Learning Deviation Payoffs in Simulation-Based Games

Samuel Sokota
Swarthmore College
sokota@ualberta.ca

Caleb Ho
Swarthmore College
caleb.yh.ho@gmail.com

Bryce Wiedenbeck
Swarthmore College
bwieden1@swarthmore.edu

Abstract

We present a novel approach for identifying approximate role-symmetric Nash equilibria in large simulation-based games. Our method uses neural networks to learn a mapping from mixed-strategy profiles to deviation payoffs—the expected values of playing pure-strategy deviations from those profiles. This learning can generalize from data about a tiny fraction of a game's outcomes, permitting tractable analysis of exponentially large normal-form games. We give a procedure for iteratively refining the learned model with new data produced by sampling in the neighborhood of each candidate Nash equilibrium. Relative to the existing state of the art, deviation payoff learning dramatically simplifies the task of computing equilibria and more effectively addresses player asymmetries. We demonstrate empirically that deviation payoff learning identifies better approximate equilibria than previous methods and can handle more difficult settings, including games with many more players, strategies, and roles.

Simultaneous-move games are among the most general tools for reasoning about incentives in multi-agent systems. The standard mathematical representation of a simultaneous-move game is the normal-form payoff matrix. It describes incentives in a multi-agent system by listing the strategies available to each agent and storing, for every combination of strategies the agents may jointly select, the resulting utility for each agent. This tabular representation grows exponentially: the payoff matrix for a normal form game with n players and m strategies per player stores utilities for each of m^n pure-strategy profiles. As a result, explicit normal-form representations are typically employed only for games with a very small number of players and strategies.

To study larger games, it is common to replace the explicit payoff table with a more compact representation of the utility function, from which payoff values can be computed on demand. However, pure-strategy utility functions are generally not sufficient for computing Nash equilibria. Instead, Nash-finding algorithms typically depend on one or more of the following quantities being efficiently computable: best responses to profiles of mixed strategies, *deviation payoffs* of mixed-strategy profiles (the expected utility of deviating to a pure-strategy, holding other agents' mixed strategies fixed), or the derivatives of these deviation payoffs. Examples of algorithms requiring each of these computations include, respectively, fictitious play (Brown 1951), replicator dynamics (Taylor and Jonker 1978), and the global Newton method (Govindan and Wilson 2003). Computing any of these quantities naively involves summing over all entries in the fully-expanded payoff matrix.

This means that analyzing large games typically requires carefully selecting a compact representation of the utility function that supports efficient computation of mixed-strategy deviation payoffs. For certain classes, like action-graph games (Jiang, Leyton-Brown, and Bhat 2011), these representations are known, but in other cases, no such representation is available. Moreover, identifying such a compact structure can be challenging, even when it does exist.

In this paper, we explore a machine learning approach to efficiently approximating deviation payoffs and their derivatives, under the hypothesis that sufficiently close approximation of these quantities will tend to yield ϵ-Nash equilibria. At the core of the approach is the idea that the expected payoffs of a mixed-strategy profile can be estimated by the payoffs of pure-strategy profiles sampled from that mixed-strategy profile. Such sampling is extremely noisy, but by sampling from many different mixed-strategy profiles and employing machine learning algorithms that generalize across related mixed strategies, we can estimate a deviation payoff function for the entire game.

Our methods are applicable in settings where a specific compact structure is not known in advance, but where payoff estimates for arbitrary pure-strategy profiles can be acquired. This setting is common in *simulation-based games* (also known as *empirical games* (Wellman 2006; Tuyls et al. 2018)), where game-theoretic models are constructed from payoffs observed in multi-agent simulations. Simulation-based games were initially used to study trading agent competitions (Wellman et al. 2006; Jordan, Kiekintveld, and Wellman 2007) and continuous double auctions (Phelps, Marcinkiewicz, and Parsons 2006; Vytelingum, Cliff, and Jennings 2008) and have since been employed in numerous applications, including designing network routing protocols (Wellman, Kim, and Duong 2013) and understanding the impacts of high-frequency trading (Wah and Wellman 2017). Recently, simulation-based games have arisen in the analysis of the meta game among AlphaGo variants (Lanctot et al. 2017) and other multi-agent reinforcement learning settings including sequential social dilemmas (Leibo et al. 2017). Many of these applications can involve large numbers of symmetric players and would benefit from analysis that scales efficiently and accurately by leveraging these symmetries.

Related Work

Much recent progress in game-theoretic settings has been achieved by the application of machine learning to large extensive-form games. In Go and poker, the extensive form is well-defined, but is far too big to be constructed explicitly, so high-level play (Silver et al. 2017; Moravčík et al. 2017) and approximate equilibrium analysis (Heinrich, Lanctot, and Silver 2015) have required the aid of machine learning based algorithms. These algorithms run simulations to estimate payoffs for various game states and use learning to generalize to unexplored parts of the game tree. Our work employs a similar approach, generating observations by simulation and making generalizations by machine learning only in the context of normal-form deviation payoffs, instead of extensive-form state or state-action values.

In the context of simultaneous-move games, a variety of game models more compact than the normal-form payoff matrix have been explored. Graphical games (Kearns 2007) exploit independence among players, action-graph games (Jiang, Leyton-Brown, and Bhat 2011) exploit independence or partial independence among strategies, and resource-graph games (Jiang, Chan, and Leyton-Brown 2017) exploit independence among sub-decisions that comprise strategies. All of these models can, in particular contexts, exponentially reduce the size of the game representation. But they all require detailed knowledge of a game's structure, which is unlikely to be available in the setting of simulation-based games.

