Modified upates for Mirror Descent based methods in two-player zero-sum games.

by

Thiruvenkadam Sivaprakasam Radhakrishnan

THESIS

Submitted as partial fulfillment of the requirements
for the degree of Master's in Computer Science
in the Graduate College of the
University of Illinois at Chicago, 2023

Chicago, Illinois

ACKNOWLEDGMENTS

The thesis has been completed. .. (INSERT YOUR TEXTS)

YOUR INITIAL

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

LIST OF TABLES

LIST OF FIGURES

LIST OF NOTATIONS

θ	Parameters of a function approximator.

SUMMARY

Put your summary of thesis.

CHAPTER 1

INTRODUCTION

Multiagent Systems (MAS) [1] are frameworks of problems of distributed nature with independent actors called agents that cooperate or compete to achieve some outcome. Due to the complexity of such problems, designing efficient algorithms for them can be challenging. Machine learning presents opportunities in creating learning algorithms for agents in Multi-agent settings. Multiagent Learning (MAL) is an area of research that studies the application of machine learning to Multiagent Systems. Reinforcement learning (RL), an area of study within machine learning is a popular choice of learning technique due to its compatability in terms of learning through interaction. Although RL algorithms have gained applications in various domains, their direct applicability in multi-agent setting is limited due to the non-stationarity problem. Multiagent Reinforcement Learning (MARL) as a research area deals with designing efficient and performant RL algorithms for general multiagent problems by incorporating ideas from game theory, online learning, and evolutionary biology etc. Apart from the non-stationarity problem There are a range of problems that are currently being studied in designing Multi-agent systems, including communication,

Two-player zero-sum games are instances of multiagent systems that are purely competitive in nature. Due to their unique structure, two-player zero-sum games can also be represented as saddle-point problems that provide certain analytical properties. This presents an opportunity to study and design optimization algorithms under the dynamics of two-player zero-sum games, with an aim to extend these algorithms to general multiagent settings. Two-player zero-sum games also model other learning problems [2] and as such these advances can also be equivalently applied or adapted to solve them.

Mirror Descent is a popular first order optimization algorithm that has wide applications. In this work, we study mirror-descent based reinforcement learning algorithms in the context of two-player zero-sum games. Specifically we study Mirror Descent Policy Optimization, and Magnetic Mirror Descent, two recent algorithms that extends Mirror Descent as Single RL algorithms, and approximate equilibrium solvers. We propose novel improvements to these algorithms by incorporating existing techniques and study their convergence behaviors in normal form games. We also evaluate the performance of these algorithms in large extensive form games under function approximation. We summarize our findings regarding the effectiveness of mirror-descent based reinforcement learning algorithms in the multiagent setting and the effect of the modifications we apply. Through these study we provide some recommendations in designing MARL algorithms and close with some remarks about future research directions.

? Recent developments have demonstrated a vast potential in application of machine learning, and deep learning to a wide range of domains with efforts in creating more general large-scale foundational models to smaller specialized models that are optimized for specific use cases. Due to such progress it can be expected that there will soon be a prevalance of such learnt applications deployed in a variety of context gaining increasing power of autonomy and execution. It can be forseen that soon these models will have to be adaptive in the presence of other such models that are either competing or cooperative in nature.

Motivation:

- MARL
- Last iterate convergence
- NeuRD
- Extrapolation methods
- 2p0s game applications like GANs

1.1 Outline

The rest of the thesis is organized as follows. We begin by providing some background and definitions in section 2 that are useful for the understanding of the algorithms and methods described in section 3 and 4. Section 3 introduces Mirror Decsent and expands on Mirror Descent based methods for solving Reinforcement learning problems. Section 4 discusses combining novel improvements on top of these methods and discusses their structure and the expected effects. In Section 5, we dive into some experimental results and discuss the performance of these algorithms in different settings. We then close the thesis with some discussion.

CHAPTER 2

BACKGROUND

In this section we lay out some background on the topics relevant to this work. We first discuss foundational concepts within reinforcement learning that serves as a base for the rest of the discussion. The framework of multiagent learning problems have also been conventionally rooted in Game Theory, and hence we cover some key ideas that help establishing this consistency. Finally, we also present some preliminary concepts from online learning and optimization algorithms that are useful in understanding the approaches we study in this work.

2.1 Game Theory

Game theory is the mathematical study of interaction between agents to produce outcomes while trying to uphold certain individual or group preferences. It has a wide range of applications including economics, biology, and computer science. In this work, we mainly focus on a branch of game theory called non-cooperative game theory that assumes each agent has their own indivudual preference to uphold as opposed to a global preference.

2.1.1 Problem Representations

In game theory, the ordering of an agent's preferences over certain outcomes are formalized through a Utility or a *Payoff function* that maps outcomes to a real value. Problems are commonly represented as a *Game* that encodes information about the agents, possible actions agents can take in different situations, their preferences, and the outcome of a specific interaction. A *Strategy* typically refers to the agent's choice of taking actions given the information it has. Agent's strategies can be *pure* (deterministic) or *mixed* (distribution over possible actions). Beyond these preliminary notions, there are different representations of games depending on the structure of the problem. We now briefly describe two common representations that are relevant to this work. For a more in-depth discussion of game representations, and their significance we refer the reader to [3].

Normal-Form Games: Normal-Form or Strategic form games are the most basic and foundational form of representing problems in game theory. In a Normal-form game (NFG), all agents act simultaneously to reveal the outcome immediately after, and a payoff is assigned to each agent. A more formal definition of a normal-form game is as follows:

Definition 1 (Normal-form games) *A n-player, general-sum normal form game is given by the (N, A, u) tuple, where N is the set of players, $A = A_1 \times A_2 \ldots \times A_n$, with A_i being the set of actions available to player i, and $u = (u_i \forall i \in N)$ is the set of utility functions that map an action profile to a real utility value for each agent, $u_i : A \mapsto \mathbb{R}$.*

Normal-form games are typically represented using an n-dimensional matrix, where each dimension represents an action available to each agent, and each entry is the payoff assigned to each agent given that action profile. A few popular examples of NFGs are Rock-paper-scissors (RPS), Matching pennies, and Prisoner's dilemma etc.

Sequential Games

Normal-form games are limited in representing many real-world problems that necessitate agents to act sequentially along the temporal dimension. A common choice in representing such problems is the Extensive-form game which facilitiates sequential decision making.

Definition 2 (Extensive-form games)

Given their tree-like representation, EFGs can be inefficient to operate over. Reduced-form NFGs are the normal-form representation of an EFG, where each row/column represents a pure-strategy of the entire EFG, instead of just a single action. Given this exponential blowup of action space, reduced-form NFGs are also computationally challenging to work with. Another useful representation is the Sequence-form, where instead of the entire pure-strategy, entries represent histories of actions taken so far.

Definition 3 (Sequence form games)

2.1.2 Solution Concepts

While the above representations provide a well-defined framework to encode information about the problem, solution concepts define a notion of optimality of the agent's strategies given these representations. The main premise is identifying an *equilibrium* point in the joint strategy space of all the agents. However, how this equlibrium point is defined, and what guarantees it provides leads to various definitions of equilibrium points.

- Nash equilibrium - Existence of a nash equilibrium
- Quantal response equilibrium - Uniqueness of QRE

2.2 Reinforcement Learning

Reinforcement learning (RL) is sub-domain of machine learning that deals with designing interactive *agents* that learn to maximize a *reward* signal in an *environment*. The reward signal encodes information about the goal that the agent has to learn to achieve without any specific directions about how that goal should be acheived.

Markov Decision Process. Markov Decision Processes (MDPs) provide a framework to formalize reinforcement learning problems. A γ-discounted MDP ($\mathcal{M}$) is given by the tuple, $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, R, \gamma)$ where: $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $P(s', r|s, a) : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto [0, 1]$ is the transition dynamics function, $R(s, a) \subset \mathbb{R}$ is the reward function, $\gamma \in [0, 1]$ is the discount factor.

The objective of an agent interacting with this MDP is to learn to act in a way that maximize its *expected return* at each state. The expected return, G_t, at time step t is the discounted sum of rewards accumulated by the agent starting from t till the end of the episode (episodes?). Formally, $G_t = \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$, where the discount rate γ indicates how much importance is placed upon future rewards over immediate rewards. The objective of an agent within an MDP is to learn to act at each state so as to maximize their *expected returns*.

While the expected return quantifies the objective, policy and value functions are representations that dictate how the agent should act in a given state. Reinforcement learning algorithms typically involve learning polices and estimating value functions. A *Policy* is a mapping from a state to an action distribution $\pi : \mathcal{S} \mapsto \mathcal{A}$. A *Value function*, $V_\pi(s)$ estimates the expected reward that can be achieved from the current state under the policy π: $V_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s]$. Analogously, the *Q-value function* estimates the expected reward of taking a specific action a at a given state s and then following the policy π: $Q_\pi(s, a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a]$. The difference between Q and V functions is referred to as the advantage function $A_\pi(s, a) = Q_\pi(s, a) - V_\pi(s)$. This is the advantage of taking a particular action over following the average policy.

Value functions impose a ranking over policies in terms of the expected return achieved by an agent following a given policy. An *optimal policy* (π^*) is one that is atleast as better as any other policy ($\pi \in \Pi$) in terms of this ranking, i.e. $V_{\pi^*}(s) \geq V_\pi(s), \forall s \in \mathcal{S}$. While there maybe more than one optimal policy, they all share the same unique *optimal value function*: $V_{\pi^*} = \max_\pi V_\pi(s), \forall s \in \mathcal{S}$.

2.2.1 Tabular methods

For a given policy, the value function satisfies the **Bellman equation**:

$$V_\pi(s) \doteq \mathbb{E}_\pi \left[r + \gamma V_\pi(s')|R_t = r, S_{t+1} = s'\right] \tag{2.1}$$

For small state spaces, it is possible to learn optimal value functions by iterating using Dynamic programming. Starting with an arbitrary policy, we can learn an optimal policy interleaving Policy evaluation, and Policy improvement. In the *Policy Evaluation* step, we first estimate the value function until it satisfies Equation 2.1. Next, in the *Policy Improvement* step, we update the policy greedily at every state until we acheive a strictly better policy.

