Learning not to Regret

David Sychrovsky
Charles University
Prague, Czechia
sychrovsky@kam.mff.cuni.cz

Michal Sustr
Czech Technical University
Prague, Czechia
michal.sustr@aic.fel.cvut.cz

Elnaz Davoodi
DeepMind
Canada
elnazd@google.com

Marc Lanctot
DeepMind
Canada
lanctot@google.com

Martin Schmid
Equilibre Technologies
Prague, Czechia
schmid@equilibretechnologies.com

ABSTRACT

Regret minimization is a key component of many algorithms for finding Nash equilibria in imperfect-information games. To scale to games that cannot fit in memory, we can use search with value functions. However, calling the value functions repeatedly in search can be expensive. Therefore, it is desirable to minimize regret in the search tree as fast as possible. We propose to accelerate the regret minimization by introducing a general "learning not to regret" framework, where we meta-learn the regret minimizer. The resulting algorithm is guaranteed to minimize regret in arbitrary settings and is (meta)-learned to converge fast on a selected distribution of games. Our experiments show that meta-learned algorithms converge substantially faster than prior regret minimization algorithms.

KEYWORDS

meta learning, regret minimization, algorithmic game theory, online learning

1 INTRODUCTION

Regret minimization is a general, online convex optimization concept, where an agent repeatedly makes decision against an unknown environment [38]. Regret measures the difference between the accumulated reward and the reward that a best time-independent action would have received in hindsight. An algorithm is called regret minimizing if its regret grows sub-linearly — the average regret converges to zero [3]. Regret minimization has an elegant and important connection to games. If all players employ a regret minimizer, then their average strategy converges to a coarse correlated equilibrium [16, 18]. More importantly, in two-player zero-sum games, the average strategy converges to a Nash equilibrium. Regret minimization has become the key building block of many algorithms for finding Nash equilibria in imperfect-information games [6, 8, 10, 21, 27] (just to name a few).

To converge to an equilibrium, it is desirable to drive the average regret down as quickly as possible. In strictly adversarial settings, the convergence can be no faster than $O(T^{-1/2})$ [23]. In self-play setting, faster convergence is possible and some algorithms provably enjoy $O\left(T^{-1}\right)$. But in practice, the algorithms are much faster than the worst-case bound suggests. For example, CFR+ [34] essentially solved limit texas holdem poker with tabular self-play — one of the largest imperfect information games to be solved to this day [5]. It required only $1,579$ iterations to produce the final strategy, orders of magnitude less than the theoretical bound suggests. However, scaling to larger games requires using value functions, as the games are cannot fit in memory.

Value functions assume adversarial strategies in the subgames. Because of this, the rate of convergence in the search tree tends to be slow and closer to the adversarial bound than the self-play bound. Additionally, calling the value functions is expensive, as they are typically represented by a large neural network.

To accelerate the regret minimization, we turn to the meta-learning paradigm, namely a variant of learning to learn [1]. We learn a small neural network that serves as the regret minimizer and train it directly on a distribution of games of interest.

We start with matrix-form games and show that our learned algorithms greatly outperform the previous methods. We then turn our attention to sequential decision games [20]. In these settings, one can decompose the overall regret to individual (counterfactual) regret for the information states [14, 39]. Our approach again significantly outperforms previous state-of-the-art methods — on the very games they were designed to excel at.

While the meta-learned algorithm can produce extraordinary fast convergence for the distribution of games we train on (e.g. poker games), it can be at the cost of performance (or even lack of convergence) for the out-of-distribution games. To provide the convergence guarantees, we introduce meta-learning within the predictive regret framework [15]. Predictive regret minimization provides convergence guarantees regardless of the prediction, while a better prediction guarantees faster convergence – a perfect prediction results in zero regret [15]. This allows us to meta-learn the predictions for the distribution in question, ensuring convergence guarantees for any game. This results in an algorithm that combines the best of the worlds: fast convergence for the class of games in question while providing strong convergence guarantees elsewhere.