When a precise compact model is not known in advance, it may still be possible to induce the parameters of such a model from simulated or observed data. Duong et al. (2009) construct graphical games from payoff data, and Honorio and Ortiz (2015) construct linear influence games from behavioral data. An advantage of these approaches is that the resulting models can offer structural insight that may generalize to other related settings. However, they suffer from significant limitations in that only the specifically targeted type of structure can be learned. So these approaches are only useful when the analyst has advance knowledge of the payoff structure. In contrast, deviation payoff learning is agnostic to the detailed structure of the payoff functions, making it applicable to a far wider variety of simulation-based games.

We are aware of two papers that have addressed the problem of learning generic models of simultaneous-move games from simulation-based data. The first learns payoff functions that generalize over continuous strategy spaces (Vorobeychik, Wellman, and Singh 2007). The regression takes as input a single-parameter description of a player's strategy and a single-parameter summary of the profile of opponent strategies and outputs a utility. For all but the simplest regression methods, equilibrium computation requires using the learned functions to fill in a payoff matrix for a discretization of the strategy space. This matrix grows exponentially with the number of players, prohibiting analysis of large games with complex payoff functions.

Closest to the present work, Wiedenbeck, Yang, and Wellman (2018) learn payoff functions of symmetric games. For each strategy, a regressor takes as input a pure-strategy opponent-profile and outputs a utility. They provide an approximation method for deviation payoffs that allows for computation of role-symmetric ϵ-Nash equilibria. This approach can learn complicated payoff functions and scales well in the number of players. However, the approximation method, which is only valid for fully-symmetric games learned via RBF kernel Gaussian process regression, is prohibitively expensive, and thereby limited to small data sets from games with small numbers of pure strategies. We compare against this method in our experiments and demonstrate favorable results for deviation-payoff learning.

Background and Notation

A simultaneous-move game $\Gamma = (P, \mathbf{S}, u)$ is characterized by a set of players $P = \{1, \ldots, n\}$, a space of pure-strategy profiles $\mathbf{S}$, and a utility function u. The pure-strategy profile space $\mathbf{S} = \prod_{i \in P} S_i$ is the Cartesian product of strategy sets S_i for each player $i \in P$. An element of the strategy space $\vec{s} \in \mathbf{S}$, specifying one strategy for each player, is known as a pure-strategy profile. An $n-1$ player profile $\vec{s}_{-i}$ specifies strategies for all players except i. The utility function $u : \mathbf{S} \mapsto \mathbb{R}^n$ maps a profile to a payoff for each player; $u_i(\vec{s}) \in \mathbb{R}$ is the component payoff for player i. In a simulation-based game, an exogenously-specified simulator provides noisy point estimates of the utility function. That is, for any given profile $\vec{s}$, we can observe samples $\vec{v} \sim D_{\vec{s}}$, where $D_{\vec{s}}$ is some unknown distribution with mean $u(\vec{s})$.

A mixed strategy σ for player i is a probability distribution over player i's strategy set S_i. The set of all mixed strategies for player i is the simplex: $\Delta(S_i)$. A mixed strategy profile $\vec{\sigma}$ specifies a mixed strategy for each player. The set of mixed strategy profiles is a simplotope: $\mathcal{S} = \prod_{i \in P} \Delta(S_i)$. Similarly to pure profiles, $\vec{\sigma}_{-i}$ is the $n-1$ player profile that results from excluding player i from $\vec{\sigma}$ and $\vec{\sigma}_i$ is the mixed-strategy of player i.

We define a *deviation payoff* for player i, strategy $s \in S_i$, and mixed profile $\vec{\sigma}$ as the expected utility to player i for playing s when all $n-1$ opponents play according to $\vec{\sigma}_{-i}$:

$$\text{devPay}_i(s, \vec{\sigma}) := \mathop{\mathbb{E}}_{\vec{s}_{-i} \sim \vec{\sigma}_{-i}} u_i(s, \vec{s}_{-i}).$$

We use $\text{devPay}_i(\vec{\sigma})$ to denote the vector of deviation payoffs for all strategies of player i. If players jointly play according to mixed-strategy profile $\vec{\sigma}$, then player i achieves expected payoff $\mathbb{E}_{\vec{s} \sim \vec{\sigma}} u_i(\vec{s}) = \text{devPay}_i(\vec{\sigma}) \cdot \vec{\sigma}_i$. The *regret* of a profile $\vec{\sigma}$ is the largest amount any player could have gained by deviating to a pure strategy:

$$\epsilon(\vec{\sigma}) := \max_{i \in P} \max_{s \in S_i} \text{devPay}_i(s, \vec{\sigma}) - \mathop{\mathbb{E}}_{\vec{s} \sim \vec{\sigma}} u_i(\vec{s})$$

A profile with zero regret is a Nash equilibrium; a profile with regret below threshold ϵ is an ϵ-Nash equilibrium. Our aim is to efficiently identify low-regret ϵ-Nash equilibria.