This method of interleaving these two steps is called *Policy Iteration.* Alternatively, *Value Iteration* approximates this improvement by only updating the policy greedily with respect to the immediate next step.

At step k of learning a tabular policy,

- **Policy Evaluation**: $v_{k+1}(s) \doteq \mathbb{E}_\pi \left[R_{t+1} + \gamma v_k(S_{t+1}) | S_t = s\right]$
- **Value Iteration**: $v_{k+1}(s) \doteq \mathbb{E}_\pi \left[R_{t+1} + \gamma v_k(S_{t+1}) | S_t = s\right]$
- **Policy Iteration**:

We can construct an optimal policy simply by acting greedily with respect to the optimal value function.

Most of the tabular, and iterative methods require an explicit model of an environment with access to the transition functions. In the absence of a model or transition functions, one can use Monet Carlo methods to approximate these probabilities through sampling.

Tabular methods are realizable in settings with small state spaces, where we have access to the perfect model of the environment in the form of MDPs. Tabular methods are also useful in establishing theory and guarantees for various algorithms. However, for most practical applications it is common to use parameterized policies and approximate value functions.

2.2.2 Approximate Value-based methods

In approximate value-based methods, the objective is to learn the weights w of a function $\hat{V}(s; w)$ that best approximates the value function V_π under some policy π.

Predictive Objective: $\overline{\mathrm{VE}}(w) \doteq \sum_{s \in \mathcal{S}} \mu(s)[V_\pi(s) - \hat{V}(s; w)]^2$.

μ is the on-policy distribution, that determines the importance given to states in-terms of the prediction error.

- If v_π is not available, on can use monte-carlo estimates for target updates
- If you use bootstrapping, then the assumption that the gradient is independent of the target becomes invalid (semi gradient methods)
- For linear function approximation, bootstrapping still works robustly
- On-policy control can be done in the same way as tabular case with ϵ-greedy policies
- Off-policy methods use importance sampling to weight the error based on target and behavior policies
- Off-policy methods face divergence in value estimation (due to the difference in state distributions induced by behavior and target policies) when combined with function approximation and boostrapping (deadly triad)

2.2.3 Policy Gradient methods

Policy gradient methods allow for directly learning a parameterized policy that enables action selection without the use of a (action-)value function. Policy gradient methods have the advantage that in many cases the policy space could be simpler to approximate compared to the value function space. We discuss Policy gradient methods in more detail in the next section as it serves as a base for the rest of our work.

2.3 Multi-agent Reinforcement Learning (MARL)

There has been increasing interest in designing RL algorithms for the multiagent setting.

2.3.1 Challenges in MARL

Although Policy Gradient methods and their derivatives have theoretical guarantees in single-agent settings, they are not directly extendable to multi-agent settings due to the non-stationarity of the dynamics. Also, from an algorithm design perspective, the action space explodes exponentially in multi-agent settings making it computationally challenging to apply reinforcement algorithms without decomposing the problem into more managable sub-problems first.

2.3.2 Methods in MARL

Independent RL

- Independent PG
- Centralized Training, Decentralized Execution

Self-play RL

Decentralized RL

2.4 Online Convex Optimization and Mirror Descent

(The brief background provided in this section closely follows the details as presented in [4]. For a more in-depth introduction into Online learning and Online Convex Optimization, please refer to the above work.)

In Online Learning, a learner is tasked with predicting the answer to a set of questions over a sequence of consecutive rounds. We now define an Online learning problem more formally as follows:

Definition 4 *For each round t, given an instance $x_t \in \mathcal{X}$, and a prediction $p_t \in \mathcal{Y}$, a loss function $l(p_t, y_t) \mapsto \mathbb{R}$*

At each round t, a question x_t is taken from an instance domain $\mathcal{X}$, and the learner is required to predict an answer, p_t to this question. After the prediction is made, the correct answer y_t, from a target domain $\mathcal{Y}$ is revealed and the learner suffers a loss $l(p_t, y_t)$. The prediction p_t could belong to $\mathcal{Y}$ or a larger set, $\mathcal{D}$.

The main aim of an online learning algorithm A, is to minimize the cumulative regret of a learner with respect to the best competing hypothesis h^* from the assumed hypothesis class $\mathcal{H}$.

$$Regret_T(h^*) = \sum_{t=1}^{T} l(p_t, y_t) - \sum_{t=1}^{T} l(h^*(x_t), y_t), \tag{2.2}$$

The regret of A with $\mathcal{H}$ is,

$$Regret_T(\mathcal{H}) = \max_{h^* \in \mathcal{H}} Regret_T(h^*) \tag{2.3}$$

A popular framework for studying and designing Online learning algorithms is through Convex optimization. The assumptions made around the framework provides properties that are useful in deriving convergence guarantees. We define a few terms below that are used in this section and the following ones.

Definition 5 (Strongly Smooth) *Given a convex set $\mathcal{X} \in \mathbb{R}^n$, a convex function $f : \mathcal{X} \mapsto \mathbb{R}$ is σ-strongly smooth with respect to a norm $\|.\|$, if $\|\nabla f(x) - \nabla f(y)\|_* \leq \sigma \|x - y\|, \forall x, y \in \mathcal{X}$. For a given constant L, this is also referred to as $L - smooth$.*

The typical structure of online learning expressed as an online convex optimization problem is as follows:

```
AlgorithmOCO
input: a convex set S for t = 1, 2, ...
predict a vector $w_t \in S$
receive a convex loss function $f_t : S \mapsto \mathbb{R}$
```

Reframing Equation 2.2 in terms of convex optimization, we refer to a competing hypothesis here as some vector u from the convex set S.

$$Regret_T(u) = \sum_{t=1}^{T} f_t(w_t) - \sum_{t=1}^{T} f_t(u) \tag{2.4}$$

and similarly, the regret with respect to a set of competing vectors U is,

$$Regret_T(U) = \max_{u \in U} Regret_T(u) \tag{2.5}$$

The set U can be same as S or different in other cases. Here we assume $U = S$ and $S = \mathbb{R}$ unless specified otherwise.

2.4.1 FoReL

Follow-the-Regularized-leader (FoReL) is a classic online learning algorithm that acts as a base in deriving various regret mimization algorithms. The idea of FoReL is to include a regularization term to stabilize the updates in each iteration leading to better convergence behaviors.

The learning rule can be written as,

$$\forall t, w_t = \arg\min_{w \in S} \sum_{i=1}^{t-1} f_i(w) + R(w).$$

where $R(w)$ is the regularization term. The choice of the regularization function lead to different algorithms with varying regret bounds.

2.4.2 Gradient Descent

In the case of linear loss functions with respect to some z_t, i.e., $f_t(w) = \langle w, z_t \rangle$, and $S = \mathbb{R}^d$, if FoReL is run with l_2-norm regularization $R(w) = \frac{1}{2\eta}\|w\|_2^2$, then the learning rule can be written as,

$$w_{t+1} = -\eta \sum_{i=1}^{t} z_i = w_t - \eta z_t \tag{2.6}$$

Since, $\nabla f_t(w_t) = z_t$, this can also be written as, $w_{t+1} = w_t - \eta \nabla f_t(w_t)$. This update rule is also commonly known as Online Gradient Descent. The regret of FoReL run on Online linear optimization with a euclidean-norm regularizer is:

$$Regret_T(U) \leq BL\sqrt{2T}.$$

where $U = u : \|u\| \leq B$ and $\frac{1}{T}\sum_{t=1}^{T} \|z_t\|_2^2 \leq L^2$ with $\eta = \frac{B}{L\sqrt{2T}}$.

Beyond Euclidean regularization, FoReL can also be run with other regularization functions and yield similar regret bounds given that the regularization functions are strongly convex.

Definition 6 *For any σ-strongly-convex function $f : S \mapsto \mathbb{R}$ with respect to a norm $\|.\|$, for any $w \in S$,*

$$\forall z \in \partial f(w), \forall u \in S, f(u) \geq f(w) + \langle z, u - w \rangle + \frac{\sigma}{2}\|u - w\|^2. \tag{2.7}$$

Lemma 1 *For a FoReL algorithm producing a sequence of vectors $w_1, \ldots, w_T$ with a sequence of loss functions $f_1, \ldots, f_T$, for all $u \in S$,*

$$\sum_{t=1}^{T}(f_t(w_t) - f_t(u)) \leq R(u) - R(w_1) + \sum_{t=1}^{T}(f_t(w_t) - f_t(w_{t+1}))$$

2.4.3 Hedge

- what is hedge?

Gradient descent as a FTRL variant

$$P_{\mathcal{X}}(u) = \arg\min_{x \in \mathcal{X}} \|u - x\|. \tag{2.8}$$

Proximal mapping

Definition 7 (Proximal mapping) *Given a function $f : \mathcal{X} \mapsto \mathbb{R}$, the proximal mapping of f is given by:*

$$prox_f(x) = \arg\min_{u \in \mathbb{R}} \left\{ f(u) + \frac{1}{2}\|u - x\|^2 \right\}$$

for any $x \in \mathcal{X}$.

Projected subgradient method

$$x_{k+1} = P_{\mathcal{X}}(x_k - t_k f'(x_k)) \tag{2.9}$$

where $f'(x_k) \in \partial f(x_k)$.

2.4.4 Mirror Descent

Mirror descent is a popular first order optimization algorithm that has seen wide applications in machine learning, and reinforcement learning. Mirror Descent generalizes the notion of projected subgradient method Equation 2.9 to non-euclidean settings through a generalized norm called Bergman Divergence.

