Neural Auto-Curricula

Xidong Feng[*,1], **Oliver Slumbers**[*,1], **Ziyu Wan**[2], **Bo Liu**[3],
Stephen McAleer [4], **Ying Wen**[2], **Jun Wang**[1], **Yaodong Yang**[†,5]
[1]University College London, [2]Shanghai Jiao Tong University,
[3]Institute of Automation, CAS, [4]University of California, Irvine,
[5]Institute for AI, Peking University

Abstract

When solving two-player zero-sum games, multi-agent reinforcement learning (MARL) algorithms often create populations of agents where, at each iteration, a new agent is discovered as the best response to a mixture over the opponent population. Within such a process, the update rules of *"who to compete with"* (i.e., the opponent mixture) and *"how to beat them"* (i.e., finding best responses) are underpinned by manually developed game theoretical principles such as fictitious play and Double Oracle. In this paper[1], we introduce a novel framework—Neural Auto-Curricula (NAC)—that leverages meta-gradient descent to automate the discovery of the learning update rule without explicit human design. Specifically, we parameterise the opponent selection module by neural networks and the best-response module by optimisation subroutines, and update their parameters solely via interaction with the game engine, where both players aim to minimise their exploitability. Surprisingly, even without human design, the discovered MARL algorithms achieve competitive or even better performance with the state-of-the-art population-based game solvers (e.g., PSRO) on Games of Skill, differentiable Lotto, non-transitive Mixture Games, Iterated Matching Pennies, and Kuhn Poker. Additionally, we show that NAC is able to generalise from small games to large games, for example training on Kuhn Poker and outperforming PSRO on Leduc Poker. Our work inspires a promising future direction to discover general MARL algorithms solely from data.

1 Introduction

Two-player zero-sum games have been a central interest of the recent development of multi-agent reinforcement learning (MARL) [60, 2, 53]. In solving such games, a MARL agent has a clear objective: to minimise the worst case performance, its exploitability, against any potential opponents. When both agents achieve zero exploitability, they reach a Nash equilibrium (NE) [36], a classical solution concept from Game Theory [34, 10]. Even though this objective is straightforward, developing effective algorithms to optimise such an objective often requires tremendous human efforts. One effective approach is through iterative methods, where players iteratively expand a population of agents (a.k.a *auto-curricula* [26]) where at each iteration, a new agent is trained and added to the player's strategy pool. However, within the auto-curricula process, it is often non-trivial to design effective update rules of *"who to compete with"* (i.e., opponent selections) and *"how to beat them"* (i.e., finding best responses). The problem becomes more challenging when one considers additional requirements such as generating agents that have behavioural diversity [58, 37, 3, 28] or generalisation ability [47, 38].

[1]*Equal contributions. †Corresponding to <yaodong.yang@pku.edu.cn>. Code released at https://github.com/waterhorse1/NAC

35th Conference on Neural Information Processing Systems (NeurIPS 2021).

One effective approach to designing auto-curricula is to follow game theoretical principles such as fictitious play [5] and Double Oracle (DO) methods [33, 11]. For example, in the DO dynamics, each player starts from playing a sub-game of the original game where only a restricted set of strategies are available; then, at each iteration, each player will add a best-response strategy, by training against a NE mixture of opponent models at the current sub-game, into its strategy pool. When an exact best response is hard to compute, approximation methods such as reinforcement learning (RL) methods are often applied [25]. One of the potential downsides of this approach is, under approximate best response methods, one no longer maintains the theoretical properties of DO and a worthwhile avenue of exploration is to find auto-curricula that are more conducive to these approximation solutions.

Apart from following game theoretical principles, an appealing alternative approach one can consider is to automatically discover auto-curricula from data generated by playing games, which can be formulated as a meta-learning problem [13]. However, it remains an open challenge whether it is feasible to discover fundamental concepts (e.g., NE) entirely based on data. If we can show that it is possible to discover fundamental concepts from scratch, it opens the avenue to trying to discover fundamentally new concepts at a potentially rapid pace. Although encouraging results have been reported in single-agent RL settings showing that it is possible to meta-learn RL update rules, such as temporal difference learning [38, 56, 57], and the learned update rules can generalise to unseen tasks, we believe discovering auto-curricula in multi-agent cases is particularly hard. Two reasons for this are that the discovered auto-curriculum itself directly affects the development of the agent population, and each agent involves an entire training process, which complicates the meta-learning process.

Albeit challenging, we believe a method capable of discovering auto-curricula without explicit game theoretic knowledge can potentially open up entirely new approaches to MARL. As a result, this paper initiates the study on discovering general-purpose MARL algorithms in two-player zero-sum games. Specifically, our goal is to develop an algorithm that learns its own objective (i.e., the auto-curricula) solely from environment interaction, and we offer a meta-learning framework that achieves such a goal. Our solution framework – *Neural Auto-Curriculum (NAC)* – has two promising properties. Firstly, it does not rely on human-designed knowledge about game theoretic principles, but instead lets the meta-learner decide what the meta-solution concept (i.e., who to compete with) should be during training. This property means that our meta-learner shares the ability of game theoretic principles in being able to accurately evaluate the policies in the population. Secondly, by taking the best-response computation into consideration, NAC can *end to end* offer more *suitable* auto-curricula within the approximate best-response scenario compared with previous approaches; this is particularly important since an exact best-response oracle is not always available in practice.

Our empirical results show that NAC can discover meaningful solution concepts alike NE, and based on that build effective auto-curricula in training agent populations. In multiple different environments, the discovered auto-curriculum achieves the same performance or better than that of PSRO methods [25, 3]. We additionally evaluate the ability of our discovered meta-solvers to generalise to unseen games of a similar type (e.g., training on Kuhn Poker and testing on Leduc Poker), and show that the auto-curricula found on a simple environment is able to generalise to a more difficult one. To the best of our knowledge, this is the first work that demonstrates the possibility of discovering an entire auto-curriculum in solving two-player zero-sum games, and that the rule discovered from simple domains can be competitive with human-designed algorithms on challenging benchmarks.

2 Related Work

Whilst it is theoretically possible to solve for NE in two-player zero-sum games via linear programming (LP) [34] in polynomial time [51], this is a strictly limited approach. For example, when the action space become prohibitively large, or continuous, using LP becomes intractable; other approximation methods such as fictitious play (FP) [5], DO [33, 11] or PSRO [25, 32, 31] methods are required. These methods all follow iterative best-response dynamics where at each iteration, a best response policy is found against a previous aggregated policy (e.g., DO/PSRO applies a sub-game NE, FP applies time-average strategy). Under this general iterative framework, other solution concepts include Rectified NE [3], α-Rank [35, 59] or no-regret algorithms [11]. In this paper, instead of following any existing game theoretic knowledge, we try to discover effective solution concepts solely through environment interactions via meta-learning techniques.

Meta-learning, also known as learning to learn, has recently gained increased attention for successfully improving sample efficiency in regression, classification and RL tasks. MAML [13] meta-learns

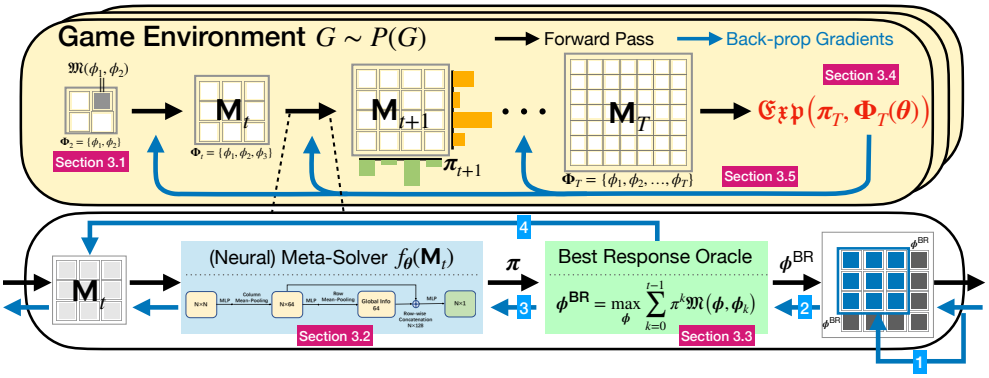

Figure 1: NAC. The top box illustrates the training process where we iteratively expand $\mathbf{M}_t$ with best responses to $f_{\boldsymbol{\theta}}(\mathbf{M}_t)$, and then calculate the exploitability (in red). The bottom box illustrates expanding the population by finding ϕ^{BR}. The blue arrows show how the gradients pass backwards, from the exploitability through the best-response trajectories. The gradient paths [1], [2] refer to the two gradient terms in Eq. (8) and the gradient paths [3], [4] refer to the two gradient paths in Eq. (7).

model parameters by differentiating the learning process for fast adaptation on unseen tasks. PROMP [42] theoretically analyses the MAML-RL formulation and addresses the biased meta-gradient issue. ES-MAML [48] bypasses the Hessian estimation problem by leveraging evolutionary strategies for gradient-free optimisation. RL^2 [12] and L2RL [54] formulate the meta-learning problem as a second RL procedure but update the parameters at a "slow" pace. Recently, there is a trend to meta-learn the algorithmic components of RL algorithms, such as the discount factor [57], intrinsic rewards [65, 64], auxiliary tasks [52, 66], objective functions for value/policy networks [4, 56, 22], off-policy update targets [62], and the bootstrapping mechanism [38]. Their success is built on the *meta-gradient* technique, which leverages gradient descent on the sequence of gradient descent updates resulting from the choice of objective function. The meta-learned RL algorithms demonstrate promising generalisation capability in solving different tasks. Apart from meta-gradients, evolutionary strategies (ES) have also been successfully applied [20]. Our paper, in comparison, has a parallel goal to these prior arts: to discover MARL algorithms that are effective for solving various two-player zero-sum games, and to demonstrate their generalisation ability across different games.

So far, there have been very few attempts to conduct meta-learning in multi-agent settings [60]. MNMPG [46] applied meta-gradients for training credit assignment modules for better decomposing the value network [49, 61]. Meta-PG [1] was proposed as a meta-gradient solution to continuous adaptations for non-stationary environments. Building on the opponent modelling technique called LOLA [15, 14], Meta-MAPG [21] extends Meta-PG [1] to multi-agent settings by considering both the non-stationarity from the environment and from the opponent's learning process. Our work rather focuses on automatically discovering multi-agent auto-curricula through meta-learning, without explicit game theoretic knowledge, that leads to general MARL algorithms for solving two-player zero-sum games.

3 Multi-Agent Meta-Learning Framework

In this section, we discuss our meta-learning framework NAC for discovering multi-agent auto-curricula in two-player zero-sum games. We follow the road-map set out by Fig. (1) which illustrates the flow of NAC. The relevant sections for each part of Fig. (1) are marked in pink.

3.1 The Meta-game

Consider two-player zero-sum games $\boldsymbol{G}$[2] drawn from some distribution $P(\boldsymbol{G})$, where players have the state-action space $\boldsymbol{S} \times \boldsymbol{A}$. We start by introducing the notion of an *agent* who is characterised by a policy ϕ, where a policy is a mapping $\phi : \boldsymbol{S} \times \boldsymbol{A} \to [0, 1]$ which can be described in both a tabular form or as a neural network. The payoff between two *agents* is defined to be $\mathfrak{M}(\phi_i, \phi_j)$ (i.e., the game engine), and represents the utility to agent i, or alternatively, the negative utility to agent j.

Our population-based framework revolves around iterative updates on the meta-game $\mathbf{M}$. At every iteration $t \in T$, a *Player* is defined by a population of fixed *agents* $\mathbf{\Phi}_t = \mathbf{\Phi}_0 \cup \{\phi_1^{\text{BR}}, \phi_2^{\text{BR}}, ..., \phi_t^{\text{BR}}\}$, where $\mathbf{\Phi}_0$ is the initial random agent pool and the ϕ_t^{BR} are discussed further in Sec. (3.3). From here,

[2] $\boldsymbol{G}$ encapsulates all of the information for a $Player$ to take part in a game (e.g., actions, reward functions).

for the sake of notation convenience, we will only consider the *single-population* case where *Players* share the same $\mathbf{\Phi}_t$. As such, the single population will generate a *meta-game* $\mathbf{M}_t$, a payoff matrix between all of the *agents* in the population, with individual entries being $\mathfrak{M}(\phi_i, \phi_j)\ \forall \phi_i, \phi_j \in \mathbf{\Phi}_t$.

3.2 The Meta-Solver

Based on $\mathbf{M}_t$, the *Player* will solve for the meta-distribution $\boldsymbol{\pi}_t$ which is defined as an aggregated *agent* over the fixed agents in $\mathbf{\Phi}_t$. These meta-solvers in recent work have stuck to human-designed solution concepts such as a uniform distribution [25] or NE and its variants [3, 37, 28], whereas we introduce a method to actively learn these distributions through solely interacting with the game. Specifically, we parameterise our meta-solver via a neural network. This network $f_{\boldsymbol{\theta}}$ with parameters $\boldsymbol{\theta}$ is a mapping $f_{\boldsymbol{\theta}} : \mathbf{M}_t \to [0, 1]^t$ which takes as input a meta-game $\mathbf{M}_t$ and outputs a *meta-distribution* $\boldsymbol{\pi}_t = f_{\boldsymbol{\theta}}(\mathbf{M}_t)$. The output $\boldsymbol{\pi}_t$ is a probability assignment to each *agent* in the population $\mathbf{\Phi}_t$ and, as we are in the *single*-population setting, we do not distinguish between different populations.

There are two characteristics that our network $f_{\boldsymbol{\theta}}$ should have: **firstly**, it should be able to handle *variable-length* inputs as the size of $\mathbf{M}_t$ is $(t \times t)$, and $f_{\boldsymbol{\theta}}$ outputs a vector of size $(t \times 1)$, where the value of t increments at every iteration. **Secondly**, $f_{\boldsymbol{\theta}}$ should have both *column-permutation invariance* and *row-permutation equivariance*. Given an input $\mathbf{M}_t$, $f_{\boldsymbol{\theta}}$ will output the distribution $\boldsymbol{\pi}_t = f_{\boldsymbol{\theta}}(\mathbf{M}_t)$ for the row *Player* and it should be equivariant to any permutation of the row index, and be invariant to any permutation of the column index as neither of these actions should affect $\boldsymbol{\pi}_t$.

Much work has been done on how to maintain permutation invariance/equivariance for neural networks such as Deepsets [63] for handling tasks defined on sets, or PointNet [39] for handling 3D point cloud data processing. Our first Multi-Layer Perceptron (MLP) based network is inspired by PointNet, which consists of three components: (1) An MLP over each element $\mathbf{M}_t^{ij}$ for a non-linear representation, (2) Column Mean-Pooling for column-permutation invariant information aggregation, which will be concatenated to each row and (3) Row-wise MLP for the row-permutation equivariant meta distribution. We also offer two alternatives, inspired by a fully Conv1d-based model [29] and a GRU-based model [7]. We refer to Appendix A for detailed architectures of our network $f_{\boldsymbol{\theta}}(\cdot)$.

3.3 Best Response Oracle

Once a meta-distribution $\boldsymbol{\pi}_t \in \Delta_{|\mathbf{\Phi_{t-1}}|}$ is obtained, the goal is to solve for a best-response against $\boldsymbol{\pi}_t$ to strengthen the population. Formally, we define

$$\mathfrak{M}(\phi, \langle \boldsymbol{\pi}_t, \mathbf{\Phi}_t \rangle) := \sum_{k=1}^{t} \boldsymbol{\pi}_t^k \mathfrak{M}(\phi, \phi_k), \tag{1}$$

to represent the payoff for agent ϕ against the aggregated agent (aggregated by $\boldsymbol{\pi}_t$) of population $\mathbf{\Phi}_t$. Consequently, we have that the best-response to an aggregated strategy is:

$$\phi_{t+1}^{\text{BR}} = \arg\max_{\phi} \sum_{k=1}^{t} \boldsymbol{\pi}_t^k \mathfrak{M}(\phi, \phi_k), \tag{2}$$

and the best-response is appended to the population to form a new fixed population, aiming to strengthen the population so as to become less exploitable. Depending on the sophistication of games, one may choose appropriate oracles. We consider different oracle implementations in Sec. (3.5).

3.4 The Learning Objective of Players

The goal of NAC is to find an auto-curricula that after T best-response iterations returns a meta-strategy and population, $\langle \boldsymbol{\pi}_T, \mathbf{\Phi}_T \rangle$, that helps minimise the exploitability, written as:

$$\min_{\boldsymbol{\theta}} \mathfrak{Expl}(\boldsymbol{\pi}_T(\boldsymbol{\theta}), \mathbf{\Phi}_T(\boldsymbol{\theta})), \text{ where } \mathfrak{Expl} := \max_{\phi} \mathfrak{M}\left(\phi, \langle \boldsymbol{\pi}_T, \mathbf{\Phi}_T \rangle\right), \tag{3}$$

$$\boldsymbol{\pi}_T = f_{\boldsymbol{\theta}}(\mathbf{M}_T), \mathbf{\Phi}_T = \left\{\phi_{\boldsymbol{T}}^{\text{BR}}(\boldsymbol{\theta}), \phi_{\boldsymbol{T-1}}^{\text{BR}}(\boldsymbol{\theta}), ..., \phi_{\boldsymbol{1}}^{\text{BR}}(\boldsymbol{\theta})\right\}. \tag{4}$$

$\mathfrak{Expl}(\cdot)$ [3] represents *exploitability* [9], a measure of the incentive to deviate from the meta-strategy over $\langle \boldsymbol{\pi}_T, \mathbf{\Phi}_T \rangle$. When exploitability reaches zero, it means one can no longer improve performance. $\phi_{\boldsymbol{t}}^{\text{BR}}(\boldsymbol{\theta})$ in Eq. (4) shows that each best-response has a dependency on $\boldsymbol{\theta}$ since $\phi_{\boldsymbol{t}}^{\text{BR}}$ is influenced

[3] In the *multi*-population case, $\mathfrak{Expl}$ expresses the notion of each *Player* having a different population and final meta-strategy, in the single-population case we only need to evaluate the deviation incentive for one population.

explicitly by the curriculum at iteration t and implicitly by all previous curricula. We believe such an objective maximises the generality of our framework on solving different types of zero-sum games.

NAC can use any oracle, however different considerations must be taken dependent on the choice. In the following sections we will discuss the technicality of directly solving for the meta-gradients of $\boldsymbol{\theta}$ with respect to a gradient-descent (GD) oracle and an RL-based oracle. Additionally, we will provide an oracle-agnostic method based on zero-order gradients that allows us to ignore oracle trajectories via Evolutionary Strategies [43]. Pseudo-code[4] for these methods is shown in Alg. (1).

Algorithm 1 Neural Auto-Curricula (NAC) - statements in teal refer only to ES-NAC, statements in red only to standard NAC.

Require: Game distribution $p(\boldsymbol{G})$, learning rate α, time window T, perturbations n, precision σ.

Randomly initialise policy pool $\boldsymbol{\phi_0}$, Initialise parameters $\boldsymbol{\theta}$ of the meta solver $f_{\boldsymbol{\theta}}$.
for each training iteration **do**
  Store current model $f_{\boldsymbol{\theta}}$.
  Sample $\boldsymbol{\epsilon}_1, ..., \boldsymbol{\epsilon}_n \sim \mathcal{N}(0, I)$ and store n models $f_{(\boldsymbol{\theta}+\boldsymbol{\epsilon}_i)}$. ▷ *If using ES, include this step*
  for each stored model f **do**
    Sample games $\{G_k\}_{k=1,...,K}$ from $p(\boldsymbol{G})$.
    for each game G_k **do**
      for each iteration t **do**
        Compute the meta-policy $\boldsymbol{\pi}_{t-1} = f(\mathbf{M}_{t-1})$.
        Compute the best response $\boldsymbol{\phi}_t^{\text{BR}}$ by Eq. (9) or Eq. (12).
        Expand the population $\boldsymbol{\Phi_t} = \boldsymbol{\Phi_{t-1}} \cup \{\boldsymbol{\phi}_t^{\text{BR}}\}$
      end for
      Compute $\mathfrak{Exp}_i(\boldsymbol{\pi}_T, \boldsymbol{\Phi}_T)$ by Eq. (3)
    end for
    Compute the meta-gradient $\mathbf{g}_k$ via Eq. (6) or Eq. (13)
    Update meta-solver's parameters $\boldsymbol{\theta}' = \boldsymbol{\theta} - \alpha \frac{1}{K} \sum_k \mathbf{g}_k$.
  end for
end for

3.5 Optimising the Meta-Solver through Meta-gradients

Based on the *Player's* learning objectives in Eq. (3), we can optimise the meta-solver as follows:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}), \text{ where } J(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{G}\sim P(\boldsymbol{G})}\left[\mathfrak{Exp}(\boldsymbol{\pi}, \boldsymbol{\Phi}|\boldsymbol{\theta}, \boldsymbol{G})\right]. \tag{5}$$

Deriving the (meta-)gradient of $\boldsymbol{\theta}$ is non-trivial, which we show in the below Remark (3.1).

Remark 3.1. *For a given distribution of game $p(\boldsymbol{G})$, by denoting exploitability at the final iteration $\mathfrak{M}\left(\boldsymbol{\phi}_{T+1}^{BR}, \langle \boldsymbol{\pi}_T, \boldsymbol{\Phi}_T \rangle\right)$ as $\mathfrak{M}_{T+1}$, the meta-gradient for $\boldsymbol{\theta}$ (see also Fig. 1) is*

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{G}}\left[\frac{\partial \mathfrak{M}_{T+1}}{\partial \boldsymbol{\phi}_{T+1}^{BR}}\frac{\partial \boldsymbol{\phi}_{T+1}^{BR}}{\partial \boldsymbol{\theta}} + \frac{\partial \mathfrak{M}_{T+1}}{\partial \boldsymbol{\pi}_T}\frac{\partial \boldsymbol{\pi}_T}{\partial \boldsymbol{\theta}} + \frac{\partial \mathfrak{M}_{T+1}}{\partial \boldsymbol{\Phi}_T}\frac{\partial \boldsymbol{\Phi}_T}{\partial \boldsymbol{\theta}}\right], \textit{where} \tag{6}$$

$$\frac{\partial \boldsymbol{\pi}_T}{\partial \boldsymbol{\theta}} = \frac{\partial f_{\boldsymbol{\theta}}(\mathbf{M}_T)}{\partial \boldsymbol{\theta}} + \frac{\partial f_{\boldsymbol{\theta}}(\mathbf{M}_T)}{\partial \mathbf{M}_T}\frac{\partial \mathbf{M}_T}{\partial \boldsymbol{\Phi}_T}\frac{\partial \boldsymbol{\Phi_T}}{\partial \boldsymbol{\theta}}, \quad \frac{\partial \boldsymbol{\phi}_{T+1}^{BR}}{\partial \boldsymbol{\theta}} = \frac{\partial \boldsymbol{\phi}_{T+1}^{BR}}{\partial \boldsymbol{\pi}_T}\frac{\partial \boldsymbol{\pi}_T}{\partial \boldsymbol{\theta}} + \frac{\partial \boldsymbol{\phi}_{T+1}^{BR}}{\partial \boldsymbol{\Phi}_T}\frac{\partial \boldsymbol{\Phi}_T}{\partial \boldsymbol{\theta}}, \tag{7}$$

$$\frac{\partial \boldsymbol{\Phi_T}}{\partial \boldsymbol{\theta}} = \left\{\frac{\partial \boldsymbol{\Phi_{T-1}}}{\partial \boldsymbol{\theta}}, \frac{\partial \boldsymbol{\phi}_T^{BR}}{\partial \boldsymbol{\theta}}\right\}, \tag{8}$$

and Eq. (8) can be further decomposed by iteratively applying Eq. (7) from iteration $T-1$ to 0.

The full proof is in Appendix B. Intuitively, in the forward process, the population adds T new agents. In the backward process, the meta-gradient traverses through the full T best-response iterations (each iteration may involve many gradient updates) and back-propagates through all trajectories. Therefore, the gradients of $\partial \boldsymbol{\Phi_T}/\partial \boldsymbol{\theta}$ need collecting from $\mathbf{M}_{T-1}$ to $\mathbf{M}_1$. Whilst this is critical in ensuring that every *agent* is influential in optimising $\boldsymbol{\theta}$, it introduces computational troubles. Firstly, due to the long-trajectory dependency, computing meta-gradients becomes inefficient due to multiple Hessian-vector products. Secondly, the gradients are susceptible to exploding/vanishing gradients in the same manner as RNNs [19]. To alleviate these issues, we introduce a truncated version similar to

[4] Pseudo-code including details of the best-response oracles is shown in Appendix C

[38], where we back-propagate up to a smaller **window size** (i.e., $n < T$) of population updates. We shall study the effect of the window size later in Figure 4. Notably, the gradients of $\partial \boldsymbol{\phi}_{t+1}^{\text{BR}} / \partial \boldsymbol{\Phi}_t$ and $\partial \boldsymbol{\phi}_{t+1}^{\text{BR}} / \partial \boldsymbol{\pi}_t$ in Eq. (7) also depends on the type of best-response subroutines. In the next section, we demonstrate two types of oracles and show how the meta-gradients are derived accordingly.

3.5.1 Gradient-Descent Best-Response Oracles

When the payoff function $\boldsymbol{G}$ is known and differentiable, one can approximate the best-response through gradient descent (GD). A one-step GD oracle example is written as:

$$\boldsymbol{\phi}_{t+1}^{\text{BR}} = \boldsymbol{\phi}_0 + \alpha \frac{\partial \mathfrak{M}\left(\boldsymbol{\phi}_0, \langle \boldsymbol{\pi}_t, \boldsymbol{\Phi}_t \rangle\right)}{\partial \boldsymbol{\phi}_0}, \tag{9}$$

where $\boldsymbol{\phi}_0$ and α denote the initial parameters and learning rate respectively. The backward gradients of one-step GD share similarities with MAML [13], which can be written as:

$$\frac{\partial \boldsymbol{\phi}_{t+1}^{\text{BR}}}{\partial \boldsymbol{\pi}_t} = \alpha \frac{\partial^2 \mathfrak{M}\left(\boldsymbol{\phi}_0, \langle \boldsymbol{\pi}_t, \boldsymbol{\Phi}_t \rangle\right)}{\partial \boldsymbol{\phi}_0 \partial \boldsymbol{\pi}_t}, \quad \frac{\partial \boldsymbol{\phi}_{t+1}^{\text{BR}}}{\partial \boldsymbol{\Phi}_t} = \alpha \frac{\partial^2 \mathfrak{M}\left(\boldsymbol{\phi}_0, \langle \boldsymbol{\pi}_t, \boldsymbol{\Phi}_t \rangle\right)}{\partial \boldsymbol{\phi}_0 \partial \boldsymbol{\Phi}_t}. \tag{10}$$

We refer to Appendix B.1 for the specification of Remark (3.1) for gradient descent-based oracles.

Though Eq. (9) and Eq. (10) can be easily extended to multi-step GD case, it becomes easily intractable to take the gradient of a computational graph that includes hundreds of gradient updates [48, 40, 30]. To solve this problem, we offer another solution for efficient back-propagation based on the *implicit gradient* method [40], which does not need the full trajectory as a dependency. The idea is that, when we arrive at the best response point such that $\partial \mathfrak{M}(\boldsymbol{\phi}_{t+1}^{\text{BR}}, \langle \pi_t, \boldsymbol{\Phi}_t \rangle) / \partial \boldsymbol{\phi}_t^{\text{BR}} = 0$, we can apply the implicit function theorem and derive the gradient by,

$$\frac{\partial \boldsymbol{\phi}_{t+1}^{\text{BR}}}{\partial \boldsymbol{\Phi}_t} = -\left[\frac{\partial^2 \mathfrak{M}(\boldsymbol{\phi}_{t+1}^{\text{BR}}, \langle \boldsymbol{\pi}_t, \boldsymbol{\Phi}_{\boldsymbol{t}} \rangle)}{\partial \boldsymbol{\phi}_{t+1}^{\text{BR}} \partial {\boldsymbol{\phi}_{t+1}^{\text{BR}}}^T}\right]^{-1} \frac{\partial^2 \mathfrak{M}(\boldsymbol{\phi}_{t+1}^{\text{BR}}, \langle \boldsymbol{\pi}_t, \boldsymbol{\Phi}_{\boldsymbol{t}} \rangle)}{\partial \boldsymbol{\phi}_{t+1}^{\text{BR}} \partial \boldsymbol{\Phi}_{\boldsymbol{t}}}. \tag{11}$$

See the proof in Appendix B.2. Eq. (11) allows for efficient back-propagation by ignoring the dependency on trajectories. Note that the implicit gradient methods can theoretically be applied to compute the meta-gradient *w.r.t* RL oracles (next section), but empirically we had little success.

3.5.2 Reinforcement Learning Best-Response Oracles

The above GD-based oracles require the pay-off function (i.e., the game engine) to be differentiable. Yet, for complex real-world games such as StarCraft [53], we have to rely on RL methods to approximate the best-response agent. Overall, the RL meta-gradient shares a similar structure to that of the above. The major difference is that we replace the GD terms with Policy Gradient estimator [55]. Considering the unbiased estimators for the first and the second-order meta-(policy-)gradients, we apply Differentiable Monte Carlo Estimator (DICE) [14] in Eq. (9). DICE is an unbiased higher-order gradient estimator that is fully compatible with automatic differentiation. Thanks to DICE, for an RL-based oracle, by regarding the best-response agent $\boldsymbol{\phi}_t^{\text{BR}}$ as $\boldsymbol{\phi_1}$ and the aggregated agent $\langle \boldsymbol{\pi}_t, \boldsymbol{\Phi}_t \rangle$ as $\boldsymbol{\phi_2}$ respectively, we obtain the follow equation:

$$\boldsymbol{\phi}_1 = \boldsymbol{\phi}_0 + \alpha \frac{\partial \mathcal{J}^{\text{DICE}}}{\partial \boldsymbol{\phi}_0}, \text{ where } \mathcal{J}^{\text{DICE}} = \sum_{k=0}^{H-1} \left(\prod_{k'=0}^{k} \frac{\boldsymbol{\pi}_{\boldsymbol{\phi}_1}\left(a_{k'}^1 \mid s_{k'}^1\right) \boldsymbol{\pi}_{\boldsymbol{\phi}_2}\left(a_{k'}^2 \mid s_{k'}^2\right)}{\perp\left(\boldsymbol{\pi}_{\boldsymbol{\phi}_1}\left(a_{k'}^1 \mid s_{k'}^1\right) \boldsymbol{\pi}_{\boldsymbol{\phi}_2}\left(a_{k'}^2 \mid s_{k'}^2\right)\right)} \right) r_k^1, \tag{12}$$

where $\perp$ refers to the stop-gradient operator, r_k^1 for the reward for agent 1, and H represents the trajectory length. We refer to Appendix B.3 for how DICE provides the unbiased first and second-order meta gradient, and the specification of Remark (3.1) for RL-based oracles. This RL-based formulation is limited in the fact that it is does not directly extend to SOTA RL techniques such as value-based methods [18], and therefore we next introduce a zero-order method that is able to tackle the case of a non-differentiable pay-off function with any best-response oracle.

3.6 Optimising the Meta-Solver through Evolution Strategies

Inspired by the generality of Evolutionary Strategies (ES) [43] in optimising black-box functions and ES-based MAML [48], we also propose an ES based framework that can cope with any best-response oracle and underlying game engine. We name our method ES-NAC.

The ES approach of [43] states that if we have an objective function $F(\boldsymbol{\theta})$ over a network parameterised by $\boldsymbol{\theta}$, we can apply Gaussian perturbations to the parameters so that a gradient estimate of the

objective function can be achieved by $\nabla_{\boldsymbol{\theta}}\mathbb{E}_{\boldsymbol{\epsilon}\sim\mathcal{N}(0,I)}F(\boldsymbol{\theta}+\sigma\boldsymbol{\epsilon}) = \frac{1}{\sigma}\mathbb{E}_{\boldsymbol{\epsilon}\sim\mathcal{N}(0,I)}F(\boldsymbol{\theta}+\sigma\boldsymbol{\epsilon})\boldsymbol{\epsilon}$. In our framework, if we set the objective function $F(\boldsymbol{\theta}) = \mathfrak{Exploitability}_T(\boldsymbol{\pi}_T, \boldsymbol{\Phi}_T)$ and $\boldsymbol{\pi}_T = f_{\boldsymbol{\theta}}(\mathbf{M}_T)$, we can have an objective function as a function of $\boldsymbol{\theta}$ which can be perturbed. This allows us to write the gradient of a surrogate objective of Eq. (5) as follows:

$$\nabla_{\boldsymbol{\theta}}\hat{J}_\sigma(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{G}\sim P(\boldsymbol{G}),\boldsymbol{\epsilon}\sim\mathcal{N}(0,I)}\left[\frac{1}{\sigma}\big(\mathfrak{Exploitability}_T(\boldsymbol{\pi}_T, \boldsymbol{\Phi}_T)\Big|\boldsymbol{\theta}+\boldsymbol{\epsilon}, \boldsymbol{G}\big)\boldsymbol{\epsilon}\right]. \tag{13}$$

Additionally, we make use of control variates [27] to reduce the variance of the estimator whilst remaining unbiased, for example we apply forward finite differences [6] whereby the exploitability of the unperturbed meta-solver $f_{\boldsymbol{\theta}}$ is subtracted from the perturbed meta-solver, that is

$$\nabla_{\boldsymbol{\theta}}\mathbb{E}_{\boldsymbol{\epsilon}\sim\mathcal{N}(0,I)}F\big(\boldsymbol{\theta}+\sigma\boldsymbol{\epsilon}\big) = \frac{1}{\sigma}\mathbb{E}_{\boldsymbol{\epsilon}\sim\mathcal{N}(0,I)}\big(F(\boldsymbol{\theta}+\sigma\boldsymbol{\epsilon}) - F(\boldsymbol{\theta})\big)\boldsymbol{\epsilon}. \tag{14}$$

The key benefit of ES-NAC is that it is agnostic to the best-response oracle choice, as only the final exploitability is required. Unlike the implicit formulation we provided in Sec. (3.5.1), it is not restricted by the fixed-point condition, which we note is difficult to attain for RL oracles, and therefore may be more widely applicable. This is particularly useful in practice since most games require hundreds of game simulations for each entry in $\mathbf{M}$ (e.g., StarCraft [53]), in which case we lose the applicability of either GD or RL-based oracles. We note that Alg. (1) encapsulates ES-NAC when the number of perturbations $n > 0$, and that lines in teal refer only to the ES formulation.

4 Experiments

We validate the effectiveness of NAC on five types of zero-sum environments[5] with different levels of complexity. They are Games of Skill (GoS) [8], differentiable Lotto [3], non-transitive mixture game (2D-RPS) [37], iterated matching pennies (IMP) [15, 21] and Kuhn Poker [23]. All selected games are non-trivial to solve as an effective solver has to consider both *transitive* and *non-transitive* dynamics in the policy space [8, 44]. The motivation behind our selections is to evaluate the performance of NAC under the different oracles proposed in Sec. (3.5) and Sec. (3.6). Specifically, we test the gradient descent-oracle (GD) in GoS, Lotto and 2D-RPS and the RL oracle in IMP. For ES-NAC, we conduct experiments on Kuhn poker [23] with two approximate tabular oracles (V1, V2), an exact tabular oracle[6] and a PPO [45] oracle. We conduct the experiments on multiple random seeds for NAC and the details of how we conduct meta-testing on baseline algorithms and NAC are reported in Appendix E.3. More details of all of the applied oracles and their hyper-parameters are in Appendix F, and details of the baseline implementations are in Appendix E.

We select the baselines to be vanilla self-play (i.e., best responding to the latest agent in the population) and the PSRO variants, including PSRO [25], PSRO-Uniform [3] (equivalent to Fictitious Play [5]) and PSRO-rN [3]. Their implementations can be found in OpenSpiel [24]. We believe these methods offer strong benchmarks for NAC since they are all underpinned by game theoretic principles, and NAC tries to discover solution algorithms purely from data. Results are presented in the form of answering five critical questions *w.r.t* the effectiveness of NAC.

Question 1. ***How does NAC perform in terms of exploitability on different games?***

Firstly, we are interested in whether NAC can learn an auto-curricula that can solve games effectively. In order to characterise performance, we measure the exploitability, Eq. (3), of NAC and compare it against other baselines. Surprisingly, results in Fig. (2) suggest that, by learning an effective meta-solver, NAC is able to solve games without explicit game theoretic solution concepts. In particular, NAC performs at least as well as PSRO (only slightly worse than PSRO in IMP), and in multiple games outperforms PSRO. We notice that NAC performs better in the presence of *approximate* best-responses. One explanation is that Double Oracle relies upon a *true* best-response oracle to guarantee convergence, when it comes to PSRO where only *approximate* best responses are available, the principles of sub-game NE may not necessarily fit with PSRO anymore. In contrast, NAC considers the outcomes of approximate best-responses in an end-to-end fashion; therefore, the auto-curricula for each player tends to be adaptive, leading to the development of a stronger population. Overall, we believe these results suggest a promising avenue of research for solving larger-scale games (e.g., StarCraft [53] or XLand [50]) with no exact best responses available and no prior game theoretic solution concepts involved.

[5] To stay self-contained, we provide a detailed description for each game in Appendix E.1.

[6] Training performance for the exact tabular oracle provided in Appendix D.1

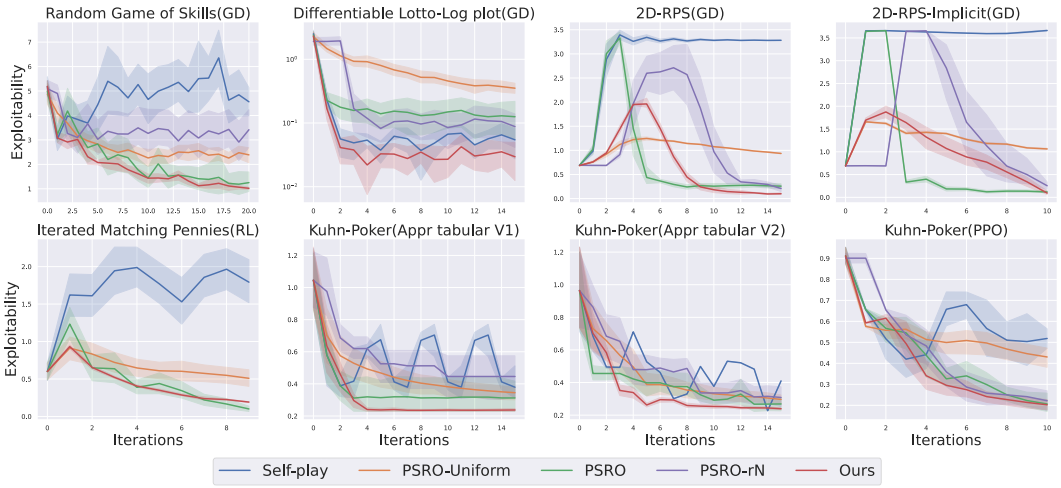

Figure 2: Exploitability results on five different environments with differing best-response oracles. NAC performs at least as well as the best baseline in all settings, and often outperforms the PSRO baselines. Settings of adopted oracles in each game can be found in Appendix F.

Question 2. ***What does the learned curricula (i.e., $f_\theta(\mathbf{M}_t)$) look like?***

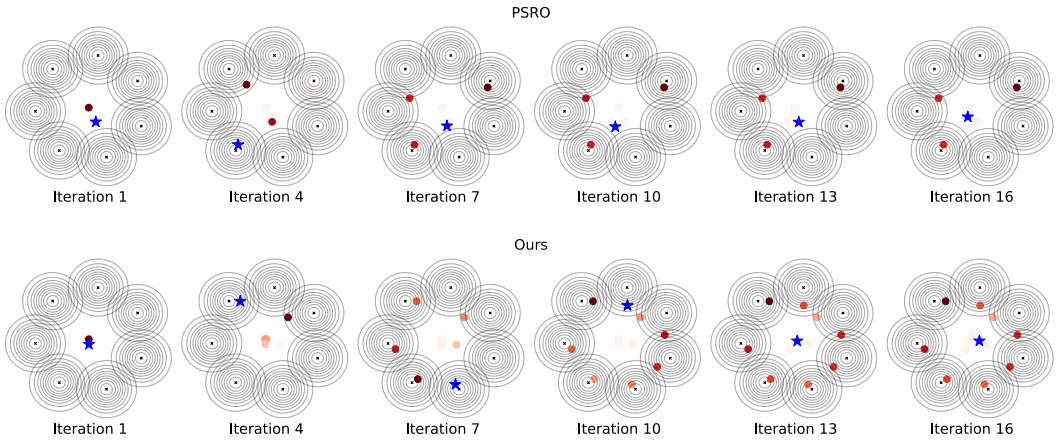

Figure 3: Visualisations of curricula on 2D-RPS. Red points denote the meta-solver output, and darker refers to higher probability in $\boldsymbol{\pi}$. The blue star denotes the latest best-response. To achieve low exploitability, one needs to climb up and explore each Gaussian. PSRO fails to explore fully, where as NAC creates an effective curricula to explore all modes and assigns each of them high weights.

To address this question[7], we visualise the auto-curricula on the 2D-RPS game between PSRO and NAC in Fig. (3). PSRO is successful in climbing up some Gaussians before iteration 7; however, it fails to offer an effective auto-curricula that can lead it to discover other Gaussians. PSRO fails to select an auto-curricula that takes into consideration whether the best-response oracle is capable of learning a *suitable* best-response. This result is inline with [37] and we believe it is because the approximate best responses may lead to a local optimum in the policy space for PSRO-type methods. In contrast, NAC adaptively generates an auto-curricula which is more *suitable* for approximate best-responses, as evidenced by a wide spread of agents over the plane, and lower exploitability.

Question 3. ***Does back-propogation through multiple best-response iterations help the training?***

As shown by the blue lines in Fig. (1), the backward meta-gradient will propagate through multiple iterations of best-response processes. To demonstrate its effects, in Fig. (4c) we conduct an ablation

[7]We also visualise Kuhn-Poker policy for understanding how NAC works in Appendix D.2

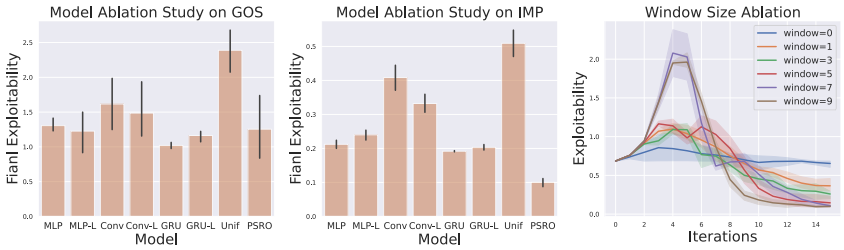

Figure 4: (a), (b) The effect of different neural architectures on the exploitability in GoS and IMP games. (c) The effect of window size on the exploitability in 2D-RPS game.

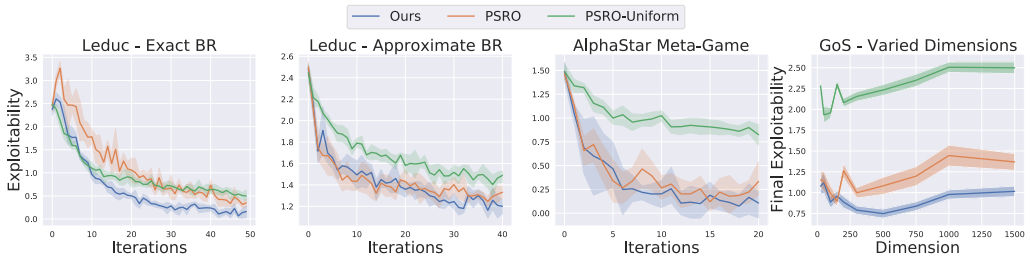

Figure 5: (a) Exploitability when trained on Kuhn Poker with an exact tabular BR oracle using ES-NAC and tested on Leduc Poker. (b) Same as (a) with approximate tabular BR V2 (c) Exploitability when trained on GoS with a GD oracle and tested on the AlphaStar meta-game from [8] (d) Final exploitability when trained on 200 Dimension GoS and tested on a variety of dimension size GoS.

study on NAC by varying the how many best-response iterations (i.e., the window size) we consider, by controlling how many agents are added into the population before computing the meta-gradient. A window size of 0 refers to the setting where we completely detach the gradients of the best-response process. We can see that NAC achieves lower exploitability when considering multiple best-response iterations, which reflects the effectiveness of NAC in offering a more *suitable* and appropriate curricula to strengthen the whole population.

Question 4. ***How is NAC affected by the architecture and capacity of meta-solver?***

In Sec. (3.2), we provide several neural architectures for the meta-solver. Thus, to understand the effect of these different architectures, we conduct an ablation study on GoS and IMP. We specify six different models by varying both the architecture and the size. Results in Fig. (4a, 4b) show that, firstly, the permutation invaraiance/equivariance is not a necessary property, as the GRU-based models achieve great performance. Secondly, the effect of the meta-solver's architecture is heavily dependent on the game. The performance of all three models are comparable for GOS while only MLP and GRU work well for IMP. Different games may need different meta-solver's architecture and GRU-based meta-solver tend to work better. In addition, the increase of network capacity does not correspond to performance improvement. We refer the reader to Appendix (F) for details of our model choices for different games.

Question 5. ***What is the generalisation ability of the neural meta-solver by NAC?***

The most promising aspect of NAC is that the neural auto-curricula (i.e., meta-solvers) have the ability to generalise to different out-of-distribution games. This is particularly impactful, as it allows for training on simpler games and then deploying them on larger, more difficult games. We test the generalisation capability of NAC in two settings. First, we take our meta-solver trained over 200 dimensional GoS and test on new unseen GoS of varying dimension. We consider this to be the most direct way of ascertaining whether the neural meta-solvers are able to generalise to larger, more difficult games, and whether the in-task performance still holds out-of-task. Fig. (5d) plots the final exploitability after 20 PSRO iterations against the dimension of the GoS, and noticeably, NAC still outperforms the PSRO baselines in all dimensions larger than the training dimension. Additionally,

we test our trained meta-solver on the AlphaStar meta-game generated by [8][8] in Fig. (5c), which is also considered to be a form of a GoS. Interestingly, our meta-solver is able to perform well on a GoS that is outside of the task distribution and therefore has a different type of underlying dynamics.