2 BACKGROUND

An online algorithm m for the regret minimization task repeatedly interacts with an environment through d available actions. The environment can be influenced by an adversary, or even unknown. At each step t, the algorithm specifies a strategy $\boldsymbol{\sigma}_t$ from a probability simplex Δ^d and observes the subsequent reward $\boldsymbol{x}_t \in \mathbb{R}^d$ coming back from the environment. A sequence of such strategies and rewards, up to a horizon T, is

$$\boldsymbol{\sigma}_0 \to \boldsymbol{x}_1, \boldsymbol{\sigma}_1 \to \boldsymbol{x}_2, \boldsymbol{\sigma}_2 \to \cdots \to \boldsymbol{x}_{T-1}, \boldsymbol{\sigma}_{T-1} \to \boldsymbol{x}_T. \quad (1)$$

The instantaneous regret experienced in a step of the sequence is

$$\boldsymbol{r}(\boldsymbol{\sigma}, \boldsymbol{x}) = \boldsymbol{x} - \langle \boldsymbol{\sigma}, \boldsymbol{x} \rangle \boldsymbol{1},$$

and the cumulative regret over the entire sequence is

$$\boldsymbol{R}_T = \sum_{t=1}^{T} \boldsymbol{r}(\boldsymbol{\sigma}_{t-1}, \boldsymbol{x}_t).$$

We say the algorithm is a regret minimizer, if the external regret $R_T^{\text{ext}} = \max_{a \in \{1,\dots,d\}} \sum_{t=1}^{T} \boldsymbol{r}(\delta_a, \boldsymbol{x}_t)$ grows sublinearly in T, where δ_a is a Kronecker delta. We will use the online algorithm in two-player zero-sum matrix games and sequential imperfect-information games. We refer the reader to [20] for a more formal treatment of these games, which we omit for brevity. We employ the online algorithms at each state a player $i \in \{1, 2\}$ can't distinguish (i.e. at each infostate), as in prior work on counterfactual regret minimization [40].

In a sequential game, the players take turns making decisions at each infostate, until they arrive at a terminal state z where the player receives its reward $u_i(z) \in \mathbb{R}$. We denote the maximum difference in rewards as $\Delta_{\max} = \max_z u_i(z) - \min_z u_i(z)$. The expected reward (in the whole game) is $u_i(\boldsymbol{\sigma}) = \mathbb{E}_{z\sim\boldsymbol{\sigma}}\, u_i(z)$, where $\boldsymbol{\sigma} = (\boldsymbol{\sigma}_1, \boldsymbol{\sigma}_2)$ is a strategy profile of all players at all infostates.

The best response to the other player's strategy $\boldsymbol{\sigma}_{\text{-}i}$ is $br(\boldsymbol{\sigma}_{\text{-}i}) \in \arg\max_{\boldsymbol{\sigma}_i} u_i(\boldsymbol{\sigma}_i, \boldsymbol{\sigma}_{\text{-}i})$. A profile is a Nash equilibrium $\boldsymbol{\sigma}^*$, if the individual player strategies are mutual best responses. The game value $u^* = u_1(\boldsymbol{\sigma}^*)$ is the utility player 1 can achieve under a Nash equilibrium. Finally, the exploitability of player i strategy (i.e. the gap from a Nash equilibrium) is $expl_i(\boldsymbol{\sigma}_i) := \left[u_i(\boldsymbol{\sigma}^*) - \min_{\boldsymbol{\sigma}_{\text{-}i}} u_i(\boldsymbol{\sigma}_i, \boldsymbol{\sigma}_{\text{-}i})\right]$.

3 PRIOR WORK

From the very dawn of the field, search with value functions was a fundamental concept of computer games research. Turing's chess algorithm from 1950 was able to think two moves ahead [12], and Shannon's work on chess from 1950 includes an extensive section on evaluation functions to be used within a search [32]. Samuel's checkers program from 1959 already combines search and value functions that are learned through self-play and bootstrapping [25]. TD-Gammon improves upon those ideas and uses neural networks to learn those complex value functions — only to be again used within search [35]. The combination of decision-time search and value functions has been present in the remarkable milestones where computers bested their human counterparts in challenging games — DeepBlue for Chess [11] and AlphaGo for Go [33].