Many common games, especially large simulation-based games, exhibit player symmetries. Two players $i, j \in P$ are symmetric if the game created by permuting i and j is equivalent to Γ. To aid in representing player symmetries, we partition a game's players into roles. A *role* $R \subseteq P$ is a maximal set of symmetric players. A game with one role is fully-symmetric; a game with n roles is fully-asymmetric. The set of roles $\mathcal{R}$ induces a partition $\{S_R : R \in \mathcal{R}\}$ over the set of pure strategies $S := \bigsqcup_{R \in \mathcal{R}} S_R$, where S_R denotes the pure strategies available to players in role R. We use the notation $\mathcal{S}_\mathcal{R} \subset \mathcal{S}$ for the set of mixed-strategy profiles in which all players sharing a role also share a mixed strategy (i.e. the set of role-symmetric mixed-strategy profiles). For $\vec{\sigma} \in \mathcal{S}_\mathcal{R}$, we define $\vec{\sigma}_{-R}$ as the $n-1$ player role-symmetric mixed-strategy profile induced by excluding a player in role R from $\vec{\sigma}$. Similarly, $\vec{\sigma}_R$ is the mixed-strategy in $\vec{\sigma}$ played by the players in role R. We define $\pi \colon \mathcal{S}_\mathcal{R} \to \prod_{R \in \mathcal{R}} \Delta(S_R)$ to be the natural projection operator identifying the shared mixed-strategy of each role in a role symmetric mixed-strategy profile together (note that π is bijective). To illustrate the difference between $\mathcal{S}_\mathcal{R}$ and $\pi(\mathcal{S}_\mathcal{R})$ we can consider a small game with role partition $\mathcal{R} = \{R_1 = \{1,2\}, R_2 = \{3,4,5\}\}$. In this case $\vec{\sigma} \in \mathcal{S}_\mathcal{R}$ takes the form $(\vec{\sigma}_{R_1}, \vec{\sigma}_{R_1}, \vec{\sigma}_{R_2}, \vec{\sigma}_{R_2}, \vec{\sigma}_{R_2})$, while $\pi(\vec{\sigma}) = \vec{\sigma}_\pi$ takes the form $(\vec{\sigma}_{R_1}, \vec{\sigma}_{R_2})$. For a pure-strategy $s \in S_R$ and a role-symmetric mixed-strategy profile $\vec{\sigma} \in \mathcal{S}_\mathcal{R}$, a deviation payoff is unambiguously defined by

$$\text{devPay}(s, \vec{\sigma}) := \mathop{\mathbb{E}}_{\vec{s}_{-R} \sim \vec{\sigma}_{-R}} u_s(\vec{s}_{-R}),$$

where $u_s(\vec{s}_{-R})$ denotes the payoff to a player playing strategy s in the strategy profile $(s, \vec{s}_{-R})$. In this case, we use $\text{devPay}_R(\vec{\sigma})$ to denote the vector of deviation payoffs for $s \in S_R$. Using this notation, regret can be defined

$$\epsilon(\vec{\sigma}) := \max_{R \in \mathcal{R}} \max_{s \in S_R} \text{devPay}(s, \vec{\sigma}) - \text{devPay}_R(\vec{\sigma}) \cdot \vec{\sigma}_R.$$

Methods

Deviation payoff learning uses an exogenous simulator or other data source to estimate deviation payoffs for a role-symmetric mixed-strategy profile by first sampling a pure-strategy profile according to the players' mixed strategies and then querying the simulator at the sampled pure-strategy profile. We construct a data set of such samples and then solve a regression problem to produce a regressor: $S \times \pi(\mathcal{S}_\mathcal{R}) \to \mathbb{R}$. For this methodology to be successful, it is crucial that the simulator possess a significant amount of known player symmetry (otherwise the dimensionality of the domain of the regressor can be very large, making the regression problem difficult). Once this regressor is trained, it allows us to compute candidate role-symmetric ϵ-Nash equilibria. We then gather additional samples in the neighborhood of each equilibrium candidate and iteratively refine the model.

Approximating Deviation Payoffs

Given a simulator which draws samples from $D_{\vec{s}}$ given $\vec{s}$, we can approximate deviation payoffs for $\vec{\sigma}$ by first sampling $\vec{s}$ from $\vec{\sigma}$, and then querying the simulator for payoffs $\vec{v}$ at $\vec{s}$. The expected values of the payoffs $\vec{v}$ are deviation payoffs for deviating from $\vec{\sigma}$ to $\vec{s}$. Thus, learning deviation payoffs can be cast as a regression problem. In general, finding a good solution to this regression problem can be very difficult, especially when alloted only a small number of queries for a large game. In this paper, we are most interested in analyzing large games using only a small number of queries, so we ease the difficulty of the regression problem by limiting ourselves to only learning the deviation payoff values of role-symmetric mixed-strategy profiles. This constraint significantly reduces the dimensionality of the domain of the regressor in the presence of player symmetries. While this constraint also ultimately limits the methods of this paper to approximating *role-symmetric* Nash equilibria, it is what allows us to consider extremely large games (in the presence of player symmetries) that would otherwise be intractible to analyze with a small number of queries.

Formally, we would like to construct a regressor minimizing the quantity,

$$\sum_{s \in S} \int_{\vec{\sigma} \in \mathcal{S}_\mathcal{R}} \left[\text{devPay}(s, \vec{\sigma}) - \text{regressor}(s, \pi(\vec{\sigma}))\right]^2.$$

To approximate the true loss, we use

$$\sum_{\vec{\sigma}_\pi \in \Sigma \subset \pi(\mathcal{S}_\mathcal{R})} \mathcal{L}(\vec{v}, \vec{s}, \vec{\sigma}_\pi) \text{ where } \vec{v} \sim D_{\vec{s}} \text{ and } \vec{s} \sim \pi^{-1}(\vec{\sigma}_\pi)$$

as the training loss, where

$$\mathcal{L}(\vec{v}, \vec{s}, \vec{\sigma}_\pi) = \sum_{s \in \{\vec{s}_i : i \in P\}} \left[\vec{v}_s - \text{regressor}(s, \vec{\sigma}_\pi)\right]^2,$$

and Σ is a finite number of randomly sampled projected role-symmetric mixed-strategy profiles. In the limit as every projected role-symmetric mixed-strategy profile is sampled an infinite number of times, the minimum of the training loss coincides with the minimum of the true loss at almost every profile.

We use a multiheaded neural network as our regressor. The network takes a projected role-symmetric mixed-strategy profile $\vec{\sigma}_\pi \in \pi(\mathcal{S}_\mathcal{R})$ as input and has a head for each pure-strategy $s \in S$ whose output is the predicted deviation payoff $\text{devPay}(s, \pi^{-1}(\vec{\sigma}_\pi))$.

Figure 1 shows an example of a learned game with three pure strategies. Each simplex represents the deviation payoff function of the labeled pure-strategy. Each point on a

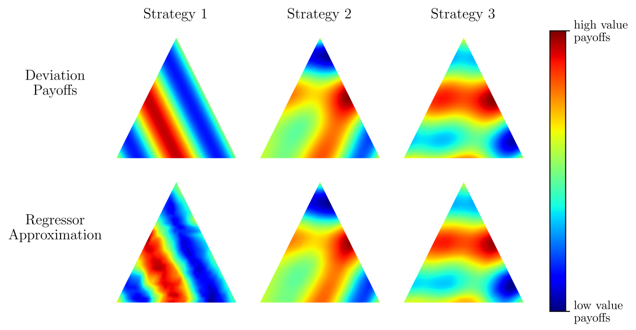

Figure 1: A regressor can closely approximate the true deviation payoffs in a 100-player symmetric game using payoff samples.

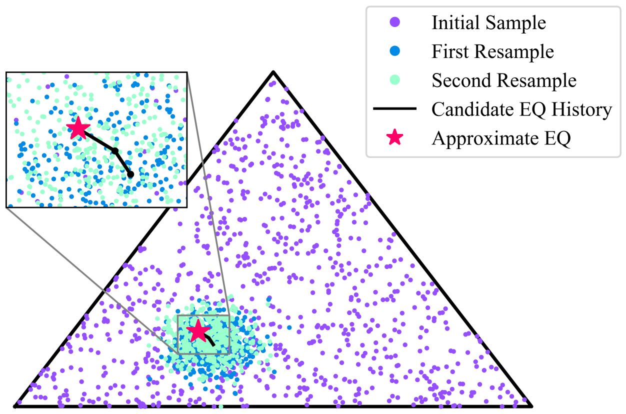

Figure 2: The black path shows movement of the candidate equilibrium across two iterations of sampling and re-solving.

simplex (in barycentric coordinates) represents a mixed-strategy. The heat corresponds to the value of the deviation payoff at that mixed-strategy. The close match between top (true deviation payoffs) and bottom (neural network estimates) rows is typical; we find that the network is consistently able to learn deviation payoffs with high accuracy.

Approximating Role-Symmetric Nash Equilibria

We choose to learn deviation payoffs because they provide sufficient information to solve for Nash equilibria. Replicator dynamics relies solely on deviation payoffs, while other role-symmetric Nash equilibrium computation techniques rely on a combination of deviation payoffs and their derivatives (which can easily be provided by differentiable regressors such as neural networks).

The existence of a Nash equilibrium depends only on the deviation payoffs at that mixed-strategy profile, irrespective of deviation payoffs elsewhere in mixed-strategy profile simplotope. So to find good approximate equilibria it is better to emphasize learning in areas of the simplotope where there are likely to be Nash equilibria, even at the cost of weaker learning in other areas.

In order to do this in a query-efficient manner, we integrate sampling and training with equilibrium computation, as described in Algorithm 1. Specifically, the initial sample of mixed strategies is taken roughly uniformly at random across the projected mixed-strategy profile simplotope (this can be done with a Dirichlet distribution). A regressor trained on these examples is used to solve for equilibria. Then, additional samples are taken in the neighborhood of the candidate Nash equilibria and we solve for Nash equilibria again using a regressor trained on the entire set of examples. The process of resampling and re-solving can be repeated for several iterations. We find that this process significantly lowers deviation payoff error in the areas of retraining, and is sufficient to find low regret approximate equilibria, even with a small number of training examples.

Figure 2 shows an example of the sampling process across two iterations of resampling and re-solving. A candidate equilibrium is identified using the initial purple samples. This candidate is revised using the blue samples taken in a neighborhood of the candidate. Finally, the revised candidate is revised again using the green samples taken in its neighborhood. The black path shows the movement of the candidate equilibrium across iterations. The pink star is the final guess for the location of the equilibrium.

Algorithm 1 Approximating Role-Symmetric Nash Equilibria

```
procedure APPXRSNASHEQUILIBRIA(
INITIALQUERIES, RESAMPLEQUERIES, NUMITERS):
    $[\vec{\sigma}_\pi] \leftarrow \text{sample}_{\pi(\mathcal{S}_\mathcal{R})}(\text{initialQueries})$
    $[\vec{s}] \leftarrow \text{sampleProfiles}([\vec{\sigma}_\pi])$
    $[\vec{v}] \leftarrow \text{samplePayoffs}([\vec{s}])$
    data $\leftarrow ([\vec{\sigma}_\pi], [\vec{s}], [\vec{v}])$
    regressor.fit(data)
    repeat
        $[\vec{\sigma}^*_\pi] \leftarrow \text{findNash}(\text{regressor})$
        $[\vec{\sigma}_\pi] \leftarrow \text{sampleNbhd}([\vec{\sigma}^*_\pi], \text{resampleQueries})$
        $[\vec{s}] \leftarrow \text{sampleProfiles}([\vec{\sigma}_\pi])$
        $[\vec{v}] \leftarrow \text{samplePayoffs}([\vec{s}])$
        data $\leftarrow$ data + $([\vec{\sigma}_\pi], [\vec{s}], [\vec{v}])$
        regressor.fit(data)
    until numIters
    return findNash(regressor)
```

In general, sampling the neighborhood of a point on a manifold can be difficult. To do so, we sample from an isotropic Gaussian at the image of each mixed-strategy in the projected role-symmetric mixed-strategy profile under the coordinate patch parameterizing the mixed-strategy simplex for that role, and then send each sample back to its respective simplex under the inverse of its coordinate patch. Sometimes, the samples will not fall within the simplex when they are preimaged by the coordinate patch. This problem becomes particularly pronounced as the number of pure strategies grows large (intuitively, this is because simplices become pointier with the number of dimensions). Rejection sampling quickly becomes highly impractical. To counteract this, we project the preimage of each sample into its simplex (Chen and Ye 2011). Projection sends samples that fall outside the simplotope onto its edges, thereby putting more emphasis onto the edges than rejection sampling would. In practice, we find that overall performance benefits from the extra emphasis on the edges of the simplotope, which are otherwise difficult to learn because there is less nearby volume within the simplotope from which to generalize.

Formally, say we would like to sample a projected role-symmetric mixed-strategy profile $\vec{\sigma}'_\pi$ in the neighborhood of $\vec{\sigma}_\pi$. Let $\alpha_R \colon \Delta(S_R) \to U_R$ be the coordinate patch parameterizing the mixed-stategy simplex for role R. Then the mixed-strategy $(\vec{\sigma}'_\pi)_R$ for role R in the sampled projected role-symmetric mixed-strategy profile is given by

$$(\vec{\sigma}'_\pi)_R = \mathcal{P}(\alpha_R^{-1}(\sigma_E)) \quad \text{where} \quad \sigma_E \sim \mathcal{N}(\alpha_R((\vec{\sigma}_\pi)_R), k \cdot \mathbb{I}),$$

and $\mathcal{P}$ is the projection operator described by (Chen and Ye 2011). We use the notation σ_E to capture the intuition that σ_E is the Euclidean representation of a mixed-strategy.

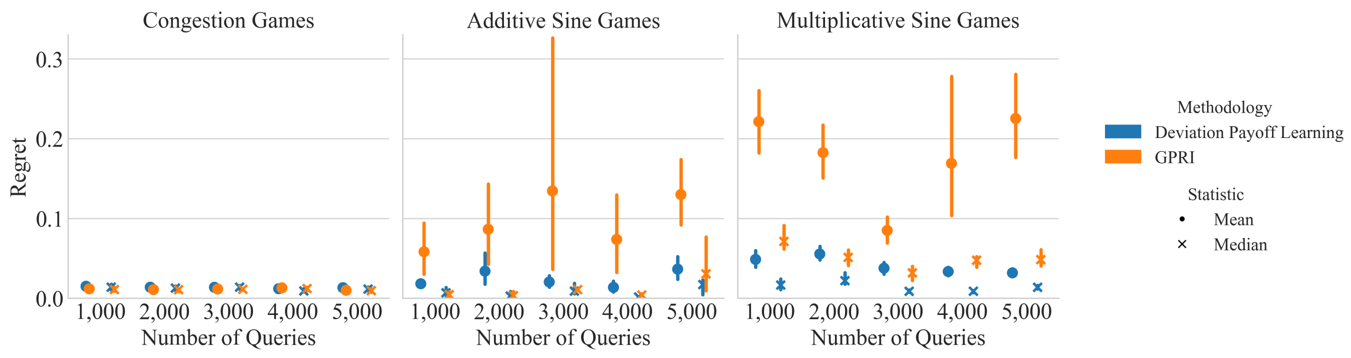

Figure 3: Gaussian process regression integration and deviation payoff learning achieve similar performance in congestion games, but deviation payoff learning significantly outperforms Gaussian process regression integration in more difficult games.

Experiments

Experimental Specification

In the following experiments, we employed a network with three dense hidden layers of 128, 64, and 32 nodes, followed by a head for each strategy with 16 hidden nodes and a single output. This architecture was tuned based on 100 player, five pure-strategy games. We held the structure fixed throughout our experiments, other than varying the number of input nodes and heads to match the number of pure strategies. We split queries half and half between the initial sample and resampling for our experiments and used ten iterations of resampling. For resampling, we chose the variance $k \cdot \mathbb{I}$ of the distribution we sampled from such that the expected distance between a random sample and the mean was about 0.05 (this requires a different constant k for each dimension).

To perform our experiments, we generate action-graph games to serve as a proxy for simulators. We consider three classes of games: congestion games, additive sine games, and multiplicative sine games. Additive and multiplicative sine games are action-graph games whose function nodes compute respectively the sum or the product of a low-degree polynomial and a long-period sine function. Each class of game is defined by a distribution over the parameters of action-graph games of a particular structure. Using the performance on sampled games, we present expectations of the median and mean performance on each class, with 95% confidence intervals. In order to have a basis of comparison across different settings, the payoffs of every game are normalized to mean zero and standard deviation one. Since we are generating the underlying games, we are able to measure the true regret of the computed approximate equilibria to use as our metric of performance.

Comparison to Existing Work

We compare deviation payoff learning to Gaussian process regression integration (GPRI) (Wiedenbeck, Yang, and Wellman 2018). GPRI is the existing state of the art for generic model learning of fully-symmetric, simultaneous-move games. However, the means by which GPRI reconstructs deviation payoffs from learned utilities is expensive and scales poorly both with the number of training examples and the number of pure strategies in the game. In contrast, we show that deviation payoff learning can handle orders of magnitude more training examples and pure strategies, and also generalizes to games with role-asymmetries.

In our comparison, we use symmetric 100 player games with five pure strategies and up to 5,000 queries to the simulator. We were unable to make a comparison on larger numbers of strategies or larger data sets due to the limitations of GPRI. In Figure 3 we see that both deviation payoff learning and GPRI achieve strong results on congestion games. However, on harder games, the performance of GPRI deteriorates significantly more than that of deviation payoff learning. Thus, in addition to having a much wider range of applicability than GPRI, deviation payoff learning tends to achieve much lower regret within the range of applicability of GPRI. Note that the regret scale on these graphs is dramatically larger than on any of the subsequent plots examining deviation payoff learning.

Scaling in Noise

It is typical in an empirical game for the simulator to be noisy. In our experimental setup, two different noise models are plausible: noise in player payoffs and noise in pure-strategy payoffs. When there is noise in player payoffs, each player receives their own noisy payoff. This provides a natural form of noise reduction, as we can average over the payoffs of the players choosing the same pure-strategy. Noise in pure-strategy payoffs, where one noisy payoff is provided for each pure-strategy supported in the payoff profile, offers a strictly more challenging problem. In our experiment, we consider the harder case of noise in pure-strategy payoffs with the understanding that our method will only become more robust under noise in player payoffs. We use symmetric 100 player, five pure-strategy additive sine games. The noise is sampled from a mean zero Gaussian of varying width.

We see in Figure 4 that deviation payoff learning can tolerate large amounts of noise while maintaining strong performance. Adding mean zero Gaussian noise with standard deviation 1/10 appears to have a very marginal effect on learning. Even with noise having standard deviation 1/2, which is on the same order of magnitude as the standard de-

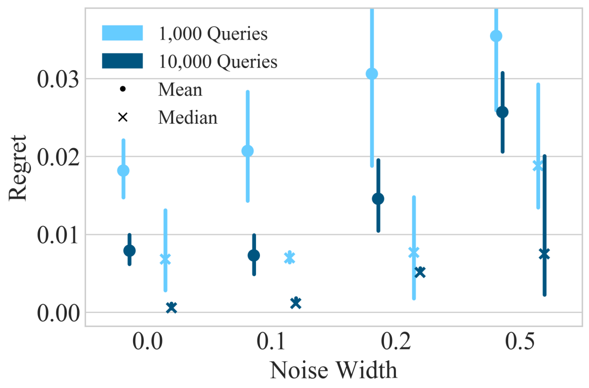

Figure 4: Deviation payoff learning is robust to noise.

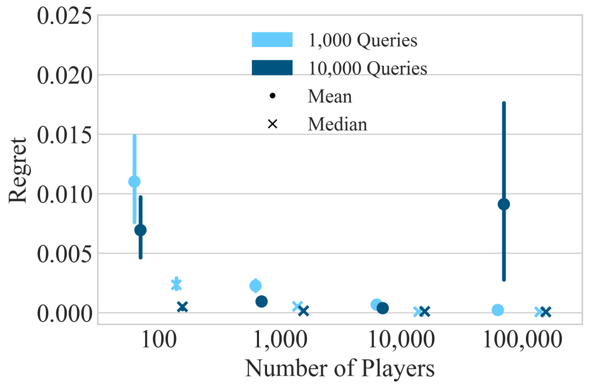

Figure 5: The performance of deviation payoff learning actually improves with the number of players for additive sine games.

viation of payoffs of the game, deviation payoff learning is still able to find comparably good mixed strategies to those GPRI computed with the same number of queries, but without noisy payoffs (see Figure 3). The robustness of deviation payoff learning to noise is not surprising, as the central idea of the learning process is to determine the mean of the distribution of payoffs. Adding mean zero noise increases the variance of this distribution but does not fundamentally change the problem.

Scaling in the Number of Players

One of the advantages of learning deviation payoffs directly is that the learning process scales well with the number of players. In particular, unlike existing methods, deviation payoff learning avoids learning the tabular representation of the utility function, which grows exponentially with the number of players (although the deviation payoff function does tend to become more complex due to it being a weighted sum over the exponentially growing utility function).

In Figure 5, we present results from symmetric additive sine games with five pure strategies. It appears that deviation payoff learning actually enjoys increased performance

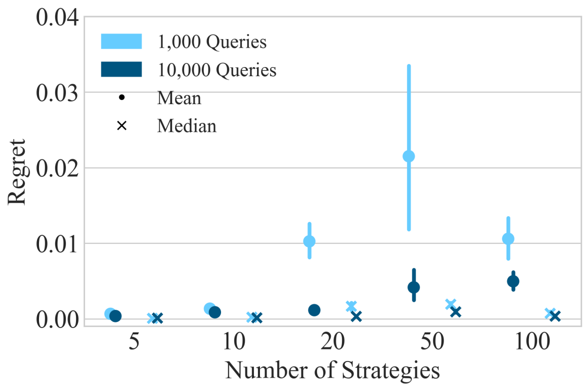

Figure 6: Deviation payoff learning can scale into games with large numbers of pure strategies when there are many players.

with larger numbers of players for additive sine games. It is possible that this is a result of having more training examples (since pure strategies are less often unsupported when there are more players). This is strong evidence that deviation payoff learning scales extremely well in the number of players (though we are not suggesting that the performance will improve with more players for games in general).

Scaling in the Number of Pure-Strategies

Past analyses of simulation-based games with large numbers of pure strategies, such as Dandekar et al. (2015), have largely been limited to exploring strategies iteratively to find small-support Nash equilibria. GPRI is similarly limited due to difficulty handling games with more than five pure-strategies. In contrast, deviation payoff learning offers a significant advancement in that it allows for direct analysis of games with a much larger number of pure strategies. This opens the possibility of discovering large-support Nash equilibria in games with many strategies.

As the number of pure strategies becomes large with respect to the number of players, it increases the difficulty of the learning problem because there are fewer training examples per pure-strategy deviation payoff function. In order to minimize this effect and focus on scaling in the number of pure strategies, we fix a large number of players for this experiment. Results are presented from symmetric 10,000 player additive sine games.

In Figure 6, we see that, in the case in which the number of players significantly exceeds the number of pure strategies, deviation payoff learning achieves surprisingly strong results for additive sine games even for large numbers of strategies. However, we note that our preliminary experiments suggest that performance scaling deteriorates significantly faster when the number of players in the game is more comparable to the number of pure strategies.

Scaling in the Number of Roles

Perhaps the most significant advantage deviation payoff learning can provide is the ability to handle games with role asymmetries. In practice, many games of interest are not

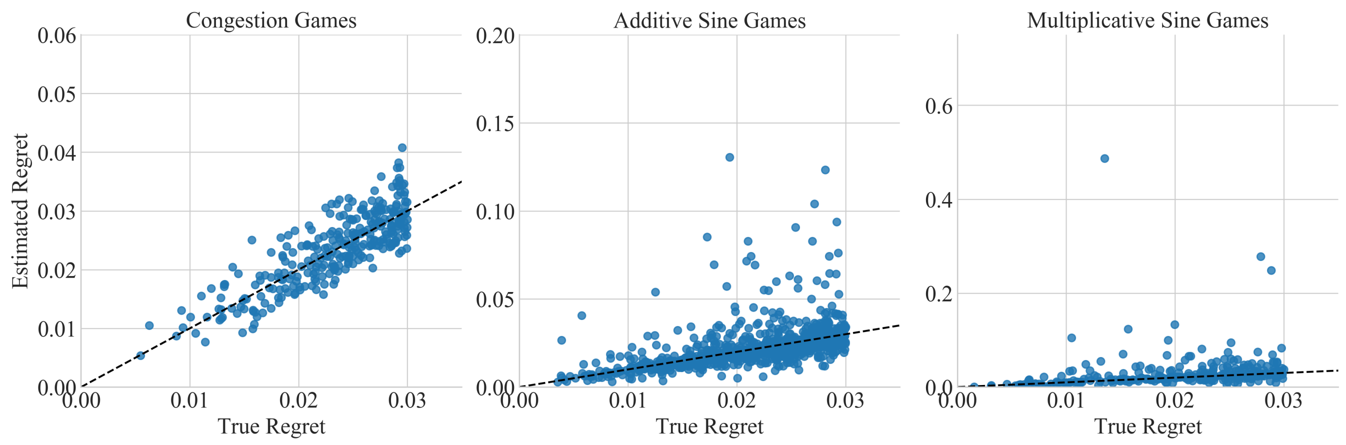

Figure 7: Validating equilibria can help catch mistakes, but on particularly hard (multiplicative sine) games, the sampling estimate can be overly conservative. Dashed lines indicate $45°$, where true and estimated regrets are equal.

fully symmetric. Yet existing methods have difficulty representing multi-role games. We assess the ability of deviation payoff learning to handle role asymmetries by fixing the number of players and strategies and varying the number of roles. Each role is allotted an equal number of players and an equal number of strategies. Results are from 1,200 player, 12 pure-strategy additive sine games.

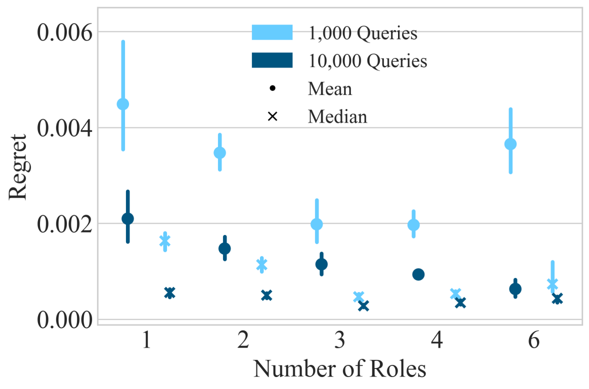

Figure 8: Deviation payoff learning scales well with role asymmetries.

We observe in Figure 8 that, for the most part, deviation payoff learning scales positively with the number of roles. This can likely be attributed to the game size decreasing as the number of roles increases (holding $|S|$ fixed). In other words, the number of terms in the deviation payoff function decreases with the number of roles.

Validating Learned Models

We have demonstrated that deviation payoff learning tends to produce low regret mixed strategies. While many results for congestion games and multiplicative sine games were omitted due to space constraints, we found that performance tended to trend similarly. Even in settings where outliers cause higher expected regret, median regret mostly remains low. It is desirable to be able to distinguish outlying, high regret mixed-strategy profiles returned by deviation payoff learning from the more typical, low regret mixed-strategy profiles. In simulated games, accessing the regret directly is not possible, but approximating deviation payoffs also allows us to compute an approximation of regret (details can be found in the Background and Notation).

We propose that after equilibria have been identified in the model, the analyst verify the results by simulating many profiles sampled from each equilibrium to estimate its regret. If this estimate agrees with the learned model, analysts can have confidence in the result, but disagreement may indicate that additional data is required. We found that a large number of samples is required to achieve reliable estimates, so Figure 7 shows results of 1,000 queries per role-symmetric mixed-strategy profile in symmetric 100 player five pure-strategy games. Each mixed-strategy profile was found by rejection sampling to have low true-game regret, and we compare against regret estimated by sampling. In congestion games, these predictions are quite accurate, while in sine games, overestimates are more common.

Conclusion

We have developed a novel methodology for learning ϵ-Nash equilibria in simulation-based games. We have shown that deviation payoff learning outperforms the previous best and sets new performance standards for scaling in noise, number of players, number of strategies, and role asymmetries. We believe that the tools presented here will enable new simulation-based analysis of many exciting multi-agent interaction domains.

References

Brown, G. W. 1951. Iterative solution of games by fictitious play. *Activity analysis of production and allocation* 13(1):374–376.

Chen, Y., and Ye, X. 2011. Projection onto a simplex. *arXiv preprint arXiv:1101.6081*.

Dandekar, P.; Goel, A.; Wellman, M. P.; and Wiedenbeck, B. 2015. Strategic formation of credit networks. *ACM Transactions on Internet Technology (TOIT)* 15(1):3.

Duong, Q.; Vorobeychik, Y.; Singh, S. P.; and Wellman, M. P. 2009. Learning graphical game models. In *IJCAI*, 116–121.

Govindan, S., and Wilson, R. 2003. A global newton method to compute nash equilibria. *Journal of Economic Theory* 110(1):65–86.

Heinrich, J.; Lanctot, M.; and Silver, D. 2015. Fictitious self-play in extensive-form games. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, 805–813.

Honorio, J., and Ortiz, L. 2015. Learning the structure and parameters of large-population graphical games from behavioral data. *Journal of Machine Learning Research*.

Jiang, A. X.; Chan, H.; and Leyton-Brown, K. 2017. Resource graph games: A compact representation for games with structured strategy spaces. In *AAAI*, 572–578.

Jiang, A. X.; Leyton-Brown, K.; and Bhat, N. a. R. 2011. Action-Graph Games. *Games and Economic Behavior* 71(1):141–173.

Jordan, P. R.; Kiekintveld, C.; and Wellman, M. P. 2007. Empirical game-theoretic analysis of the tac supply chain game. In *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, 193. ACM.

Kearns, M. 2007. Graphical games. In Vazirani, V.; Nisan, N.; Roughgarden, T.; and Tardos, E., eds., *Algorithmic game theory*. Cambridge, UK: Cambridge University Press. chapter 7, 159–180.

Lanctot, M.; Zambaldi, V.; Gruslys, A.; Lazaridou, A.; Tuyls, k.; Perolat, J.; Silver, D.; and Graepel, T. 2017. A unified game-theoretic approach to multiagent reinforcement learning. In Guyon, I.; Luxburg, U. V.; Bengio, S.; Wallach, H.; Fergus, R.; Vishwanathan, S.; and Garnett, R., eds., *Advances in Neural Information Processing Systems 30*, 4190–4203. Curran Associates, Inc.

Leibo, J. Z.; Zambaldi, V.; Lanctot, M.; Marecki, J.; and Graepel, T. 2017. Multi-agent reinforcement learning in sequential social dilemmas. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*, 464–473. International Foundation for Autonomous Agents and Multiagent Systems.

Moravčík, M.; Schmid, M.; Burch, N.; Lisỳ, V.; Morrill, D.; Bard, N.; Davis, T.; Waugh, K.; Johanson, M.; and Bowling, M. 2017. Deepstack: Expert-level artificial intelligence in heads-up no-limit poker. *Science* 356(6337):508–513.

Phelps, S.; Marcinkiewicz, M.; and Parsons, S. 2006. A novel method for automatic strategy acquisition in n-player non-zero-sum games. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, 705–712. ACM.

Silver, D.; Schrittwieser, J.; Simonyan, K.; Antonoglou, I.; Huang, A.; Guez, A.; Hubert, T.; Baker, L.; Lai, M.; Bolton, A.; et al. 2017. Mastering the game of go without human knowledge. *Nature* 550(7676):354.

Taylor, P. D., and Jonker, L. B. 1978. Evolutionary stable strategies and game dynamics. *Mathematical biosciences* 40(1-2):145–156.

Tuyls, K.; Pérolat, J.; Lanctot, M.; Leibo, J. Z.; and Graepel, T. 2018. A generalised method for empirical game theoretic analysis. *CoRR* abs/1803.06376.

Vorobeychik, Y.; Wellman, M. P.; and Singh, S. 2007. Learning payoff functions in infinite games. *Machine Learning* 67(1-2):145–168.

Vytelingum, P.; Cliff, D.; and Jennings, N. R. 2008. Strategic bidding in continuous double auctions. *Artificial Intelligence* 172(14):1700–1729.

Wah, E., and Wellman, M. P. 2017. Latency arbitrage in fragmented markets: A strategic agent-based analysis. *Algorithmic Finance* 5:69–93.

Wellman, M. P.; Jordan, P. R.; Kiekintveld, C.; Miller, J.; and Reeves, D. M. 2006. Empirical game-theoretic analysis of the tac market games. In *AAMAS-06 Workshop on Game-Theoretic and Decision-Theoretic Agents*.

Wellman, M. P.; Kim, T. H.; and Duong, Q. 2013. Analyzing incentives for protocol compliance in complex domains: A case study of introduction-based routing. In *12th Workshop on the Economics of Information Security*.

Wellman, M. P. 2006. Methods for empirical game-theoretic analysis (extended abstract). In *Proceedings of the National Conference on Artificial Intelligence*, 1152–1155.

Wiedenbeck, B.; Yang, F.; and Wellman, M. P. 2018. A regression approach for modeling games with many symmetric players. In *32nd AAAI Conference on Artificial Intelligence*.