Secondly, we introduce an example of our meta-solver showing the ability to scale-up to different games, namely we train on the Kuhn Poker environment (2^{12} pure strategies) and test on the Leduc Poker environment (3^{936} pure strategies). As shown in Fig. (5a, 5b) the trained meta-solver is able to outperform the PSRO algorithms when used on Leduc Poker, which suggests NAC enjoys effective generalisation abilities for both an exact best-response oracle and an approximate best-response oracle. We hypothesise that, whilst Leduc Poker is different from Kuhn Poker, the "Poker" nature of both games means they encapsulate similar dynamics, allowing our meta-solver to perform favourably.

5 Conclusion

We introduce a method for discovering auto-curricula on two-player zero-sum games based on meta-learning. To our best knowledge, we are the first to show that it is entirely possible to perform as well as solutions underpinned in game-theoretic concepts that are designed through human insights, without any active design of the auto-curricula itself. In particular, we show that our NAC method can learn in small games and generalise to larger games, more difficult games that follow a similar underlying structure. We believe this initiates an exciting and promising research area in which large-scale difficult games can be solved effectively by training on simplified versions of the game.

Author Contributions

We summarise the main contributions from each of the authors as follows:

Xidong Feng: Idea proposing, algorithm design, code implementation and experiments running (on 2D-RPS, 2D-RPS-Implicit and IMP), and paper writing.

Oliver Slumbers: Algorithm design, code implementation and experiments running (on Gos, Blotto, Kuhn-Poker), and paper writing.

Ziyu Wan: Code implementation and experiments running for RL based NAC in Appendix D.4.

Bo Liu: Experiments running for Kuhn-Poker.

Stephen McAleer: Project discussion and paper writing.

Ying Wen: Project discussion.

Jun Wang: Project discussion and overall project supervision.

Yaodong Yang: Project lead, idea proposing, experiment supervision, and whole manuscript writing. The work was done at King's College London.

References

[1] Maruan Al-Shedivat, Trapit Bansal, Yuri Burda, Ilya Sutskever, Igor Mordatch, and Pieter Abbeel. Continuous adaptation via meta-learning in nonstationary and competitive environments. *arXiv preprint arXiv:1710.03641*, 2017.

[2] Bowen Baker, Ingmar Kanitscheider, Todor Markov, Yi Wu, Glenn Powell, Bob McGrew, and Igor Mordatch. Emergent tool use from multi-agent autocurricula. *arXiv preprint arXiv:1909.07528*, 2019.

[3] D Balduzzi, M Garnelo, Y Bachrach, W Czarnecki, J Pérolat, M Jaderberg, and T Graepel. Open-ended learning in symmetric zero-sum games. In *ICML*, volume 97, pages 434–443. PMLR, 2019.

[4] Sarah Bechtle, Artem Molchanov, Yevgen Chebotar, Edward Grefenstette, Ludovic Righetti, Gaurav Sukhatme, and Franziska Meier. Meta learning via learned loss. In *2020 25th International Conference on Pattern Recognition (ICPR)*, pages 4161–4168. IEEE, 2021.

[8] We provide the results on the other tens of meta-games from [8] in Appendix D.3

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

[6] Krzysztof Choromanski, Mark Rowland, Vikas Sindhwani, Richard E. Turner, and Adrian Weller. Structured evolution with compact architectures for scalable policy optimization, 2018.

[7] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[8] Wojciech Marian Czarnecki, Gauthier Gidel, Brendan Tracey, Karl Tuyls, Shayegan Omidshafiei, David Balduzzi, and Max Jaderberg. Real world games look like spinning tops. *arXiv preprint arXiv:2004.09468*, 2020.

[9] Trevor Davis, Neil Burch, and Michael Bowling. Using response functions to measure strategy strength. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 28, 2014.

[10] Xiaotie Deng, Yuhao Li, David Henry Mguni, Jun Wang, and Yaodong Yang. On the complexity of computing markov perfect equilibrium in general-sum stochastic games. *arXiv preprint arXiv:2109.01795*, 2021.

[11] Le Cong Dinh, Yaodong Yang, Zheng Tian, Nicolas Perez Nieves, Oliver Slumbers, David Henry Mguni, and Jun Wang. Online double oracle. *arXiv preprint arXiv:2103.07780*, 2021.

[12] Yan Duan, John Schulman, Xi Chen, Peter L Bartlett, Ilya Sutskever, and Pieter Abbeel. Rl: Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779*, 2016.

[13] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*, pages 1126–1135. PMLR, 2017.

[14] Jakob Foerster, Gregory Farquhar, Maruan Al-Shedivat, Tim Rocktäschel, Eric Xing, and Shimon Whiteson. Dice: The infinitely differentiable monte carlo estimator. In *International Conference on Machine Learning*, pages 1529–1538. PMLR, 2018.

[15] Jakob N Foerster, Richard Y Chen, Maruan Al-Shedivat, Shimon Whiteson, Pieter Abbeel, and Igor Mordatch. Learning with opponent-learning awareness. *arXiv preprint arXiv:1709.04326*, 2017.

[16] Robert S Gibbons. *Game theory for applied economists*. Princeton University Press, 1992.

[17] Sergiu Hart. Discrete colonel blotto and general lotto games. *International Journal of Game Theory*, 36(3):441–460, 2008.

[18] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[19] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[20] Rein Houthooft, Richard Y Chen, Phillip Isola, Bradly C Stadie, Filip Wolski, Jonathan Ho, and Pieter Abbeel. Evolved policy gradients. *arXiv preprint arXiv:1802.04821*, 2018.

[21] Dong-Ki Kim, Miao Liu, Matthew Riemer, Chuangchuang Sun, Marwa Abdulhai, Golnaz Habibi, Sebastian Lopez-Cot, Gerald Tesauro, and Jonathan P How. A policy gradient algorithm for learning to learn in multiagent reinforcement learning. *arXiv preprint arXiv:2011.00382*, 2020.

[22] Louis Kirsch, Sjoerd van Steenkiste, and Juergen Schmidhuber. Improving generalization in meta reinforcement learning using learned objectives. In *International Conference on Learning Representations*, 2019.

[23] Harold W Kuhn. 9. a simplified two-person poker. In *Contributions to the Theory of Games (AM-24), Volume I*, pages 97–104. Princeton University Press, 2016.

[24] Marc Lanctot, Edward Lockhart, Jean-Baptiste Lespiau, Vinicius Zambaldi, Satyaki Upadhyay, Julien Pérolat, Sriram Srinivasan, Finbarr Timbers, Karl Tuyls, Shayegan Omidshafiei, et al. Openspiel: A framework for reinforcement learning in games. *arXiv preprint arXiv:1908.09453*, 2019.

[25] Marc Lanctot, Vinicius Zambaldi, Audrunas Gruslys, Angeliki Lazaridou, Karl Tuyls, Julien Perolat, David Silver, and Thore Graepel. A unified game-theoretic approach to multiagent reinforcement learning, 2017.

[26] Joel Z Leibo, Edward Hughes, Marc Lanctot, and Thore Graepel. Autocurricula and the emergence of innovation from social interaction: A manifesto for multi-agent intelligence research. *arXiv e-prints*, pages arXiv–1903, 2019.

[27] Hao Liu, Richard Socher, and Caiming Xiong. Taming maml: Efficient unbiased meta-reinforcement learning. In *International Conference on Machine Learning*, pages 4061–4071. PMLR, 2019.

[28] Xiangyu Liu, Hangtian Jia, Ying Wen, Yaodong Yang, Yujing Hu, Yingfeng Chen, Changjie Fan, and Zhipeng Hu. Unifying behavioral and response diversity for open-ended learning in zero-sum games. *arXiv preprint arXiv:2106.04958*, 2021.

[29] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation, 2015.

[30] Jonathan Lorraine, Paul Vicol, and David Duvenaud. Optimizing millions of hyperparameters by implicit differentiation. In *International Conference on Artificial Intelligence and Statistics*, pages 1540–1552. PMLR, 2020.

[31] Stephen McAleer, John Lanier, Pierre Baldi, and Roy Fox. XDO: A double oracle algorithm for extensive-form games. *Reinforcement Learning in Games Workshop, AAAI*, 2021.

[32] Stephen McAleer, John Lanier, Roy Fox, and Pierre Baldi. Pipeline PSRO: A scalable approach for finding approximate nash equilibria in large games. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[33] H Brendan McMahan, Geoffrey J Gordon, and Avrim Blum. Planning in the presence of cost functions controlled by an adversary. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pages 536–543, 2003.

[34] Oskar Morgenstern and John Von Neumann. *Theory of games and economic behavior*. Princeton university press, 1953.

[35] Paul Muller, Shayegan Omidshafiei, Mark Rowland, Karl Tuyls, Julien Perolat, Siqi Liu, Daniel Hennes, Luke Marris, Marc Lanctot, Edward Hughes, et al. A generalized training approach for multiagent learning. In *International Conference on Learning Representations*, 2019.

[36] John F Nash et al. Equilibrium points in n-person games. *Proceedings of the national academy of sciences*, 36(1):48–49, 1950.

[37] Nicolas Perez Nieves, Yaodong Yang, Oliver Slumbers, David Henry Mguni, and Jun Wang. Modelling behavioural diversity for learning in open-ended games. *arXiv preprint arXiv:2103.07927*, 2021.

[38] Junhyuk Oh, Matteo Hessel, Wojciech M Czarnecki, Zhongwen Xu, Hado van Hasselt, Satinder Singh, and David Silver. Discovering reinforcement learning algorithms. *arXiv preprint arXiv:2007.08794*, 2020.

[39] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660, 2017.

[40] Aravind Rajeswaran, Chelsea Finn, Sham Kakade, and Sergey Levine. Meta-learning with implicit gradients, 2019.

[41] Kate Rakelly, Aurick Zhou, Chelsea Finn, Sergey Levine, and Deirdre Quillen. Efficient off-policy meta-reinforcement learning via probabilistic context variables. In *International conference on machine learning*, pages 5331–5340. PMLR, 2019.

[42] Jonas Rothfuss, Dennis Lee, Ignasi Clavera, Tamim Asfour, and Pieter Abbeel. Promp: Proximal meta-policy search. *arXiv preprint arXiv:1810.06784*, 2018.

[43] Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. Evolution strategies as a scalable alternative to reinforcement learning, 2017.

[44] Ricky Sanjaya, Jun Wang, and Yaodong Yang. Measuring the non-transitivity in chess, 2021.

[45] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[46] Jianzhun Shao, Hongchang Zhang, Yuhang Jiang, Shuncheng He, and Xiangyang Ji. Credit assignment with meta-policy gradient for multi-agent reinforcement learning. *arXiv preprint arXiv:2102.12957*, 2021.

[47] Avi Singh, Huihan Liu, Gaoyue Zhou, Albert Yu, Nicholas Rhinehart, and Sergey Levine. Parrot: Data-driven behavioral priors for reinforcement learning. *arXiv preprint arXiv:2011.10024*, 2020.

[48] Xingyou Song, Wenbo Gao, Yuxiang Yang, Krzysztof Choromanski, Aldo Pacchiano, and Yunhao Tang. Es-maml: Simple hessian-free meta learning. *arXiv preprint arXiv:1910.01215*, 2019.

[49] Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z Leibo, Karl Tuyls, et al. Value-decomposition networks for cooperative multi-agent learning. *arXiv preprint arXiv:1706.05296*, 2017.

[50] Ended Learning Team, Adam Stooke, Anuj Mahajan, Catarina Barros, Charlie Deck, Jakob Bauer, Jakub Sygnowski, Maja Trebacz, Max Jaderberg, Michael Mathieu, et al. Open-ended learning leads to generally capable agents. *arXiv preprint arXiv:2107.12808*, 2021.

[51] Jan van den Brand. A deterministic linear program solver in current matrix multiplication time. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 259–278. SIAM, 2020.

[52] Vivek Veeriah, Matteo Hessel, Zhongwen Xu, Janarthanan Rajendran, Richard L Lewis, Junhyuk Oh, Hado van Hasselt, David Silver, and Satinder Singh. Discovery of useful questions as auxiliary tasks. In *NeurIPS*, 2019.

[53] Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

[54] Jane X Wang, Zeb Kurth-Nelson, Dhruva Tirumala, Hubert Soyer, Joel Z Leibo, Remi Munos, Charles Blundell, Dharshan Kumaran, and Matt Botvinick. Learning to reinforcement learn. *arXiv preprint arXiv:1611.05763*, 2016.

[55] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

[56] Zhongwen Xu, Hado van Hasselt, Matteo Hessel, Junhyuk Oh, Satinder Singh, and David Silver. Meta-gradient reinforcement learning with an objective discovered online. *arXiv preprint arXiv:2007.08433*, 2020.

[57] Zhongwen Xu, Hado van Hasselt, and David Silver. Meta-gradient reinforcement learning. *arXiv preprint arXiv:1805.09801*, 2018.

[58] Yaodong Yang, Jun Luo, Ying Wen, Oliver Slumbers, Daniel Graves, Haitham Bou Ammar, Jun Wang, and Matthew E Taylor. Diverse auto-curriculum is critical for successful real-world multiagent learning systems. In *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*, pages 51–56, 2021.

[59] Yaodong Yang, Rasul Tutunov, Phu Sakulwongtana, and Haitham Bou Ammar. $\alpha\alpha$-rank: Practically scaling α-rank through stochastic optimisation. In *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, pages 1575–1583, 2020.

[60] Yaodong Yang and Jun Wang. An overview of multi-agent reinforcement learning from game theoretical perspective. *arXiv preprint arXiv:2011.00583*, 2020.

[61] Yaodong Yang, Ying Wen, Jun Wang, Liheng Chen, Kun Shao, David Mguni, and Weinan Zhang. Multi-agent determinantal q-learning. In *International Conference on Machine Learning*, pages 10757–10766. PMLR, 2020.

[62] Tom Zahavy, Zhongwen Xu, Vivek Veeriah, Matteo Hessel, Junhyuk Oh, Hado van Hasselt, David Silver, and Satinder Singh. A self-tuning actor-critic algorithm. *arXiv preprint arXiv:2002.12928*, 2020.

[63] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Ruslan Salakhutdinov, and Alexander Smola. Deep sets. *arXiv preprint arXiv:1703.06114*, 2017.

[64] Zeyu Zheng, Junhyuk Oh, Matteo Hessel, Zhongwen Xu, Manuel Kroiss, Hado Van Hasselt, David Silver, and Satinder Singh. What can learned intrinsic rewards capture? In *International Conference on Machine Learning*, pages 11436–11446. PMLR, 2020.

[65] Zeyu Zheng, Junhyuk Oh, and Satinder Singh. On learning intrinsic rewards for policy gradient methods. *Advances in Neural Information Processing Systems*, 31:4644–4654, 2018.

[66] Wei Zhou, Yiying Li, Yongxin Yang, Huaimin Wang, and Timothy Hospedales. Online meta-critic learning for off-policy actor-critic methods. *Advances in Neural Information Processing Systems*, 33, 2020.