Recently, this powerful framework of search aided with (learned) value functions has been extended to imperfect information games [26]. Regret minimization has quickly become the state-of-the-art method for search [7, 9, 28, 31, 37**?**] with the notable exceptions of [13, 24].

Meta learning has a long history when used for optimization [1, 29, 30, 36]. Prior work has considered bandits in Bayesian settings [2], and meta learning paradigm in games has also been used to "warm start" the initial strategies in games [17].

This work considers the meta-learning paradigm in the context of regret minimization in games. Rather than hand-crafting an optimization rule, one can learn a rule that is tailor-made for the domain in hand. This structure of the learning problem closely resembles the "learning to learn by gradient descent by gradient descent" framework [1].

4 LEARNING NOT TO REGRET

We first describe the meta-learning framework for regret minimization. Then we introduce two variants of meta-learned algorithms, with and without regret minimization guarantees. Finally, we describe the setup in which we would like to minimize the regrets.

4.1 Meta-learning framework

Given a distribution of regret-minimization tasks $\mathcal{F}$, we aim find an online algorithm m which efficiently minimizes regret after T steps. Formally, let $\boldsymbol{r}(\delta_a, \boldsymbol{x}_t | f, \theta)$ be the instantaneous regret for playing only action a, experiencing reward $\boldsymbol{x}_t$ at step $t \leq T$, given a task $f \sim \mathcal{F}$, and some parameters θ of the online algorithm. We define the expected cumulative swap regret as

$$\mathcal{L}(\theta) = \mathbb{E}_{f\sim\mathcal{F}}\left[\sum_{t=1}^{T} \max_{a\in\{1,\dots,d\}} \boldsymbol{r}(\delta_a, \boldsymbol{x}_t | f, \theta)\right]. \quad (2)$$

Note that minimizing swap regret bounds the gap to a correlated equilibrium, while minimizing external regret bounds the gap to a (less restrictive) coarse-correlated equilibrium [4]. As Nash equilibria are equivalent to each of the correlation classes in the zero-sum setting, we are free to choose whichever regret we would like to minimize.

We represent the algorithm m via a neural network θ and train it to minimize (2). Minimizing swap regret rather then the external regret leads to more stable learning. It forces the algorithm to produce a strategy with low regret in every step, rather than at the end of the optimization.

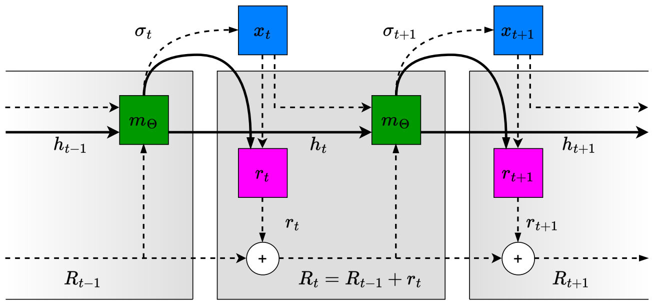
By utilizing a recurrent architecture we are also able to represent algorithms that are history and/or time dependent. The dependence is captured by the hidden state $\boldsymbol{h}$ of the network. See also Section 5 for more details.

4.2 Neural Online Algorithm

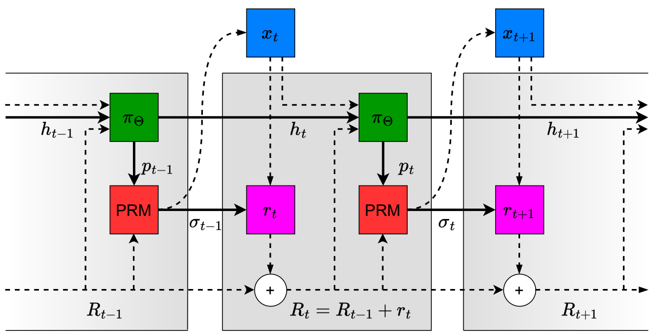
The simplest option is to directly parameterize the online algorithm as m_θ. We refer to this setup as neural online algorithm (NOA). The m_θ receives as input the rewards $\boldsymbol{x}_t$ and keeps track of its hidden state $\boldsymbol{h}_t$. The gradient $\partial\mathcal{L}/\partial\theta$ can be estimated by sampling a batch of tasks and applying backpropagation through the computation graph as shown in Figure 1a. The gradient originates in the collection of regrets $\boldsymbol{r}_{1\dots T}$ and propagates through the strategies $\boldsymbol{\sigma}_{0\dots T-1}$ and hidden states $\boldsymbol{h}_{0\dots T-1}$. We don't allow the gradient to propagate through the rewards $\boldsymbol{x}_{1\dots T}$ or the cumulative regrets $\boldsymbol{R}_{1\dots T}$. Thus, the only way to influence the earlier optimization steps is through the hidden states $\boldsymbol{h}_{0\dots T-1}$ of the neural network. In experiments, we observe strong empirical performance for NOA. However, there is no guarantee that m_θ is regret minimizing.

4.3 Neural Predictive Regret Matching

Next, we turn to the recent predictive regret matching (PRM) [15] to help us with regret minimization guarantees, see also Algorithm 1. The algorithm has two functions, NextStrategy and ObserveReward, which alternate over the sequence (1). The PRM



(a) Neural online algorithm (NOA).



(b) Neural predictive regret matching (NPRM).

Figure 1: Computational graphs of the proposed online algorithms. The gradient flows only through the solid edges. The $\boldsymbol{h}_t$ denotes the hidden state of the neural network.

is an extension of regret matching (RM) and uses an additional predictor π. The predictor $\pi : \mathbb{R}^d \to \mathbb{R}^d$ makes a prediction $\boldsymbol{p}_t$ of the next anticipated reward $\boldsymbol{x}_{t+1}$. The PRM algorithm incorporates $\boldsymbol{p}_t$ to compute the next strategy $\boldsymbol{\sigma}_t$, by regret matching over predictive regret ξ_t [1]. The RM algorithm can be instantiated as PRM with $\pi = \mathbf{0}$. Unless stated otherwise, we use PRM with a simple predictor $\pi = \boldsymbol{x}_t$, i.e. it predicts the next reward will be the same as the current one.

We introduce neural predictive regret matching (NPRM), a variant of PRM which uses a predictor π_θ parameterized by a neural network θ. The π_θ receives as input the rewards $\boldsymbol{x}_t$, cumulative regret $\boldsymbol{R}_t$ and keeps track of its hidden state $\boldsymbol{h}_t$. We train π_θ to minimize (2), just like NOA. The computational graph is shown in Figure 1b. The output of the network $\boldsymbol{p}_t$ is used by NextStrategy to obtain the strategy $\boldsymbol{\sigma}_t$. Similar to NOA, the gradient originates in the collection of regrets $\boldsymbol{r}_{1\ldots T}$ and propagates through the strategies $\boldsymbol{\sigma}_{0\ldots T-1}$, hidden states $\boldsymbol{h}_{0\ldots T-1}$ and additionally the predictions $\boldsymbol{p}_{1\ldots T-1}$. We do not propagate the gradient through the the rewards $\boldsymbol{x}_{1\ldots T}$ or through the cumulative regrets $\boldsymbol{R}_{1\ldots T}$. Again, any time-dependence comes only through the hidden states $\boldsymbol{h}_{0\ldots T-1}$.

Importantly, we show that cumulative regret of NPRM grows sub-linearly:

[1]Note the prediction changes the actual observed $\boldsymbol{x}_{t+1}$, unless we are at a fixed point.

Algorithm 1: Predictive regret matching [15]

```
R_0 ← 0 ∈ ℝ^d,   σ_0 ← 1/d ∈ Δ^d
function NextStrategy()
  | ξ_t ← [R_{t-1} + r(σ_{t-1}, p_t)]^+
  | if ξ_t ≠ 0    return σ_t ← ξ_t / ‖ξ_t‖_1
  | else          return σ_t ← arbitrary point in Δ^d
function ObserveReward(x_t)
  | R_t ← R_{t-1} + r(σ_{t-1}, x_t)
  | p_t ← π(x_t)
```

Theorem 1 (Correctness of Neural-Predicting). *Let $\pi_\theta : \mathbb{R}^d \to [-\Delta_{\max}, \Delta_{\max}]^d$ be a regret predictor and let m be a PRM algorithm. Then m that uses π_θ is a regret minimizer.*

Proof. Since the reward $\boldsymbol{x}$ and prediction $\boldsymbol{p}$ for any action is bounded by the maximum utility difference $\Delta_{\max}$, for arbitrary $\boldsymbol{p}$ it holds

$$\|\boldsymbol{r}(\boldsymbol{\sigma}, \boldsymbol{x}) - \boldsymbol{r}(\boldsymbol{\sigma}, \boldsymbol{p})\|_2 \le \Delta_{\max} d.$$

Using the PRM regret bound [15, Thm 3], we obtain

$$\begin{aligned} R_T &\le \sqrt{2} \left(\sum_{t=1}^{T} \|\boldsymbol{r}(\boldsymbol{\sigma}_{t-1}, \boldsymbol{x}_t) - \boldsymbol{r}(\boldsymbol{\sigma}_{t-1}, \boldsymbol{p}_t)\|_2 \right)^{\frac{1}{2}} \\ &\le \sqrt{2}\,(\Delta_{\max} d T)^{\frac{1}{2}} \in O\left(\sqrt{T}\right). \end{aligned}$$

□

As NPRM is regret minimizing regardless of the prediction $\boldsymbol{p}_t$, our network outputs $\boldsymbol{p}_t$, rather than strategy $\boldsymbol{\sigma}_t$ as NOA does. This allows us to achieve the best of both words — adaptive learning and algorithm with a small cumulative regret in our domain, while keeping $O(\sqrt{T})$ worst case regret guarantees.

In self-play settings, the predictions $\boldsymbol{p}_t = \boldsymbol{x}_t$ used by PRM are often close to $\boldsymbol{x}_{t+1}$, as the opponent's strategy changes gradually. This substantially improves the convergence speed when compared to the non-predictive variants [15]. The NPRM can recover such a predictor: the network π_θ can simply represent the identity function, as it receives $\boldsymbol{x}_t$ on the input.

However, in a setting against a best responding opponent (or when value functions are used), PRM can perform poorly. As the best response can drastically change at each time step, it is difficult for PRM to make good predictions. In our experiments (Section 5), we observe that PRM has degraded performance compared to self-play, in fact it is even worse than when we used no predictions (i.e. RM). In contrast, NPRM can meta-learn the predictor to the task distribution $\mathcal{F}$. In experiments, we observe that NPRM vastly outperforms PRM.

4.4 Best Response, Search and Value Functions

In imperfect information search, regret minimization is used in combination with a value function. Regret minimizers are used to minimize counterfactual regret at each infostate of the search tree. The value function returns values under a zero-regret (optimal) policy in the subgame(s), given the policy in the search tree.

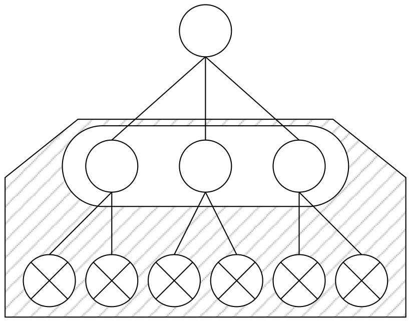


Figure 2: Public game tree of a matrix game. The enclosed areas highlight infostates for each player. Highlighted area is solved and represented by the value function.

In our simplified setting, we imitate the value function by ensuring the opponent utilizes a (zero-regret) best response to the player's strategy. