goal. The scorer receives $+1$ reward, and the opponent receives -1 reward. The state consists of the entire field including the locations of the players and ball. The players may move in any of the four cardinal directions or not move. Actions taken by the players are executed in a random order, and if the player possessing the ball moves last then the first player may take possession of the ball by colliding with them. A graphical example of the soccer environment can be seen in Figure 3.

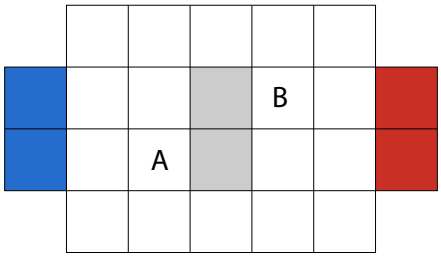

Figure 3: Grid-world soccer environment. The letters represent the respective players. The ball may spawn in either middle highlighted tile, and the player's goal is to score in the opposite net.

In our experiments, we learn policies for Player 1 using Double DQN [50]. The policies are deep neural networks with two hidden layers of size 50, and ReLU activation functions. Player 2 plays a strategy over five policies, using the same shape neural network as Player 1, generated using the double oracle algorithm [32, 28]. These policies are frozen for the duration of the experiments. Further details of the environment and experiments are in Section A.1.

4.1.1 Empirical Verification of Q-Mixing

We now turn to our first question: whether VIQM can obtain Q-values for mixed-strategy opponents. To answer this, we run the VIQM algorithm against a fixed opponent mixed strategy (Algorithm 1). We construct dynamics models for each opponent by considering the opponent's policy and the multiagent dynamics model as a single entity. Then we may approximate the relative observation-occupancy distribution by rolling out 30 episodes against each policy and estimating the distribution.

In our experiment an optimal policy was reached in fourteen iterations. The resulting policy best-responded to each individual opponent and the mixture. This empirically validates our first hypothesis.

4.1.2 Coverage of Opponent Strategy Space

Our second question is whether Q-Mixing can produce high-quality responses for any opponent mixture. Our evaluation of this question employs the same five opponent pure strategies as the previous experiment. We first trained a baseline, *BR(Uniform)*, directly against the uniform mixed-strategy opponent. The baseline was trained using 300000 simulation steps. The same hyperparameters were used to train against each of the opponent's pure strategies, with the simulation budget split equally. The Q-values trained respectively are used as the components for Q-Mixing, and an OPC is also trained from their respective replay buffers. This is repeated for five random seeds. We simulate the performance of each method against a representative selection of the entire opponent's mixed-strategy space. We select all opponent mixtures truncated to the tenths place, resulting in 860 strategy distributions.

Figure 4 shows Q-Mixing's performance across the opponent mixed-strategy space. Learning in this domain is fairly easy, so both methods are expected to win against almost every mixed-strategy opponent. Nevertheless, Q-Mixing generalizes across strategies better, albeit with slightly higher variance. While the improvement of Q-Mixing is incremental, we interpret this first evaluation as validating the promise of Q-Mixing for coverage across mixtures.

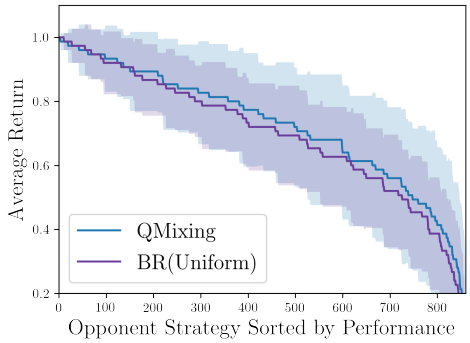

Figure 4: Q-Mixing's coverage of the opponent's strategy space. Each strategy is a mixture over the 5 opponents. The strategies are sorted per-method by the method's performance. Shaded regions represent the standard error over 5 seeds. BR(Mixture) is trained against the uniform mixture opponent. Both methods use the same number of experiences.

4.2 Cyber-Security Game

Not all environments are simple enough for an agent to acquire optimal Q-values, or have stateful environment information. In this section, we look at the performance of Q-Mixing under partial observation. We study the following research questions on a cyber-security game called Attack Graphs:

1. Can Q-Mixing work with non-optimal Q-values over observations from a complex environment?
2. Can using an opponent policy classifier to update the opponent distribution improve performance?
3. Can policy distillation be used to compress a Q-Mixing model, while preserving performance?

Attack Graphs model the interaction between a cyber-attacker and a cyber-defender [37]. The nodes of the graph represent security conditions (e.g., vulnerability on a server), and the edges correspond to exploits that activate a security condition, with some probability of success. The attacker's aim is to reach a set of goal nodes, which provides a large reward for the attacker and a loss for the defender. The defender receives a noisy observation of the graph influenced by false alarm(s) and false negative signal(s), while the attacker observes the true state of the graph. The players take actions trying to infiltrate or protect the nodes, while each action they take incurs a fixed cost.

A variety of cyber-security games on attack graphs have been studied [33, 9, 10, 34, 53]. In our experiments, we follow the game formulation of Wright et al. [53], which motivated the development of BR oracle based on Deep RL for this domain. The graph used is an Erdős-Rényi graph, with 30 nodes, and 100 edges. A visualization of the environment and more details are included in Section A.2. The policies in this environment are deep neural networks with two hidden layers of size 256 with tanh activation functions. They are trained using the Double DQN algorithm [50], and frozen for the duration of the experiments. We report results for the defender.

4.2.1 Empirical Verification of Q-Mixing

We experimentally evaluate Q-Mixing on the Attack-Graph environment, allowing us to confirm our algorithm's robustness to complex environments. First, a set of 37 opponent policies are generated through the double oracle algorithm [32]. Three pure strategies are randomly chosen from this set, and a BR is trained against each. Then a mixture of the policies is chosen (arbitrarily), using $\sigma_{-i} = [0.3, 0.5, 0.2]$. A baseline $\text{BR}(\sigma_{-i})$ is trained directly against the mixture. We evaluate both Q-Mixing and $\text{BR}(\sigma_{-i})$ against each of the three pure-strategy opponents as well as the mixture.

A summary of the performance against each opponent strategy is presented in Table 1. We observe that Q-Mixing: Prior performs worse than $\text{BR}(\sigma_{-i})$. Neither Q-Mixing: Prior nor $\text{BR}(\sigma_{-i})$ perform well against the pure-strategy opponents, when compared to their respective BRs. To further understand the relationship between these two methods, we plot $\text{BR}(\sigma_{-i})$'s training curve in Figure 5. From this graph we can see that Q-Mixing: Prior achieves a performance equivalent to $\text{BR}(\sigma_{-i})$, after $\text{BR}(\sigma_{-i})$

Table 1: Performance transfer across opponent strategies in a cyber-security game. The defender plays the policy method, and the attacker follows the opponent strategy. Total reward received and 95% confidence interval over five random seeds are reported.

Policy	Opponent Strategy			
	π^0_{-i}	π^1_{-i}	π^2_{-i}	σ_{-i}
BR(π^0_{-i})	$\mathbf{-122.3 \pm 4.84}$	-175.8 ± 09.79	-122.4 ± 9.36	-140.1 ± 22.07
BR(π^1_{-i})	-186.2 ± 3.70	$\mathbf{-41.0 \pm 05.65}$	-173.1 ± 8.18	$\mathbf{-133.4 \pm 57.50}$
BR(π^2_{-i})	-164.2 ± 5.90	-213.7 ± 15.53	$\mathbf{-36.6 \pm 3.42}$	$\mathbf{-138.2 \pm 65.38}$
BR(σ_{-i})	-151.7 ± 1.01	-64.4 ± 02.61	-88.5 ± 6.78	$\mathbf{-101.6 \pm 32.26}$
Q-Mixing: Prior	-163.1 ± 5.01	-90.8 ± 24.39	-144.3 ± 8.48	-132.8 ± 26.84
Q-Mixing: Obs. Freq.	-163.3 ± 5.73	-79.6 ± 07.40	-144.2 ± 2.52	-117.6 ± 03.35
Q-Mixing: Opp. Classifier	$\mathbf{-120.0 \pm 0.93}$	$\mathbf{-33.0 \pm 01.09}$	-145.4 ± 1.39	$\mathbf{-99.1 \pm 00.76}$

has finished exploring and has begun to converge. If we are in a training regime where we may already have access to BRs, then this would enable free policy construction.

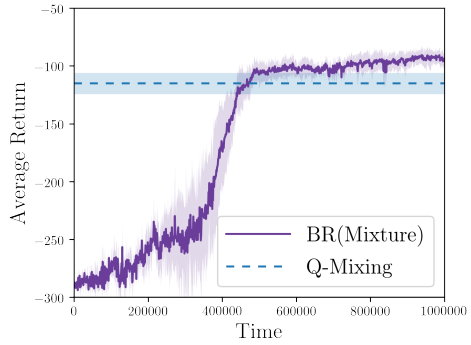

Figure 5: Learning curve of the BR policy trained against the mixed-strategy opponent. This is compared to the performance of Q-Mixing: Prior.

4.2.2 Opponent Classification

This brings us to our next research question: how can we improve performance against individual opponent policies? We hypothesize that if a player is able to correctly identify which opponent pure strategy is being played, they can weight the importance of the Q-values learned against that pure strategy higher.

To verify this hypothesis, we train an OPC using the replay buffers associated with each BR policy $\mathbb{B}(\pi_i)$. These are the same buffers that were used to train the BRs, and cost no additional compute to collect. This data is used to train an OPC that outputs a distribution over opponent pure strategies for each observation. We first look at *observation-frequency* as a baseline OPC, and then consider a learned OPC.

The observation-frequency OPC uses a relative observation occupancy as a measure of how likely the opponent is playing a particular pure strategy. We weight the importance of each pure strategy by the frequency the observation occurs in its corresponding replay-buffer, compared to the total frequency across all replay buffers. If a observation does not occur in any replay buffers, we equally weight each Q-value. The observation frequency calculation is computed as:

$$\psi_{\text{Observation Frequency}}(\pi^k_{-i}, o_i) = \frac{\text{count}(o_i, \mathbb{B}(\pi^k_{-i}))}{\sum_{\pi^j_{-i} \in \Pi_{-i}} \text{count}(o_i, \mathbb{B}(\pi^j_{-i}))},$$

where $\text{count}(o_i, \mathbb{B})$ returns the number of times o_i appears in the replay buffer $\mathbb{B}$.

The OPC version utilizes a deep neural network to classify each observation to the corresponding opponent pure strategy. The neural network has three hidden layers of sizes 200, 128, and 128

respectively with tanh activations. To train this classifier we take each experience and assign it a class label based off the opponent. The classifier is trained using cross-entropy loss.

As reported in Table 1, the observation-frequency OPC improves over Q-Mixing: Prior, within noise, but dramatically reduces the variance. The opponent-classifier version further improves over observation-frequency, bringing the performance on par with BR(σ_{-i}) with much lower variance. In particular, the OPC performs comparably against the first two opponent pure strategies with their respective best-responders. The performance against the third pure strategy is poor; however, this is the least likely opponent, and this may have enabled the stronger performance on the other opponents. This indicates that the opponent-classifier is able to correctly identify which pure strategy their opponent is playing, and weight it high enough to not introduce noise from the other component Q-values. This evidence supports our hypothesis that using an opponent-classifier to weight the opponent distribution can provide performance improvements against mixed-strategy opponents.

4.2.3 Policy Distillation

In Q-Mixing we need to compute Q-values for each of the opponent's pure strategies. This can be a limiting factor in parametric policies, like deep neural networks, where our policy's complexity grows linearly in the size of the support of the opponent's mixture. This can become unwieldy in both memory and computation. To remedy these issues, we propose using policy distillation to compress a Q-Mixing policy into a smaller parameters space [19, 40].

In the policy distillation framework, a larger neural network referred to as the "teacher" is used as a training target for a smaller neural network called the "student". In our experiment the Q-Mixing policy is the teacher to a student neural network that is the size of a single best-response policy. The student is trained in a supervised learning framework, where the dataset is the concatenated replay buffers from training pure-strategy best-responses. The is the same dataset that was used in opponent classifying, that was notably generated without running any additional simulations. A batch of data is sampled from the replay-buffer and the student predicts Q^S the teacher's Q^T Q-values for each action. The student is then trained to minimize the KL-divergence between the predicted Q-values and the teacher's true Q-values. There is a small wrinkle, the policies produce Q-values, and KL-divergence is a metric over probability distributions. To make this loss function compatible, the Q-values are transformed into a probability distribution by softmax with temperature τ. The temperature parameter allows us to control the softness of the maximum operator. A high temperature produces actions that have a near-uniform probability, and as temperature is lowered the distribution concentrates weight on the highest Q-Values [46]. The benefit of a higher temperature is that more information can be passed from the teacher to the student about each state. The full policy distillation loss is:

$$\mathcal{L}_{\text{Distill}} = \sum_{i}^{|D|} \text{softmax}(\frac{Q^T}{\tau}) \ln(\frac{\text{softmax}(Q^T/\tau)}{\text{softmax}(Q^S/\tau)})$$

, where D is the dataset of concatenated replay buffers. Additional training details are descibed in Section A.4.

The learning curve of the student is reported in Figure 6. We found that the student was able to recover the performance of Q-Mixing: OPC, albeit with higher variance, within five minutes. This study did not look at tuning the student, and further improvements with the same methodology may be possible. This result confirms our hypothesis that policy distillation is able to effectively compress a policy derived by Q-Mixing.

5 Related Work

5.1 Multiagent Learning

The most relevant line of work in multiagent learning studies the interplay between centralized learning and decentralized execution [47, 26]. In this regime, agents are able to train with additional information about the other agents that would not be available during evaluation (e.g., the other agent's state [38], actions [7], or a coordination signal [15]). The key question then becomes: how to create a policy that can be evaluated without the additional information? A popular approach is to decompose the joint-action value into independent Q-values for each agent [16, 18, 45, 38, 31]. Another direction

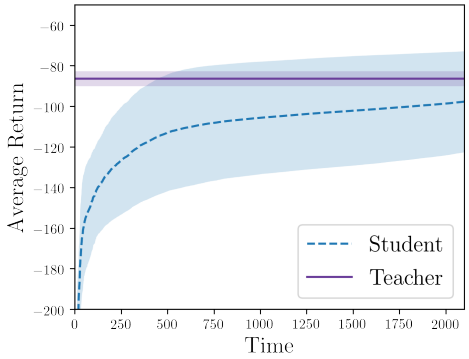

Figure 6: Simulation performance of Q-Mixing: OPC (teacher) being distilled into a smaller network (student). Results are averaged over 5 random seeds.

looks at learning a centralized critic, which can train independent agent policies [17, 30, 13]. Some work has proposed constructing meta-data about the agent's current policies as a way to reduce the learning instability present in environments where other agents' policies are changing [11, 35].

A set of assumptions that can be made is that all players have fixed strategy-sets. Under these assumptions agents could maintain more sophisticated beliefs about their opponent [55], and extend this to recursive-reasoning procedures [54]. Extending these works to fit the Q-Mixing problem statement is tangential to this study; however, may be a promising direction for improving the quality of the OPC. One more potential extension of the OPC is to consider alternative goals. Instead of focusing exclusively on predicting the opponent, in safety critical situations an agent will want to consider an objective that accounts for inaccurate prediction of their opponent. The Restricted Nash Response Johanson et al. [22] encapsulates this measure by balancing maximal performance if the prediction is correct with reasonable performance for inaccurate predictions.

Instead of building or using complete models of opponents, one may use implicit representation of their opponents. By choosing to build an explicit model of their opponent they circumvent needing a large amount of data to reconstruct the opponent policy. An additional benefit is that there are less likely to be errors in the model that need to be overcome. He et al. [18] proposes DRON which uses a learned latent action prediction of the opponent as conditioning information to the policy (in a similar nature to the opponent-actions in the joint-action value area). They also show another version DRON which uses a Mixture-of-Experts [20] operation to marginalize over the possible opponent behaviors. Bard et al. [3] proposes implicitly modelling opponents through the payoffs received from playing against a portfolio of the agent's policies.

Most multiagent learning work focuses on simultaneous learning of many agents, where there is not a distribution of opponents. This difference in methods can have strong influences in the final learned policies. For example, an agent trained concurrently with another agent may be unable to coordinate with a different agent [4]. Another potential problem is that each agent now faces a dynamic learning problem, where they must learn a moving target (the other agent's policy) [12, 49].

5.2 Multi-Task Learning

Multiagent learning is analogous to multi-task learning. In this reconstruction, each strategy/policy is analogous to solving a different task. And the opponent's strategy would be the distribution over tasks. Similar analogies to tasks can be made with objectives, goals, contexts, etc. [24, 39].

The multi-task community has roughly separated learnable knowledge into two categories [44]. *Task relevant* knowledge pertains to a particular task [23, 51]; meanwhile, *domain relevant* knowledge is common across all tasks [6, 14, 25]. Work has been done that bridges the gap between these settings; for example, knowledge about a task could be a curriculum to utilize over tasks [8]. In task relevant learning, a leading method is to identify state information that is irrelevant to decision making, and abstract it away [23, 51]. Our work falls into the same task relevant category, where we are interested in learning responses to particular opponent policies. What differentiates our work with the previous work is that we learn Q-values for each task independently, and do not ignore any information.

Progressively growing neural networks is another similar line of work [41], focused on a stream of new tasks. Schwarz et al. [42] also found that network growth could be handled with policy distillation.

5.3 Transfer Learning

Transfer learning is the study of reusing knowledge to learn new tasks/domains/policies. Within transfer learning, we look at either how knowledge is transferred, or what kind of knowledge is transferred. How to transfer knowledge has previously been studied in main two flavors [36, 27]. The *representation transfer* direction looks at how to abstract away general characteristics about the task that are likely to apply to later problems. Ammar et al. [1] present an algorithm where an agent collects a shared general set of knowledge that can be used for each particular task. The second direction directly transfers parameters across tasks; appropriately called *parameter transfer*. Taylor et al. [48] show how policies can be reused by creating a projection across different tasks' state and action spaces.

In the literature, transferring knowledge about the opponent's strategy is considered intra-agent transfer [43]. The focus of this area is on *adapting to other agents*. One line of work in this area focuses on ad hoc teamwork, where an agent must learn to quickly interact with new teammates [5, 4]. The main approach relies on already having a set of policies available, and learning to select which policy will work best with the new team [5]. Another work proposes learning features that are independent of the game, which can either be qualities general to all games or strategies [2]. Our study differs from these in its focus on the opponent's policies as the source of information to transfer.

6 Conclusions

In this study we propose Q-Mixing, an algorithm for transferring knowledge across distributions of opponents. We show how Q-Mixing relies on the theoretical relationship between an agent's action-values, and the strategy employed by the other agents. An empirical confirmation of the theory is first made on the grid world soccer environment. In this environment we show how experience against pure strategies can transfer onto mixed-strategy opponents. Moreover, we show that this transfer is able to cover the space of mixed strategies with no additional computation.

Next we tested our algorithm's robustness on a complicated cyber-security game. In this environment we first show the benefit of maintaining a belief about the opponent's policy. This belief is then used to update weights of their respective BR Q-values. Next we address the concern that a Q-Mixing policy may become too large or computationally expensive to use. To ease this concern we demonstrate that policy distillation can be used to compress a Q-Mixing policy into a much smaller parameter space.

References

[1] Haitham Bou Ammar, Eric Eaton, José Marcio Luna, and Paul Ruvolo. Autonomous cross-domain knowledge transfer in lifelong policy gradient reinforcement learning. In *24th International Conference on Artificial Intelligence*, IJCAI, pages 3345–3349, 2015.

[2] Bikramjit Banerjee and Peter Stone. General game learning using knowledge transfer. In *20th International Joint Conference on Artificial Intelligence*, IJCAI, pages 672–677, 2007.

[3] Nolan Bard, Michael Johanson, Neil Burch, and Michael Bowling. Online implicit agent modelling. In *12th International Conference on Autonomous Agents and Multiagent Systems*, pages 255–262, 2013.

[4] Nolan Bard, Jakob N. Foerster, Sarath Chandar, Neil Burch, Marc Lanctot, H. Francis Song, Emilio Parisotto, Vincent Dumoulin, Subhodeep Moitra, Edward Hughes, Iain Dunning, Shibl Mourad, Hugo Larochelle, Marc G. Bellemare, and Michael Bowling. The Hanabi challenge: A new frontier for AI research. *Artificial Intelligence*, 280, 2020.

[5] Samuel Barrett and Peter Stone. Cooperating with unknown teammates in complex domains: A robot soccer case study of ad hoc teamwork. In *29th AAAI Conference on Artificial Intelligence*, AAAI, pages 2010–2016, 2015.

[6] Rich Caruana. Multitask learning. *Machine Learing*, 28(1):41–75, 1997.

[7] Caroline Claus and Craig Boutilier. The dynamics of reinforcement learning in cooperative multiagent systems. In *15th National Conference on Artificial Intelligence*, AAAI, pages 746–752, 1998.

[8] Wojciech Czarnecki, Siddhant Jayakumar, Max Jaderberg, Leonard Hasenclever, Yee Whye Teh, Nicolas Heess, Simon Osindero, and Razvan Pascanu. Mix & match agent curricula for reinforcement learning. In *35th International Conference on Machine Learning*, ICML, pages 1087–1095, 2018.

[9] Karel Durkota, Viliam Lisý, Branislav Bošanský, and Christopher Kiekintveld. Optimal network security hardening using attack graph games. In *24th International Conference on Artificial Intelligence*, IJCAI, pages 526–532, 2015.

[10] Karel Durkota, Viliam Lisý, Christopher Kiekintveld, Branislav Bošanský, and Michal Pechoucek. Case studies of network defense with attack graph games. *IEEE Intelligent Systems*, 31:24–30, 2016.

[11] Jakob Foerster, Nantas Nardelli, Gregory Farquhar, Triantafyllos Afouras, Philip H. S. Torr, Pushmeet Kohli, and Shimon Whiteson. Stabilising experience replay for deep multi-agent reinforcement learning. In *34th International Conference on Machine Learning*, ICML, pages 1146–1155, 2017.

[12] Jakob Foerster, Richard Y. Chen, Maruan Al-Shedivat, Shimon Whiteson, Pieter Abbeel, and Igor Mordatch. Learning with opponent-learning awareness. In *17th International Conference on Autonomous Agents and Multiagent Systems*, AAMAS, pages 122–130, 2018.

[13] Jakob N. Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. In *32nd AAAI Conference on Artificial Intelligence*, AAAI, pages 2974–2982, 2018.

[14] David Foster and Peter Dayan. Structure in the space of value functions. *Machine Learning*, 49(2):325–346, 2002.

[15] Amy Greenwald and Keith Hall. Correlated-Q learning. In *20th International Conference on Machine Learning*, ICML, pages 242–249, 2003.

[16] Carlos Guestrin, Daphne Koller, and Ronald Parr. Multiagent planning with factored MDPs. In *14th International Conference on Neural Information Processing Systems*, NeurIPS, pages 1523–1530, 2001.

[17] Jayesh K. Gupta, Maxim Egorov, and Mykel Kochenderfer. Cooperative multi-agent control using deep reinforcement learning. In Gita Sukthankar and Juan A. Rodriguez-Aguilar, editors, *16th International Conference on Autonomous Agents and Multiagent Systems*, AAMAS, pages 66–83, 2017.

[18] He He, Jordan Boyd-Graber, Kevin Kwok, and Hal Daumé III. Opponent modeling in deep reinforcement learning. In *33rd International Conference on Machine Learning*, ICML, pages 1804–1813, 2016.

[19] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. In *NeurIPS: Deep Learning and Representation Learning Workshop*, 2014.

[20] Robert A. Jacobs, Michael I. Jordan, Steven J. Nowlan, and Geoffrey E. Hinton. Adaptive mixtures of local experts. *Neural Computing*, 3(1):79–87, March 1991.

[21] Max Jaderberg, Wojciech M. Czarnecki, Iain Dunning, Luke Marris, Guy Lever, Antonio Garcia Castañeda, Charles Beattie, Neil C. Rabinowitz, Ari S. Morcos, Avraham Ruderman, Nicolas Sonnerat, Tim Green, Louise Deason, Joel Z. Leibo, David Silver, Demis Hassabis, Koray Kavukcuoglu, and Thore Graepel. Human-level performance in 3D multiplayer games with population-based reinforcement learning. *Science*, 364(6443):859–865, 2019.

[22] Michael Johanson, Martin Zinkevich, and Michael Bowling. Computing robust counter-strategies. In *Advances in Neural Information Processing Systems 20*, 2007.

[23] Nicholas K. Jong and Peter Stone. State abstraction discovery from irrelevant state variables. In *19th International Joint Conference on Artificial Intelligence*, IJCAI, pages 752–757, 2005.

[24] Leslie Pack Kaelbling. Learning to achieve goals. In *13th International Joint Conference on Artificial Intelligence*, IJCAI, pages 1094–1099, 1993.

[25] George Konidaris and Andrew Barto. Autonomous shaping: Knowledge transfer in reinforcement learning. In *23rd International Conference on Machine Learning*, ICML, pages 489–496, 2006.

[26] Landon Kraemer and Bikramjit Banerjee. Multi-agent reinforcement learning as a rehearsal for decentralized planning. *Neurocomputing*, 190:82 – 94, 2016.

[27] Andrew K. Lampinen and Surya Ganguli. An analytic theory of generalization dynamics and transfer learning in deep linear networks. In *7th International Conference on Learning Representations*, ICLR, 2019.

[28] Marc Lanctot, Vinicius Zambaldi, Audrūnas Gruslys, Angeliki Lazaridou, Karl Tuyls, Julien Pérolat, David Silver, and Thore Graepel. A unified game-theoretic approach to multiagent reinforcement learning. In *31st International Conference on Neural Information Processing Systems*, NuerIPS, pages 4193–4206, 2017.

[29] Michael L. Littman. Markov games as a framework for multi-agent reinforcement learning. In *11th International Conference on International Conference on Machine Learning*, ICML, pages 157–163, 1994.

[30] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. In *31st International Conference on Neural Information Processing Systems*, NeurIPS, pages 6382–6393, 2017.

[31] Anuj Mahajan, Tabish Rashid, Mikayel Samvelyan, and Shimon Whiteson. Maven: Multi-agent variational exploration. In *33rd Conference on Neural Information Processing Systems*, NeurIPS, 2019.

[32] H. Brendan McMahan, Geoffrey J. Gordon, and Avrim Blum. Planning in the presence of cost functions controlled by an adversary. In *20th International Conference on International Conference on Machine Learning*, ICML, pages 536–543, 2003.

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

[34] Thanh H. Nguyen, Mason Wright, Michael P. Wellman, and Satinder Baveja. Multi-stage attack graph security games: Heuristic strategies, with empirical game-theoretic analysis. In *Workshop on Moving Target Defense*, MTD '17, pages 87–97, 2017. ISBN 978-1-4503-5176-8.

[35] Shayegan Omidshafiei, Jason Pazis, Christopher Amato, Jonathan P. How, and John Vian. Deep decentralized multi-task multi-agent reinforcement learning under partial observability. In *34th International Conference on Machine Learning*, ICML, 2017.

[36] S. J. Pan and Q. Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, Oct 2010.

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

[38] Tabish Rashid, Mikayel Samvelyan, Christian Schroeder de Witt, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. QMIX: Monotonic value function factorisation for deep multi-agent reinforcement learning. In *35th International Conference on Machine Learning*, ICML, pages 4295–4304, 2018.

[39] Sebastian Ruder. An overview of multi-task learning in deep neural networks. *CoRR*, abs/1706.05098, 2017.

[40] Andrei A. Rusu, Sergio Gomez Colmenarejo, Caglar Gulcehre, Guillaume Desjardins, James Kirkpatrick, Razvan Pascanu, Volodymyr Mnih, Koray Kavukcuoglu, and Raia Hadsell. Policy distillation. In *International Conference on Learning Representations*, ICLR, 2015.

[41] Andrei A. Rusu, Neil C. Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *CoRR*, abs/1606.04671, 2016.

[42] Jonathan Schwarz, Jelena Luketina, Wojciech M. Czarnecki, Angieszka Grabska-Barwinska, Yee Whye Teh, Razvan Pascanu, and Raia Hadsell. Progress & compress: A scalable framework for contiual learning. In *35th International Conference on Machine Learning*, ICML, 2018.

[43] Felipe Silva and Anna Costa. A survey on transfer learning for multiagent reinforcement learning systems. *Journal of Artificial Intelligence Research*, 64, 03 2019.

[44] Matthijs Snel and Shimon Whiteson. Learning potential functions and their representations for multi-task reinforcement learning. In *13th International Conference on Autonomous Agents and Multiagent Systems*, AAMAS, pages 637–681, 2014.

[45] Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z. Leibo, Karl Tuyls, and Thore Graepel. Value-decomposition networks for cooperative multi-agent learning. In *17th International Conference on Autonomous Agents and Multiagent Systems*, AAMAS, 2018.

[46] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, second edition, 2018.

[47] Ming Tan. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *10th International Conference on Machine Learning*, ICML, pages 330–337, 1993.

[48] Matthew E. Taylor, Peter Stone, and Yaxin Liu. Value functions for RL-based behavior transfer: A comparative study. In *20th National Conference on Artificial Intelligence*, AAAI, pages 880–885, 2005.

[49] Gerald Tesauro. Extending Q-learning to general adaptive multi-agent systems. In *16th International Conference on Neural Information Processing Systems*, NeurIPS, page 871–878, 2003.

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

[51] Thomas J. Walsh, Lihong Li, and Michael L. Littman. Transferring state abstractions between MDPs. In *ICML-06 Workshop on Structural Knowledge Transfer for Machine Learning*, 2006.

[52] Xiaofeng Wang and Tuomas Sandholm. Learning near-Pareto-optimal conventions in polynomial time. In *16th International Conference on Neural Information Processing Systems*, NeurIPS, pages 863–870, 2003.

[53] Mason Wright, Yongzhao Wang, and Michael P. Wellman. Iterated deep reinforcement learning in games: History-aware training for improved stability. In *20th ACM Conference on Economics and Computation*, EC, pages 617–636, 2019.

[54] Tianpei Yang, Jianye Hao, Zhaopeng Meng, Chongjie Zhang, Yan Zheng, and Ze Zheng. Efficient detection and optimal response against sophisticated opponents. In *International Joint Conference on Artificial Intelligence*, 2019.

[55] Yan Zheng, Zhaopeng Meng, Jianye Hao, Zongzhang Zhang, Tianpei Yang, and Changjie Fan. A deep bayesian policy reuse approach against non-stationary agents. In *32nd Conference on Neural Information Processing Systems*, 2018.

A Experimental Details

Double DQN was used for all experiments [50]. Determining the correct hyperparameters to utilize is a non-trivial problem, because the learning dynamics may vary given different opponent policies. To this end, we sought to find a method for selecting hyperparameters that performs well against a diversity in opponents, while also being computationally tractable to run. We choose hyperparameters that performed best against a uniform-mixture of a fixed set of opponents (five for the soccer environment, and three for cyber-security environment). The opponent policies were generated through PSRO, and were sampled from the resulting strategy sets. For both experiments we also included a random opponent to each strategy set, because we expect this opponent to be one of the more challenging opponents to learn against.

However, there's a chicken-and-egg problem present. In order to utilize PSRO we would also need the aformentioned hyperparameters. As a stop-gap, we choose initial hyperparameters, by evaluating their performance against a random opponent. In summary, for both environments we select hyperparameters by:

1. Sample 200 possible hyperparameter settings, and choose the one that best-performs against a random opponent.
2. Run PSRO until it exceeds a three day walltime.
3. Sample a fixed set of policies from the strategy set generated from PSRO.
4. Sample 200 hyperparameter settings, and evaluate them against uniform mixed-strategy of the four PSRO policies and the random opponent.

We chose to evaluate our hyperparameters against the mixed-strategy opponent, because we believed it offered the most benefit to the baseline method. Future work could look at the interplay of the hyperparameter selection method and the respective performance of both Q-Mixing, and learning a BR directly against a mixed-strategy.

The non-standard hyperparameters listed throughout the appendix are defined as follows:

Timesteps Total number of experiences collected during training.

Exploration Fraction Fraction of the training timesteps used for exploration. The exploration policy is ϵ-greedy, and starts with $\epsilon = 1.0$, and linearly decays to *Exploration Final* hyperparameter.

Exploration Final ϵ The final ϵ value.

Training Frequency Timestep frequency for performing updates.

Training Starts Number of timesteps experienced before training begins.

Number of Simulations The number of simulated episodes performed for evaluation.

A.1 Soccer

A.1.1 Policy & Environment

In this experiment we select five opponent policies, and hold them fixed throughout the experiments. We consider the hyperparameters in Table 3 as candidates, and found the hyperparameters listed in Table 2 to perform best.

To ensure that each method receives the same simulation budget, we allow each pure-strategy BR 60000 timesteps. An interesting future direction is investigating the trade-off of simulation budget and performance that exists between these methods.

The trained DQN was approximated using a 2 hidden layer neural network. The hidden layers each had 50 units and were fully connected with a ReLU activation. The possible actions are moving in any of the four cardinal directions, or staying in place. The input is a vector of length 120, to represent each of the 20 positions on the board having one of the following states:

- Player 0 is on this square and does not have the ball.
- Player 0 is on this square and is holding the ball.

Table 2: Soccer environment hyperparameters against a mixed strategy.

Hyperparameter	Value
Optimizer	Adam
Learning Rate	0.0003
Buffer Size	3000
Gamma	0.99
Timesteps	300000
Batch Size	64
Exploration Fraction	0.33
Exploration Final ϵ	0.01
Training Frequency	1
Training Starts	300

Table 3: Soccer environment considered hyperparameters.

Hyperparameter	Value
Batch Size	32, 64
Buffer Size	300, 1000, 3000, 10000
Learning Rate	1e-3, 3e-3, 1e-4, 3e-4
Timesteps	10000, 30000, 100000, 300000
Exploration Fraction	0.1, 0.3, 0.4, 0.7
Training Starts	100, 300, 1000

- Player 1 is on this square and does not have the ball.
- Player 1 is on this square and is holding the ball.
- The ball is on the ground on this square.
- Unoccupied space.

A.1.2 Opponent Policy Classifier

The hyperparameters selected for training the opponent classifier are listed in Table 4. The replay buffers gathered from training best-responses against each opponent were merged into one dataset. The classifier was trained to predict the opponent for each observation in the dataset. This resulted in 15000 data points, which were randomly split 90-10 between training and validation.

Table 4: Markov-Soccer opponent policy classifier hyperparameters.

Hyperparameter	Value
Optimizer	Adam
Learning Rate	$5 \cdot 10^{-5}$
Loss	Cross Entropy
Batch Size	64

The classifier was a neural network with the same architecture as a single policy; however, the last layer is modified to choose opponents rather than actions. We did not perform a hyperparameter search on this network or learning algorithm.

A.2 Cyber Security Game

In the cyber security environment, two agents played an instance of the attack graph game. We choose the r_{30} graph from Wright et al. [53] that was previously studied using empirical game theoretic analysis and deep reinforcement learning. This graph is an Erdős-Rényi random graph with 30 nodes (including 6 goal nodes) and 100 edges. A visualization of the attack graph is presented in Figure 7.

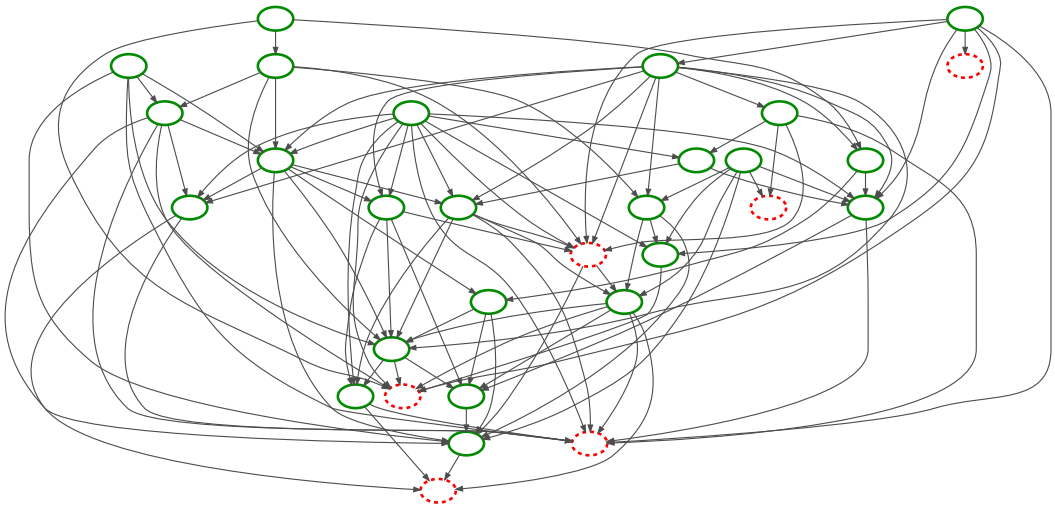

Figure 7: The Attack Graph game enviroment. Goal nodes are represented using red dashed circles. Used with permission from Wright et al. [53].

The candidate hyperparameters are listed in Table 6, with the chosen hyperparameters in Table 5]. Note, that some hyperparameters differ between the attacker and defender. This is because the policies experiences diffference observations, have different action spaces, and experience different rewards (that are non-zero sum).

Table 5: AttackGraph environment hyperparameters.

Hyperparameter	Value
Optimizer	Adam
Learning Rate	$5{\cdot}10^{-5}$
Gradient Norm Clip	10
Buffer Size	30000
Gamma	0.99
Batch Size	32
Exploration Final ϵ	0.03
Attacker Timesteps	700000
Attacker Exploration Fraction	0.3
Defender Timesteps	1000000
Defender Exploration Fraction	0.5
Number of Simulations	250

Table 6: AttackGraph environment considered hyperparameters.

Hyperparameter	Value
Learning Rate	$5{\cdot}10^{-3}$, $5{\cdot}10^{-4}$, $5{\cdot}10^{-5}$
Gradient Norm Clip	None, 0.1, 1, 10
Buffer Size	10000, 30000, 50000, 70000
Batch Size	32, 64
Timesteps	25e4, 40e4, 50e4, 70e4, 100e4, 200e4
Exploration Fraction	0.2, 0.3, 0.5, 0.7

Both the attacker and defender were modelled with a two hidden layer neural network, where each hidden layer had 256 units and was fully connected with tanh activations. The defender had an observation size of 240 and action size of 31; similarly, the attacker had an observation size of 241 and an action size of 106. The attackers's state space was composed of the following features:

Is Active Observation of the true state of the graph, which contains a bit for whether each node has been activate. 30 values.

Can Attack A bit representing whether the preconditions for the and-nodes or or-edges have been met. 105 values.

In Attack Set A bit for each and-node or or-edge representing whether it is currently in the action set. 105 values.

Time Steps Left Number of timesteps left for action set building. 1 value.

The defender's state space was composed of the following features:

Had Alert Observation of the graph that indicates which nodes had previously had an alert raised. This includes a history of the last three observations because the defender only receives an observation of the graph and not the true graph state, resulting in 90 values.

Was Defended For each node in *had alert* whether the node was defended. 90 values.

In Defense Set During action selection a set of nodes are selected. This feature space is used to auto-repressively indicate whether each node is currently in the defense set. 30 values.

Time Steps Left Number of timesteps left for action set building. 30 values (To match the previous work we continued the convention of copying the node for each value. This was originally done because it allowed convolution layers to be tested.).

The action space for this environment is combinatorial, because each agent was able to select any subset of the nodes/edges in the graph. Following Wright et al. [53], we used greedy action set building to choose the player's actions. We refer the reader to their paper for extensive details.

A.3 Opponent Classifier

The hyperparameters considered for the cyber-security OPC are listed in Table 8, and the chosen hyperparameters are in Table 7. Similar to the soccer environment, the replay buffers from training best-response to each opponent were merged into a single dataset. The classifier was trained to predict, given an observation from the dataset, which opponent was being played when the observation was encountered. The dataset of 90000 experiences was split 90-10 between training and validation. The validation set was used to select from 20 possible hyperparameter configurations.

Table 7: Cyber-security opponent policy classifier hyperparameters.

Hyperparameter	Value
Optimizer	Adam
Learning Rate	$5 \cdot 10^{-5}$
Loss	Cross Entropy
Batch Size	64

Table 8: Cyber-security opponent policy classifier considered hyperparameters.

Hyperparameter	Value
Learning Rate	0.0001
Batch Size	32, 64, 128, 256
Hidden Layer Sizes	50x2, 128x2, 200-100, 200x2, 200-128x2

The opponent classifier was a 3 hidden layer neural network. The classifier received the same observation as the defender, size 240, and was passed through fully-connected layers with sizes 200, 128, and 128 respectively with ReLU activations. The network is the same as the policy network, with the last layer modified to choose opponents rather than actions. We did not perform a hyperparameter search on this network or learning algorithm.

A.4 Policy Distillation

In the policy distillation framework, a larger neural network referred to as the "teacher" is used as a training signal for a smaller neural network called the "student". In our experiment the Q-Mixing policy is the teacher to a student neural network that is the size of a single BR policy. The student is trained via supervised learning, reusing the pure-strategy BRs' replay buffers as a dataset. A batch of data is sampled from the replay-buffer and the student predicts Q^S the teacher's response Q^T. The student is trained to imitate the softmax policy of the teacher. The full policy distillation loss is

$$\mathcal{L}_{\text{Distill}} = \sum_{i}^{|D|} \text{softmax}(\frac{Q^T}{\tau}) \ln \frac{\text{softmax}(\frac{Q^T}{\tau})}{\text{softmax}(\frac{Q^S}{\tau})},$$

where D is the dataset of concatenated replay buffers.

The hyperparameters used in policy distillation are listed in Table 9. The student policy is the same neural network that's used in computing the best-responses to individual policies; it is described in Section A.2. We did not perform a hyperparameter search on this network or learning algorithm.

Table 9: Policy distillation hyperparameters.

Hyperparameter	Value
Optimizer	Adam
Learning Rate	0.003
Batch Size	64