Definition 8 (Bergman Divergence) *Given a convex set $\mathcal{X} \subset \mathbb{R}^n$, and a differentiable σ-strongly convex function $\omega : \mathcal{X} \mapsto \mathbb{R}$, the Bregman Divergence associated with the function ω is defined as,*

$$B_\omega(x, y) = \omega(x) - \omega(y) - \langle \nabla\omega(y), x - y \rangle.$$

Bergman Divergence is not a true norm, since it does not satisfy the triangle inequality.

Non-euclidean Proximal Gradient method

$$x_{k+1} = \arg\min_{x \in \mathcal{X}} \left\{ \left\langle \frac{1}{L_k} \nabla f(x_k) - \nabla\omega(x_k), x \right\rangle + \frac{1}{L_k} g(x) + \omega(x) \right\}. \tag{2.10}$$

For an alternate view of deriving Mirror Descent as an improvement of FoReL, please refer to [4][Section 2.6].

CHAPTER 3

MIRROR DESCENT IN REINFORCEMENT LEARNING

Given the generality of the mirror descent algorithm as a first order optimization method, there has been continued efforts to incorporate it as a single/multi-agent reinforcement learning algorithm. Extensive studies of mirror descent under various settings and assumptions help in deriving much needed theoretical guarantees for mirror descent based reinforcement learning algorithms in terms of sample complexity, and convergence rates. In this work, we study two such algorithms, namely MDPO (Mirror Descent Policy Optimization), and MMD (Magnetic Mirror Descent). We first begin with a discussion of Softmax Policy Gradients to set a base framework for the rest of the chapter before moving on to the above algorithms.

3.1 Softmax Policy Gradients

In case of a parameterized policy π_θ, where $\theta \in \Theta$ represent the policy parameter space, the aim of policy gradient methods is to maximize some objective $J(\theta)$. Similar to the value-based approximation methods, the policy can also be directly learnt using gradient ascent. A prototypical performance measure is simply the value of the initial state under the current policy: $J(\theta) = V_{\pi_\theta}(s_0)$. The Policy Gradient Theorem [5, Chapter 13.2] establishes that the gradient of this objective function can be estimated without knowledge of the environment's state distribution as long as it is stationary conditioned on the current policy.

$$\nabla_\theta J(\theta) \propto \sum_s \mu(s) \sum_a q_\pi(s,a) \nabla_\theta \pi_t heta(a|s)$$

Softmax Policy Gradients For discrete action settings, it is common to use a parameterized function $y_\theta(s,a)$ to represent action preferences or *logits*, and the policy is then extracted using a softmax operator on top: $\pi(a|s,\theta) \doteq \frac{e^{y_\theta(s,a)}}{\sum_b e^{y_\theta(s,b)}}$. Softmax parameterization is the most popular form of policy representation in RL under function approximation settings. Policy gradient method with a softmax parameterization is typically referred to as Softmax Policy Gradients (SPG).

Reinforce: The most fundamental policy gradient method *REINFORCE*, uses monte-carlo estimations to approximate the performance measure.

$$\nabla_\theta J(\theta_t)_{|\theta=\theta_t} \propto \mathbb{E}_\pi \left[G_t \nabla_\theta \log \pi_{\theta_t}(S_t, A_t) \right] \tag{3.1}$$

Although Monte-carlo estimates of the returns are unbiased, they can be of high variance. A baseline that is independent of the action can be used to reduce this variance. A popular choice for such a baseline an approximate value function such as the one from value-based approximation methods ($V(s;w)$).

Actor-Critic Methods Apart from using this approximate value function as a baseline, we can also use them to better estimate the peformance measure used in the objective. This leads to *actor-critic* methods, that learn a parameterized policy (called the actor), guided by an approximate value function is referred to as the critic.

3.1.1 Trust-region methods

- Trust region methods - PPO an approximation of TRPO with hueristic objective

3.2 Mirror Descent Policy Optimization

The first method we discuss is the Mirror Descent Policy Optimization [6] (MDPO). MDPO tackles the problem developing a stable reinforcement learning algorithm through the framework of trust-region optimization.

(This seems like it mostly belongs in 3.1.2 and then you need to rework this introduction; thiru: I moved the PG section back to Chapter 2, and it includes an introduction to these methods.)

Due to the presence of a proximal regularization (Needs founddations somewhere), mirror descent in itself can be interpreted as a trust-region optimization method. Building on top of existing theoretical guarantees for mirror descent in tabular RL settings (Which are? need to establish some background either in 2 or here), MDPO derives practical RL algorithms that is performant in function approximation settings. Using the formulation of a parameterized policy from 3.1, MDPO performs an approximation of the following update at each iteration:

$$\pi_{k+1}(.|s) \leftarrow \arg\max_{\pi\in\Pi} \mathbb{E}_{a\sim\pi}[A_{\pi_k}(s,a)] - \frac{1}{t_k} KL(s;\pi,\pi_k) \tag{3.2}$$

using stochastic gradient ascent.

The gradient of the KL component of the above objective for one step of SGD is zero, and hence it is necessary to take multiple gradient steps every iteration to correctly approximate the mirror descent objective. (Make sure you include the citation at least the first time you reference them by name so that the reader can get evrything synced up) Tomar et. al [6], derive on-policy and off-policy variants of MDPO, and in this work we mainly focus on the on-policy algorithm for easier comparison with the other methods discussed in this work, and to remove additional factors involved in off-policy learning (Needs an explanation why). For a parameterized policy π_θ the on-policy MDPO update is given by:

$$\theta_{k+1} \leftarrow \arg\max_{\theta} J(\theta, \theta_k)$$

$$\text{where, } J(\theta, \theta_k) = \mathbb{E}_{s\sim\rho_{\theta_k}}[\mathbb{E}_{a\sim\pi_\theta}[A_{\theta_k}(s,a)] - \frac{1}{t_k} KL(s; \pi_\theta, \pi_{\theta_k}]$$

For m steps MDPO uses the following gradient to update the policy parameters,

$$\nabla_\theta J(\theta, \theta_k)|_{\theta=\theta_k^{(i)}} = \mathbb{E}_{s\sim\rho_{\theta_k} a\sim\theta_k}[\frac{\pi_{\theta_k}^{(i)}}{\pi_{\theta_k}} \nabla_\theta \log \pi_{\theta_k}^{(i)}(a|s) A_{\theta_k}(s,a)] - \frac{1}{t_k}\mathbb{E}_{s\sim\rho_{\theta_k}}[\nabla_\theta KL(s; \pi_\theta, \pi_{\theta_k})_{|\theta=\theta_k^{(i)}}] \tag{3.3}$$

where $i = 0, 1, \ldots, m-1$.

MDPO has strong connections to the other constrained optimization RL algorithms and entropy regularized algorithms. We refer the reader to [6] for a detailed discussion of connections between on-policy MDPO to PPO and TRPO, and off-policy MDPO to SAC.

3.2.1 Tabular MDPO

Normal form games allow for tabular policy representations with direct or softmax parameterizations. Algorithms can typically use exact value estimations for all possible actions (payoffs) given the opponent policy, as opposed to a sample-based expected value estimations. This is referred to as the all-actions setting (also known as *full-feedback* or first-order information setting). In this setting, the gradient computation in Equation 3.3 becomes:

$$\nabla\theta_{|\theta=\theta_k^{(i)}} = \nabla_\theta \sum_{a\in A}[\pi_{\theta_k}^{(i)}(a)A_{\theta_k}(a)] - \frac{1}{t_k}[\nabla_\theta KL(\pi_\theta, \pi_{\theta_k})]$$

We use this formulation of MDPO update for the tabular experiments and the original on-policy formulation for the neural experiments.

3.3 Magnetic Mirror Descent

Another extension of Mirror Descent to reinforcement learning is Magnetic Mirror Descent [7], that attempts to create a unified approach to work well in both single and multiagent settings. This work studies the relation between equilibrium solving and Variational inequalities with composite structures. Taking advantage of this connection, a equilibrium solving algorithm has linear last-iterate convergence guarantees is proposed. Moreover, this approach extends well as a reinforcement learning algorithm in single agent, and multiagent settings. In this section, first we outline the connection between Varational inequalities and equlibrium solving as presented in [7]. Then we outline the Magnetic Mirror Descent algorithm and its convergence properties.

3.3.1 Connection between Variational Inequalities and QREs

A Variational Inequality (VI) problem, written as $VI(Z, F)$ is generally defined as follows:

Definition 9 *Given $\mathcal{Z} \subseteq \mathbb{R}^n$ and mapping $F : \mathcal{Z} \to \mathbb{R}^n$, the variational inequality problem VI($\mathcal{Z}, F$) is to find $z_* \in \mathcal{Z}$ such that,*

$$\langle F(z_*), z - z_* \rangle \geq 0 \quad \forall z \in \mathcal{Z}.$$

The VI problem described above is very general, and as such a wide range of problems can be cast into this framework [8]. We mainly focus on the relation between VI problems with a similar structure, and QREs.

Finding the QRE of a two-player zero-sum game can be represented as the regularized saddle point problem. Given $\mathcal{X} \subseteq \mathbb{R}^n$, $\mathcal{Y} \subseteq \mathbb{R}^m$, and $g_1 : \mathbb{R}^n \mapsto \mathbb{R}$, $g_2 : \mathbb{R}^m \mapsto \mathbb{R}$, find:

$$\min_{x \in \mathcal{X}} \max_{y \in \mathcal{Y}} \alpha g_1(x) + f(x, y) + \alpha g_2(y), \tag{3.4}$$

where g_1, and g_2, are strongly-convex functions. For a two-player zero-sum game, $f(x, y)$ represents the payoff matrix, and g_1, g_2 represent entropy-regularization of the strategies of the two players. (It isn't immediately clear where the entropy and regularization are in (3.3)) The solution (x_*, y_*) to Equation 3.4 has the following first-order optimality conditions:

$$\begin{aligned} &\langle \alpha \nabla g_1(x_*) + \nabla_{x_*} f(x_*, y_*), x - x_* \rangle \geq 0, \forall x \in \mathcal{X}. \\ &\langle \alpha \nabla g_2(y_*) + \nabla_{y_*} f(x_*, y_*), y - y_* \rangle \geq 0, \forall y \in \mathcal{Y}. \end{aligned} \tag{3.5}$$

The optimality conditions Equation 3.5 are equivalent to the optimality conditions of a VI($\mathcal{Z}, G$) with the following composite objective $G = F + \alpha \nabla g$, where $\mathcal{Z} = \mathcal{X} \times \mathcal{Y}$, $F(z) =$

$[\nabla_x f(x,y) - \nabla_y f(x,y)]$, and $\nabla g = [\nabla_y g_1(x), \nabla_x g_2(y)]$. Hence, the solution to the VI: $z^* = (x^*, y^*)$ is also the solution to the saddle point problem stated in Equation 3.4. So, the reguarlized saddle point problem Equation 3.4 of finding QREs can be cast as a VI problem allowing us to tap into the vast array of existing techniques for solving Variational inequalities.

3.3.2 MMD Algorithm

Various algorithms have been proposed to solve the VI problem 9. In particular, the proximal point method has linear last iterate convergence for Variational inequality problems with a strongly monotone operator [9]. This algorithm was extended to composite objectives [10], and to non-euclidean spaces with Bergman divergence as a proximity measure, that allows for non-euclidean proximal regularization [11].

The non-euclidean proximal gradient algorithm, that is more generally applicable to any VI problem with a monotone operator, performs the following update at each iteration:

$$z_{t+1} = \arg\min_{z \in \mathcal{Z}} \eta(\langle F(z_t), z\rangle + \alpha g(z)) + B_\psi(z; z_t). \tag{3.6}$$

where $z_1 \in$ int dom $\psi \cap \mathcal{Z}$, and ψ is a strongly convex function with respect to $\|.\|$ over $\mathcal{Z}$.

The algorithm that is termed Magnetic Mirror Descent (MMD) uses the same update as Equation 3.6 with g taken to be either ψ, or $B_\psi(.; z')$. In the former, ψ is as a strongly convex regularizer that makes the objective smoother and encouranges exploration. In the latter form, B_ψ is another proximity term that forces the iterates (z_{t+1}) to stay close to some *magnet* (z'). For all our discussion, we only consider the former update rule which is more widely applicable. In this case, the main MMD algorithm [7, (Algorithm 3.6)] becomes, (This is an equation, not al algorithm, because it doesn't tell you how to compute the minimumum)

$$z_{t+1} = \arg\min_{z \in \mathcal{Z}} \eta(\langle F(z_t), z\rangle + \alpha\psi(z)) + B_\psi(z; z_t). \tag{3.7}$$

The MMD algorithm using the above update rule has the following linear convergence guarantee.

Theorem 1 *[7, Theorem 3.4] Assuming that the solution z_* to the problem VI $(\mathcal{Z}, F + \alpha\nabla g)$ lies in the int dom ψ, then*

$$B_\psi(z_*; z_{t+1}) \leq \left(\frac{1}{1+\eta\alpha}\right)^t B_\psi(z_*; z_1),$$

if $\alpha > 0$, and $\eta \leq \frac{\alpha}{L^2}$.

3.3.3 Behavioral form MMD

The update rule Equation 3.7 admits closed form in some instances, while requires approximation through gradient updates in other cases. For a parameterized policy π_θ, when ψ is negative entropy it can be restated in RL terms as follows:

$$\pi_{\theta_{k+1}}(s,a) = \arg\max_\theta \mathbb{E}_{s\sim\rho_{\theta_k}}\left[\mathbb{E}_{a\sim\pi_{\theta_k}}[Q_{\theta_k}(s,a)] + \alpha H(\pi_\theta) - \frac{1}{\eta} KL(\pi_\theta, \pi_{\theta_k})\right], \tag{3.8}$$

where $H(\pi_\theta)$ is the entropy of the policy being optimized.

This behavioral form update can also be approximated using gradient updates similar to MDPO. For m steps MMD uses the following gradient to update the policy parameters,

$$\nabla_\theta J(\theta, \theta_k)|_{\theta=\theta_k^{(i)}} = \mathbb{E}_{s\sim\rho_{\theta_k} a\sim\theta_k}[\frac{\pi_{\theta_k}^{(i)}}{\pi_{\theta_k}} \nabla_\theta \log \pi_{\theta_k}^{(i)}(a|s) A_{\theta_k}(s,a)] + \alpha H(\pi_{\theta_k}^{(i)}) - \frac{1}{\eta}\mathbb{E}_{s\sim\rho_{\theta_k}}[\nabla_\theta KL(s;\pi_\theta,\pi_{\theta_k})_{|\theta=\theta_k^{(i)}}] \tag{3.9}$$

where $i = 0, 1, \ldots, m-1$.

Tabular MMD: Similar to tabular MDPO, the gradient computation for behavioral-form MMD in single-state all-action setting becomes:

$$\nabla\theta_{|\theta=\theta_k^{(i)}} = \nabla_\theta \sum_{a\in A} [\pi_{\theta_k}^{(i)}(a) A_{\theta_k}(a)] + \alpha \nabla_\theta H(\pi_{\theta_k}) - \frac{1}{\eta}[\nabla_\theta KL(\pi_\theta, \pi_{\theta_k})] \tag{3.10}$$

3.3.4 Closed-form vs Behavioral-form

MMD with a negative entropy mirror map [7, equation (12)] has the following closed-form:

$$\pi_{k+1} \propto [\pi_t \rho^{\alpha\eta} e^{\eta Q_{\pi_k}}]^{\frac{1}{1+\alpha\eta}} \tag{3.11}$$

To examine the effect of the choice of m, we compare the performance of tabular MMD Equation 3.10 to the above closed-form.

In Fig **??**, we plot the norm of the difference between the closed-form and behavioral form policies at each iteration. It can be seen that the behavioral form approximates the closed form well as expected for most choices of step size and m. For large number of gradient steps, or large step sizes the updates are unstable. For all our tabular experiments, we use a step-size of 0.1, and $m = 10$.

3.3.5 Comparison of MMD to other Mirror Decent based methods

The non-euclidean proximal gradient method Equation 3.6, has strong connections to Mirror Descent [7, Appendix D.3]. TBD: other works also discuss this relationship between mirror descent and proximal gradient methods applied to VIs. Consequently negative entropy based MMD is also equivalent to MDPO with an added entropy regularization as detailed in [7, Appendix L] and as can be seen from Equation 3.2 and Equation 3.8.

(Cite? Also, have you introduced CFR at some point?) Sokota et. al [7] experimentally demonstrate MMD's strong performance in single, and multi-agent settings by evaluating it against popular baselines. In single agent settings MMD's performance is competitive with PPO in Atari and MuJoCo environments. MMD instantiated as a reinforcement learning algorithm performing behavioral form update at each information state performs on the same level as CFR, but relatively worse compared to CFR+.

CHAPTER 4

MODIFIED UPDATES FOR LAST-ITERATE CONVERGENCE

Our main contribution is adapting existing techniques for inducing convergence in multiagent settings with the algorithms discussed in the previous section. (Too brief. Slow down and explain to the reader what the problem is that these modified updates are intended to solve.) Last-iterate convergence of algorithms is an attractive property that has been studied widely in the context of optimization, and equilibirium learning algorithms. While there have been numerous methods proposed for equilibrium solving in games, many of them only guarantee convergence in the ergodic sense, i.e., only the average of their iterates converge to an equilibrium. However, in cases where the strategies are represented using function approximators, this presents a problem as it becomes complicated to maintain a history of past strategies or in terms of averaging them.

The main contribution of our work is in studying a few modifications to the algorithms discussed in the previous section, with the objective of either inducing and speeding up last-iterate convergence in two-player zero-sum settings. More specifically, we study the following three techniques, namely - Neural Replicator Dynamics [12], Extragradient updates [13], and Optimistic gradient updates [14], that have been previously studied in the context of learning in games, and solving saddle point problems. In this section we describe these techniques in more detail and provide our proposed modifications to the algorithms from the previous section.

4.1 Neural Replicator Dynamics (NeuRD)

Consider the problem of learning a parameterized policy π_θ in a single-state all-action problem setting. In such a setting, SPG employs the following update at each iteration t [12, Section A.1]:

$$y_{\theta_t}(a) = y_{\theta_{t-1}}(a) + \eta_t \pi_{\theta_{t-1}}(a)[r_t(a) - \bar{r_t}], \forall a \in A$$

where y represents the logits, $r_t(a)$ is the immediate reward associated with action a, and $\bar{r}_t$ is the average reward of the state.

4.1.1 Replicator Dynamics and No-regret Learning

The above SPG update is equivalent to the instant regret scaled by the current policy. This scaling makes it difficult for SPG to adapt to sudden changes in rewards associated with actions that are already less likely under the current policy. This problem is more evident in multiagent settings where the opponent's policy can change arbitrarily affecting rewards associated with actions even in single-state settings.

Motivated by this observation, Hennes et. al [12] propose to make modifications to SPG using the idea of Replicator dynamics from Evolutionary game theory. Replicator Dynamics defiens operators that describe the dynamics of a population's evolution when attempting to maximize some arbitrary utility function (cite). Replicator dynamics also have a strong connection to no-regret algorithms, and in [12, Statement 1], the authors establish the equivalence between Hedge, and discrete time RD. Due to this equivalence, RD also inherits the property of no-regret algorithms that their time-average policy converges to the Nash equilibrium (cite).

4.1.2 NeuRD

Neural Replicator Dynamics (NeuRD) is an adaptation of discrete-time Replicator Dynamics to reinforcement learning for function approximation settings (Include the math, not just the english). For a parameterized policy π_θ, the NeuRD-update rule is given by:

$$\theta_t \leftarrow \theta_{t-1} + \eta_t \sum_{a'} \nabla_\theta \mathrm{Y}_{\theta_{t-1}}(s, a') A(s, a') \tag{4.1}$$

where $Y_{\theta_{t-1}}$ represent the logits of the parameterized policy at timestep $t-1$, and A represents the advantage value.

The only difference between the NeuRD update, and the SPG update is that the gradient of the parameters are computed directly with respect to the logits. This can be viewed as a modification to SPG making it more adaptive in non-stationary settings. To prevent numerical issues stemming from accumulating the advantages into the logits resulting in unstable gradients, a logit gap based thresholding is proposed in [12]. This thresholding operator is: $\nabla_\theta(f(\theta), \eta, \beta) \doteq [\eta \nabla_\theta f(\theta)] \mathbb{I}\{f(\theta + \eta \nabla_\theta f(\theta)) \in [-\beta, \beta]\}$, where β is the *NeuRD-threshold* that determines how arbitrarily close to 0 or 1 the probabilities can get to. Since most PG methods, including the ones discussed in the previous chapter utilize softmax parameterization, the NeuRD fix is easily extendable to the policy loss component of their respective loss functions without any major changes. In our work, we also apply the NeuRD fix to the policy loss component of the MMD and MDPO loss functions (rewrite this sentence).

4.1.3 Relation to Natural Policy Gradients

4.1.4 Alternating vs Simultaneous gradient updates

As an aside, we wish to discuss two possible update schemes for discrete learning dynamics, namely - Simultaneous and Alternating updates. We also observe that while alternating updates shows average-iterate convergence, simultaneous updates do not converge.

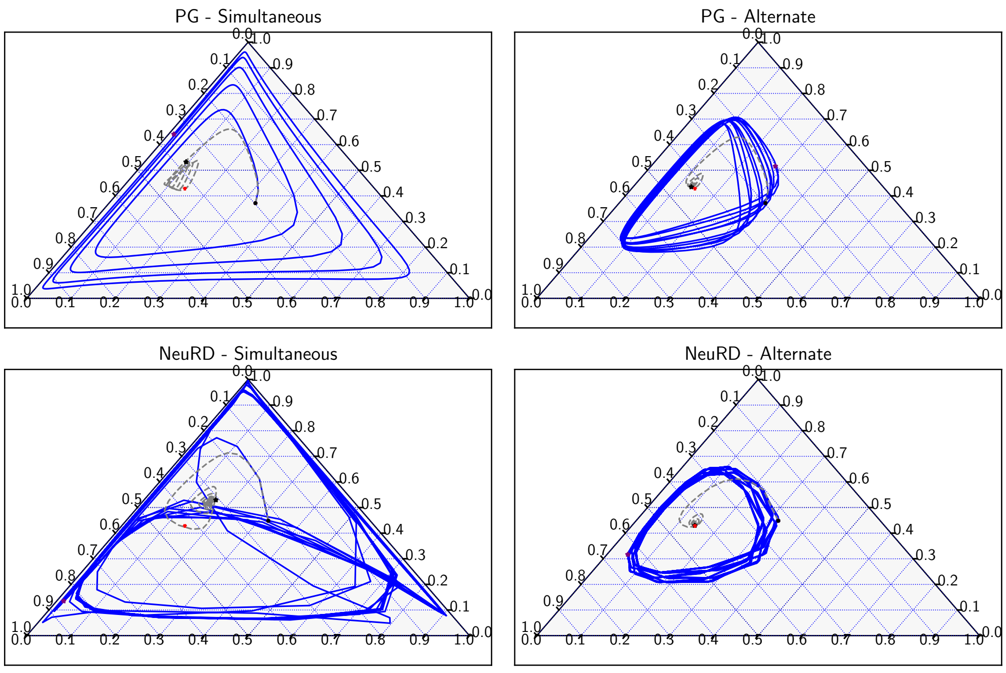


Figure 1: NeuRD converges with alternating updates, but not with simultaneous updates.

We note that all the experiments in [12] also employed alternating updates to evaluate NeuRD and SPG. A similar behavior of simultaneous gradient updates diverging was also observed in [15] for unconstrained min-max problems. They note that the sequence of iterates under alternating updates have a bounded error, and thus the average of the iterates converges to the solution. Based on these observations, we use alternating udpates for all our tabular experiments. Both of these are symmetric update schemes in the sense that both the players perform equal number of updates every iteration. Asymmetric updates have also been studied in a similar context and has been shown to improve the speed of convergence [16].

4.2 Extragradient and Optimistic Gradient

First-order optimization algorithms are studied extensively in varying contexts including solving variational inequalities, saddle-point problems, and convex optimization. While gradient based methods including the classic Arrow-Hurwicz (Since you say EG builds on this a few lines later, you need to tell the reader at least a bit more about what it is) method work well for many optimization problems, for saddle point problems they converge only under strong-convexity even in the unconstrained case [17].

4.2.1 Extragradient (EG)

The Extragradient method was introduced by G. M. Korpelevich [13] as a modification to the Arrow-Hurwicz method for solving convex-concave saddle point problems, and variational inequality problems with strongly monotone operators. The EG algorithm follows the sequence of updates as given below:

$$\begin{aligned}\bar{x}_k &= P_{\mathcal{X}}[x_k - \eta f(x_k, y_k)] \\ \bar{y}_k &= P_{\mathcal{Y}}[y_k + \eta f(x_k, y_k)] \\ x_{k+1} &= P_{\mathcal{X}}[x_k - \eta f(\bar{x}_k, \bar{y}_k)] \\ y_{k+1} &= P_{\mathcal{Y}}[y_k + \eta f(\bar{x}_k, \bar{y}_k)]\end{aligned} \tag{4.2}$$

If f is L-smooth, and $0 < \eta < 1/L$ is the step size, EG has asymptotic last-iterate convergence. (Make sure you have defined the various notions of convergence somewhere, probably in 2 or 3) [10] showed that for VIs with strongly monotone operators or composite VIs the EG method has a linear last-iterate convergence rate. (expand more)

4.2.2 Optimistic gradient (OPT)

Another modification to the Arrow-Hurwicz gradient method was proposed by L. D. Popov in [14], for solving convex-concave saddle point problems. The proposed algorithm, popularly known as **Optimistic updates** can be expressed through the following sequence of updates:

$$\begin{aligned}\bar{x}_k &= P_{\mathcal{X}}[x_k - \eta f(\bar{x}_{k-1}, \bar{y}_{k-1})] \\ \bar{y}_k &= P_{\mathcal{Y}}[y_k + \eta f(\bar{x}_{k-1}, \bar{y}_{k-1})] \\ x_{k+1} &= P_{\mathcal{X}}[x_k - \eta f(\bar{x}_k, \bar{y}_k)] \\ y_{k+1} &= P_{\mathcal{Y}}[y_k + \eta f(\bar{x}_k, \bar{y}_k)]\end{aligned} \tag{4.3}$$

(explain what has changed) Contrast to the EG method, Optimistic updates reuse the gradients of the leading points of the previous iteration $f(\bar{x}_{k-1}, \bar{y}_{k-1})$ instead of computing an *extragradient* in each iteration $f(\bar{x_k}, \bar{y_k})$ as done in Equation 4.2. Variations of the above algorithm have been studied throughout the literature. Most notably, Optimistic Gradient Descent Ascent and Optimistic Multiplicative Weights Update are well-established algorithms that use Optimistic updates and are instances of the more general Optimisitc Mirror Descent.

4.2.3 Optimistic Mirror Descent

[18] studied this algorithm in the context of Online learning with predictable sequences under the name Optimistic Mirror Descent. Optimistic updates coincides with the idea of using a predictor for the performance of next iteration to guide the updates in Online learning []. While this assumption is valid for the problem of learning with predictable sequences, it is also valid in multiagent settings if the objective function is convex and both players use regularized learning algorithms. Within this work, OMD was also studied in the context of zero-sum games with strongly uncoupled dynamics. Optimistic Mirror Descent (OMD) is an extragradient version of Mirror Descent, and OGDA can be shown to be an instantiation of OMD that uses *extrapolation from past.* OMD updates are given by:

$$\begin{aligned} x_{k+1} &= P[x_k - 2\eta f(x_k, y_k) + \eta f(x_{k-1}, y_{k-1})] \\ y_{k+1} &= P[y_k + 2\eta f(x_k, y_k) - \eta f(x_{k-1}, y_{k-1})] \end{aligned} \tag{4.4}$$

[16] studied OMD to tackle the problem of instability in training GANs. The also prove last-iterate convergence for OMD in bilinear games and detail an optimistic version of the Adam optimizer. OGDA has linear last-iterate convergence [16].

OMD was also studied in [15] for the problem of convergence in GANs under the framework of Variational Inequalities, dubbed *extrapolation from the past*. Here, extrapolation refers to the EG algorithm, and Optimism is interpreted as EG with the extrapolation done using the past iterations gradient as an estimate of the extragradient. They also prove convergence for stochastic VI problems with strong monotone operators for the average of the iterates. [19] also study the EG method for training GANs by establishing a framework of *coherence*, and show that EG has last-iterate convergence for coherent VI problems.

Optimism is often interpreted as an adaptation of the Extragradient method with the idea to avoid the additional call to a gradient oracle by estimating the extrapolation using the previous iteration's gradient. Extragradient and Optimistic Gradient Descent Ascent methods have been shown to be approximations of proximal-point method for solving saddle point methods [20].

In our work, we study the effects of adapting extragradient and optimistic updates into the algorithms discussed in the previous chapter for 2p0s games.

CHAPTER 5

TABULAR EXPERIMENTS

We now evaluate our proposed methods experimentally in both tabular and function approximation settings as approximate equilibrium solvers. Through the experiments, we aim to answer the following questions:

1. What is the last-iterate and average-iterate convergence behavior of these algorithms?
2. How does the addition of NeuRD-fix, Extragradient updates, and Optimistic upates affect the convergence rate of these algorithms (alone and in combination)?
3. Do these performance improvements scale well with the size of the game?

MMD has theoretical converegence guarantees only as a QRE solver, but shows a strong empirical performance for finding approximate Nash Equilibriums. In this work our main focus is convergence to the Nash Equilibrium, and as such we focus our main experimental results for the same. We provide some additional results for the performance of different algorithms as QRE solvers in the appendix.

5.1 Experimental Domains

We evaluate the algorithms on two normal form games namely, Perturbed RPS and Matching Pennies.

(Not sure what goes here. This seems likely to at least in part duplicate 2 as laid out, so be careful about that)

	H	T
H	1	-1
T	-1	1

(a) Matching Pennies Payoffs.

	R	P	S
R	0	-v	v
P	v	0	-1
S	-v	1	0

(b) Perturbed RPS Payoffs.

TABLE I: Tabular NFG Payoffs

Matching Pennies Matching Pennies is a classic example from Game Theory that demonstrates decision making in multiagent settings in the simplest form. In matching pennies, two players toss coins independently and the row player wins if they both land on the same side, the column player wins otherwise. The payoffs for Matching Pennies are shown in Table I.

Perturbed RPS Perturbed/Biased RPS is a modified version of the canonical normal form game Rock-Paper-Scissors with biased payoffs for each action. Table I shows the payoff matrix for Perturbed RPS.

Being symmetric matrix games, both games have a unique Nash Equilibrium: Perturbed RPS $(\frac{1}{7}, \frac{3}{7}, \frac{3}{7})$, Matching Pennies $(\frac{1}{2}, \frac{1}{2})$.

5.2 Evaluation Metrics

Our main focus being the convergence behaviors and rates, we need a notion of distance from the equilibrium point to measure the performance of these algorithms.

5.2.1 Divergence to the equilibrium

In settings with a known unique equilibrium, we can compute the distance of the current policy to the known equilibrium using a measure of distance in the policy space such as the KL-Divergence. Given the policy at iteration t, pi_t, and an equilibrium policy π_*, the metric we measure is: $KL(\pi_t||\pi_*) = \sum_{a \in A} \pi_t(a) \log \left(\frac{\pi_t(a)}{\pi_*(a)} \right)$.

5.2.2 Exploitability

In general, there might not be a unique Nash equilibrium and we might not know which equilibrium point the current policy is converging towards. This makes it tricky to use a direct measure of distance as the above metric. Exploitability is another metric that is commonly used as a notion of optimality in game theory. Exploitability measures the gain in value a player can achieve by deviating from the current policy. We measure the value that a worst-case opponent can achieve by keeping the current policy fixed by computing a best response at every state. The difference between the value that this best-response opponent can acheive and the game value is the exploitability of the current policy.

- Exploitability formal expression in terms of best responses, and value.

5.3 Experiment Setup

(something wrong; this sentence doesn't parse. Has enough facts in it that it is probably better to split into multiple sentences.) The policies that are logit-parameterized with a softmax projection and all the algorithms use full-feedback gradient updates as discussed in Chapter 3. For all the algorithms, we train both the players for 5000 training steps with alternating updates using the exact payoff vectors. For the EG variants, we train the players for only 2500 steps as they use two gradient computation per step for a fair comparison. We use the $m = 10$ gradient steps per iteration with a learning rate of 0.1 for all the runs. (Explain how this was chosen as you did for other parameters below.) For MMD and MDPO, we anneal the temperature (entropy coefficient) with the schedule $\alpha_t = 1/\sqrt{t}$. For the KL-coefficient, we use different schedules for MMD ($\eta_t = \max(1/\sqrt{t}, 0.2)$), and MDPO ($\eta_t = \max(1 - t/T, 0.2)$). MDPO's schedule is motivated by mirror-descent theory, while MMD's schedule is closer to the type of annealing schedule used for the original MMD experiments found through a hyperparameter sweep. For both of these methods we cap the KL-coefficient at 0.2 because very low values destabilize the updates, especially for the NeuRD version of the algorithms.

5.4 Results

(iankash: Currently the writing here is very choppy. Even when presenting results like this you should be telling the reader a story, not just spitting a list of facts at them.)

(iankash: Keep this as just part of 5.3)

(iankash: Relatedly, one plot with 21 lines in it is illegible (at least in places). Can you break this giant plot into multiple plots from the same data that tell each part of the story in an easier to digest way?)

(iankash: Seems like we should also have plots for the average iterate?) (iankash: Figures are missing x-axis label; Explain EG correction in them)

We plot the last-iterate convergence in terms of both evaluation metrics in Figure 2 as a function of the iterations. We also plot the average-iterate convergence for all the methods in Figure 3. As mentioned in the experiment setup, the experiment was run for only 2500 steps and one iteration counts as 2. “No mod.” indicates the base version of these algorithms without any modifications that serves as a baseline for comparison.

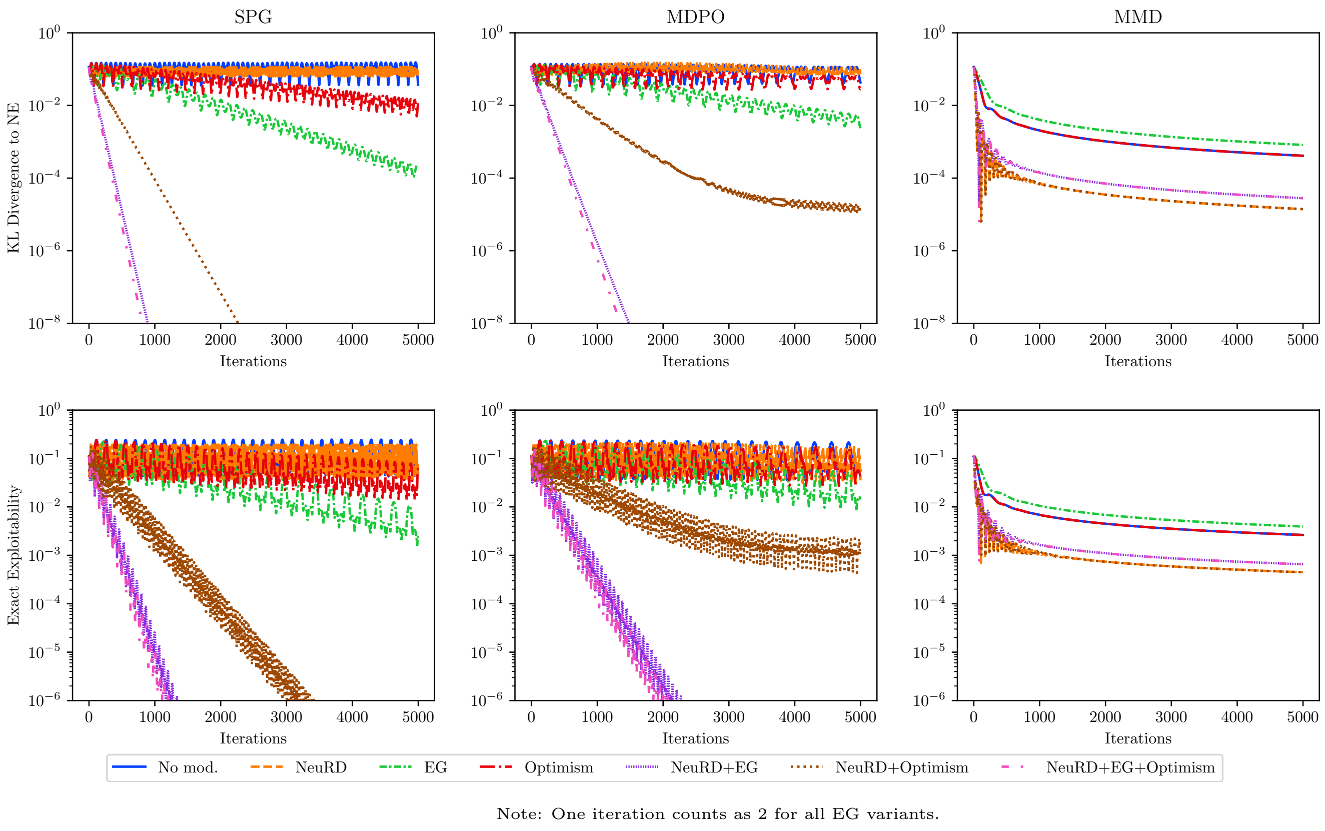
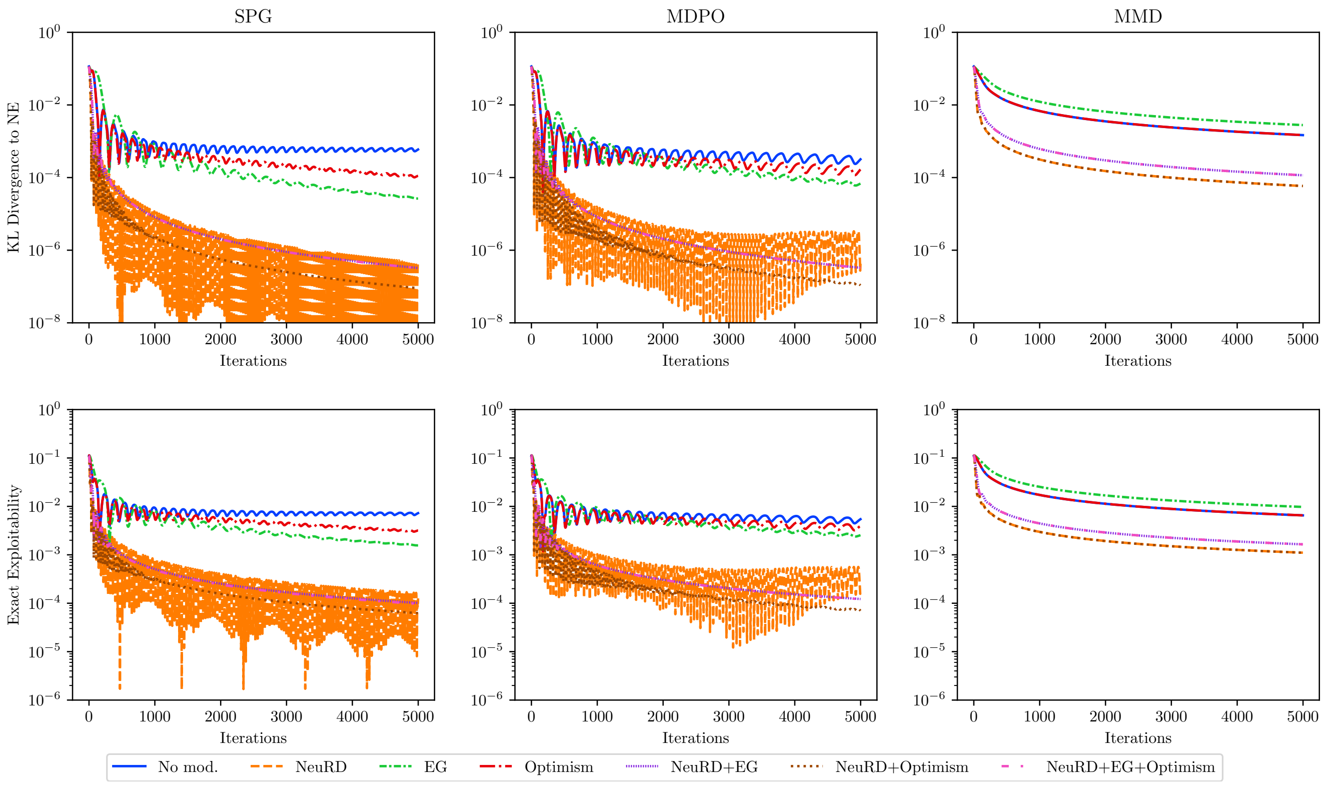


Figure 2: Last-iterate converegence in PerturbedRPS.



Note: One iteration counts as 2 for all EG variants.

Figure 3: Average-iterate converegence in PerturbedRPS.

There are many observations that can be made from the performances of these algorithms in the presence of these modifications. We group these observations by the modifications, and contrast their effects on SPG, MDPO, and MMD.

NeuRD Fix: SPG, and MDPO do not have last-iterate or average-iterate convergence guarantees, and the same is observed experimentally for their *No mod.* baselines. The NeuRD variants of SPG, and MDPO have display average-iterate convergence Figure 3, as guaranteed by its no-regret properties. However, as expected the addition of NeuRD fix does not induce last-iterate convergence for SPG, and MDPO. For MMD, which already has average iterate convergence guarantees, it speeds up the convergence speed. In fact, NeuRD-fix is the only modification that has a significant impact in terms of convergence for MMD.

EG, and Optimism: The baseline version of MMD has last-iterate convergence as guaranteed by its theory. As seen in Figure 2, EG, and Optimistic updates either induces last-iterate convergence or speeds it up in most cases. Interestingly, there are a few algorithm specific differences in terms of the improvement gained through these modifications. EG updates induce last-iterate convergence in SPG, and MDPO. On the contrary, the addition of EG updates slows down MMD. While Optimistic updates work well in SPG, they do not help with convergence for MDPO.

Combining the modifications: We also experiment with combination of these modifications to see if this leads to a superior performance. Combining EG, and Optimism leads to marginal or no improvement, as they both perform a similar modification of extrapolating the gradients. However, we can see a significant improvement in convergence speeds when EG, and

Optimistic updates are combined with the NeuRD fix. Hence, contrary to trust-region constraints, the NeuRD-fix provides an improvement that is tangential to EG/Optimistic Updates that also help in reducing the cycling around the equilibrium. For both SPG, and MDPO, 'NeuRD + EG + Optimism' provides the fastest last-iterate convergence. Whereas, for MMD, 'NeuRD', 'NeuRD + Optimism' are the fastest variants.

Finally, we highlight that the best performing variants are on top of SPG as opposed to MDPO, or MMD as can be seen from Figure 2. This is surprising as the latter methods were proposed as improvements over SPG in single, and multi-agent settings. But, this indicates that EG/Optimistic updates provide a better last-iterate convergence rates compared to adaptive entropy regularization with trust-region constraints. While the convergence rates of EG, and Optimistic updates are well-studied, it is not the case for adaptive regularization. Most notably, Optimistic-NeuRD (SPG + NeuRD + Optimism) exhibits a faster last-iterate convergence that is straightforward to extend to more complicated settings.

The tabular experiments, and the above observations provide some answers to questions 1, and 2. In the next section, we use the above observations to extend these results to function approximation settings.

CHAPTER 6

DEEP MULTI-AGENT RL EXPERIMENTS

All of the algorithms discussed in this work typically implemented in function approximation settings and it is of interest to see if the performance improvements observed carry over when both the policy parameters, and the value approximation are represented using neural networks.

Expanding on our the observations from the tabular experiments, we proceed to evaluate the algorithms in large-scale 2p0s games to answer question 3. For the neural experiments, we study the effect of the NeuRD-fix, and the utilization of Optimistic-variants of popular optimizers (Optimistic-SGD, and Optimistic-Adam) when paired with deep RL algorithms. Extragradient updates require more orchestration when learning through self-play, are more restrictive in how the training is performed. Hence, we do not study the EG variants in this section.

6.1 Experimental Domains

For these experiments we choose three 2p0s EFGs, namely - Kuhn Poker, Abrupt Dark Hex, and Phantom Tic-tac-toe.

Kuhn Poker Kuhn Poker is a smaller extensive form game that allows for more introspection and exact exploitability computation.

Abrupt Dark Hex Abrupt Dark Hex

Phantom Tic-Tac-Toe Phantom Tic-Tac-Toe (Phantom TTT)

We train all the algorithms through self-play reinforcement learning.

6.2 Evaluation Metrics

The measures used in tabular experiments are harder to compute for large-scale EFGs. Distance to an equilibrium point is harder to compute due to the absence of a unique equilibrium point. For larger games, the computation of exact best responses to measure the exact exploitability is prohibitive due to the large state space. However, we can approximate the exploitability computation if we can approximate the best response for a given fixed policy.

6.2.1 Approximate Exploitability

We can learn a local best response by training a best response agent against the fixed policy we wish to evaluate [21]. Then the exploitability can be approximated by sampling trajectories and measuring the average reward the best response agent acheived against the exploited policy. In our experiments, we train the best response agent against the fixed joint-policy learn through self-play. Let π_{BR} be a learnt best-response approximator, and $\pi_{fixed} = (\pi_1, \pi_2)$ be the joint policy to be exploited, then the approximate exploitability is given by:

$$\text{Exp}_{appx}(\pi_{BR}, \pi_{fixed}) = \frac{1}{N}\left[\sum_{i=1}^{N/2} \mathbb{E}_{a\sim\pi_1,\pi_{BR}}[R] + \sum_{i=1}^{N/2} \mathbb{E}_{a\sim\pi_{BR},\pi_2}[R]\right] \tag{6.1}$$

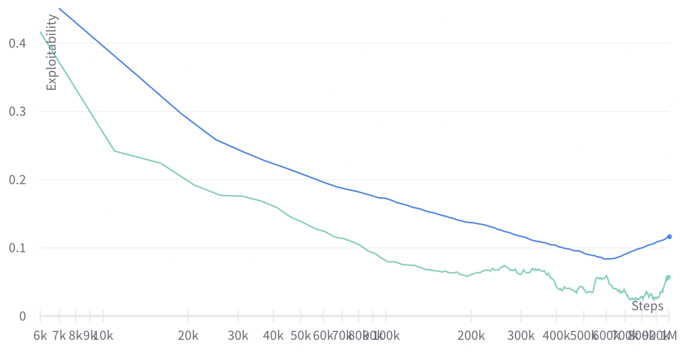
TBD: rephrase the equation in terms of cumulative episodic rewards.

6.3 Experiment Setup

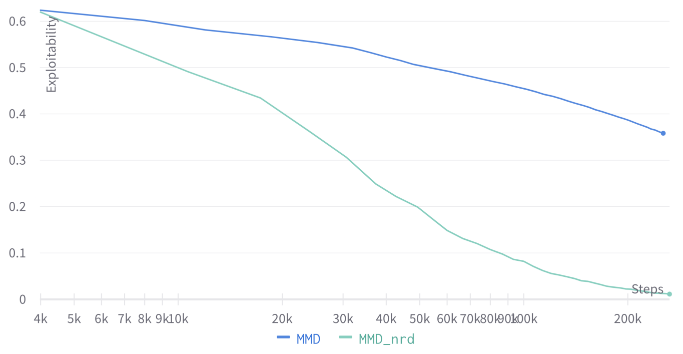
Our experiment setup is akin to that of [7]. We use RLLib [22] for implementing the algorithms, and interface with OpenSpiel's [23] game implementations using RLLib's environment adapter. For all the experiments we a use 2-layered MLP with (128, 128) hidden units for the policy and the value networks without shared parameters. GAE details We use GAE for computing the advantage estimates. For all the experiments, we train a DQN best-response agent against the fixed joint-policy to compute approximate exploitability. For the smaller games (Kuhn Poker, and Abrupt Dark Hex 2×2), we train the agents in self-play for 1M steps and report exact exploitability across the training iterations. We also train the DQN agent for 1M steps against the fixed joint-policy obtained at the end of training to compute the approximate exploitability. For Abrupt Dark Hex 3×3, and Phantom TTT we train the agents for 5M steps and only report the approximate exploitability of the final joint-policy by training the DQN agent for 5M steps. For all approximate exploitability evaluations, we evaluate the agents for $N = 1000$ episodes with the DQN agent starting first in 500 episodes, and the exploitee agent starting first in 500 episodes.

6.4 Results

Figure 4 plots the exact exploitabililty of the joint policy as a function of the iterations, and Figure 4 shows the approximate exploitability at the end of training for these games. These results show correspondence between exact and approximate exploitability as expected qualifying the latter as a valid evaluation metric for the larger games. In agreement to the tabular experiments, the NeuRD versions of these algorithms have a better performance.



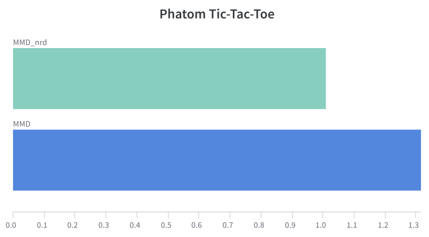
(a) Kuhn Poker



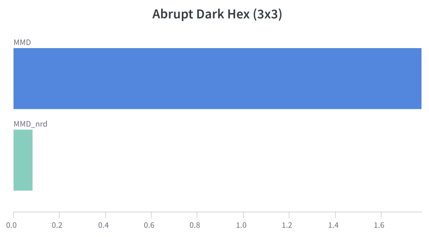
(b) Abrupt Dark Hex (2×2) (1M steps version to be added)

Figure 4: Performance in small EFGs, measured by exact exploitability.

Figure 5 shows improvement in performance by applying the NeuRD-fix in Abrupt Dark Hex (3×3) and Phantom TTT, as measured by the approximate exploitability.



(a) Kuhn Poker



(b) Abrupt Dark Hex (3×3)

Figure 5: Performance in larger EFGs, measured by approximate exploitability.

We also evaluate these algorithms by having the trained agents play against each other in a head-to-head manner.

CHAPTER 7

DISCUSSION

NeuRD-fix: The NeuRD-loss has also been previously applied on top of algorithms to improve performance or induce convergence in competitive and cooperative settings. Perolat et. al, [24] introduced Regularized Nash Dynamics (R-NAD) that utilizes NeuRD as a fixed-point approximator along with adaptive regularization [25] to achieve last-iterate convergence in the game of Stratego. Chhablani et. al, [26] motivated the choice of an advantage baseline in COMA [27] through no-regret literature and demonstrated that applying the NeuRD loss to COMA's objective improved its performance even in cooperative settings such as identical-interest games. The improved performance of adapting the NeuRD-fix with mirror-descent based methods, and other techniques for last-iterate convergence is also evident from our experimental evaluations. Through this, we reinforce the idea that the NeuRD-loss can be more generally adapted into loss functions of various algorithms in multi-agent settings. is there a benefit to adding NeuRD in single-agent setting? even if the reward function is not dynamic.

Entropy Regularization: As noted in the tabular experiments, and as observed from the performance of MDPO in the deep MARL experiments, there exist faster methods to induce last-iterate convergence that adaptive regularization.

Trust-region constraints: Trust-region constraints form the basis of state-of-the-art RL algorithms like PPO. The presence of a relatively strongly-convex proximal operator is necessary for deriving algorithms with strong performance guarantees.

CHAPTER 8

CONCLUSION

APPENDICES

Appendix A

SOME ANCILLARY STUFF

Ancillary material should be put in appendices.

Appendix B

SOME MORE ANCILLARY STUFF

CITED LITERATURE

1. Tuyls, K. and Weiss, G.: Multiagent Learning: Basics, Challenges, and Prospects. *AI Magazine* , 33(3):41–41, September 2012.

2. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y.: Generative Adversarial Nets. In *Advances in Neural Information Processing Systems* , volume 27. Curran Associates, Inc., 2014.

3. Shoham, Y. and Leyton-Brown, K.: *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations* . Cambridge University Press, December 2008.

4. Shalev-Shwartz, S.: Online Learning and Online Convex Optimization. *Foundations and Trends® in Machine Learning* , 4(2):107–194, 2012.

5. Sutton, R. S. and Barto, A. G.: *Reinforcement Learning: An Introduction* . Adaptive Computation and Machine Learning Series. Cambridge, Massachusetts, The MIT Press, second edition edition, 2018.

6. Tomar, M., Shani, L., Efroni, Y., and Ghavamzadeh, M.: Mirror Descent Policy Optimization. In *International Conference on Learning Representations* , January 2022.

7. Sokota, S., D'Orazio, R., Kolter, J. Z., Loizou, N., Lanctot, M., Mitliagkas, I., Brown, N., and Kroer, C.: A Unified Approach to Reinforcement Learning, Quantal Response Equilibria, and Two-Player Zero-Sum Games. In *The Eleventh International Conference on Learning Representations* , February 2023.

8. eds. F. Facchinei and J.-S. Pang *Finite-Dimensional Variational Inequalities and Complementarity Problems* . Springer Series in Operations Research and Financial Engineering. New York, NY, Springer, 2004.

9. Rockafellar, R. T.: Monotone Operators and the Proximal Point Algorithm. *SIAM Journal on Control and Optimization* , 14(5):877–898, August 1976.

10. Tseng, P.: On linear convergence of iterative methods for the variational inequality problem. *Journal of Computational and Applied Mathematics* , 60(1):237–252, June 1995.

11. Tseng, P.: Approximation accuracy, gradient methods, and error bound for structured convex optimization. *Mathematical Programming* , 125(2):263–295, October 2010.

12. Hennes, D., Morrill, D., Omidshafiei, S., Munos, R., Perolat, J., Lanctot, M., Gruslys, A., Lespiau, J.-B., Parmas, P., Duenez-Guzman, E., and Tuyls, K.: Neural Replicator Dynamics, February 2020.

13. Korpelevich, G. M.: The extragradient method for finding saddle points and other problems. *Matecon* , 12:747–756, 1976.

14. Popov, L. D.: A modification of the Arrow-Hurwicz method for search of saddle points. *Mathematical notes of the Academy of Sciences of the USSR* , 28(5):845–848, November 1980.

15. Gidel, G., Berard, H., Vignoud, G., Vincent, P., and Lacoste-Julien, S.: A Variational Inequality Perspective on Generative Adversarial Networks, August 2020.

16. Daskalakis, C., Ilyas, A., Syrgkanis, V., and Zeng, H.: Training GANs with Optimism, February 2018.

17. He, B., Xu, S., and Yuan, X.: On Convergence of the Arrow–Hurwicz Method for Saddle Point Problems. *Journal of Mathematical Imaging and Vision* , 64(6):662–671, July 2022.

18. Rakhlin, A. and Sridharan, K.: Optimization, Learning, and Games with Predictable Sequences, November 2013.

19. Mertikopoulos, P., Lecouat, B., Zenati, H., Foo, C.-S., Chandrasekhar, V., and Piliouras, G.: Optimistic Mirror Descent in Saddle-Point Problems: Going the Extra (Gradient) Mile. 2019.

20. Mokhtari, A., Ozdaglar, A., and Pattathil, S.: A Unified Analysis of Extra-gradient and Optimistic Gradient Methods for Saddle Point Problems: Proximal Point Approach. In *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics* , pages 1497–1507. PMLR, June 2020.

21. Timbers, F., Bard, N., Lockhart, E., Lanctot, M., Schmid, M., Burch, N., Schrittwieser, J., Hubert, T., and Bowling, M.: Approximate Exploitability: Learning a Best Response. In *Thirty-First International Joint Conference on Artificial Intelligence* , volume 4, pages 3487–3493, July 2022.

22. Liang, E., Liaw, R., Moritz, P., Nishihara, R., Fox, R., Goldberg, K., Gonzalez, J. E., Jordan, M. I., and Stoica, I.: RLlib: Abstractions for Distributed Reinforcement Learning, June 2018.

23. Lanctot, M., Lockhart, E., Lespiau, J.-B., Zambaldi, V., Upadhyay, S., Pérolat, J., Srinivasan, S., Timbers, F., Tuyls, K., Omidshafiei, S., Hennes, D., Morrill, D., Muller, P., Ewalds, T., Faulkner, R., Kramár, J., De Vylder, B., Saeta, B., Bradbury, J., Ding, D., Borgeaud, S., Lai, M., Schrittwieser, J., Anthony, T., Hughes, E., Danihelka, I., and Ryan-Davis, J.: OpenSpiel: A Framework for Reinforcement Learning in Games, September 2020.

24. Perolat, J., de Vylder, B., Hennes, D., Tarassov, E., Strub, F., de Boer, V., Muller, P., Connor, J. T., Burch, N., Anthony, T., McAleer, S., Elie, R., Cen, S. H., Wang, Z., Gruslys, A., Malysheva, A., Khan, M., Ozair, S., Timbers, F., Pohlen, T., Eccles, T., Rowland, M., Lanctot, M., Lespiau, J.-B., Piot, B., Omidshafiei, S., Lockhart, E., Sifre, L., Beauguerlange, N., Munos, R., Silver, D., Singh, S., Hassabis, D., and Tuyls, K.: Mastering the Game of Stratego with Model-Free Multiagent Reinforcement Learning. *Science* , 378(6623):990–996, December 2022.

25. Perolat, J., Munos, R., Lespiau, J.-B., Omidshafiei, S., Rowland, M., Ortega, P., Burch, N., Anthony, T., Balduzzi, D., Vylder, B. D., Piliouras, G., Lanctot, M., and Tuyls, K.: From Poincaré Recurrence to Convergence in Imperfect Information Games: Finding Equilibrium via Regularization. In *Proceedings of the 38th International Conference on Machine Learning* , pages 8525–8535. PMLR, July 2021.

26. Chhablani, C. and Kash, I. A.: Counterfactual Multiagent Policy Gradients and Regret Minimization in Cooperative Settings. In *AAAI* , 2021.

27. Foerster, J., Farquhar, G., Afouras, T., Nardelli, N., and Whiteson, S.: Counterfactual Multi-Agent Policy Gradients. *Proceedings of the AAAI Conference on Artificial Intelligence* , 32(1), April 2018.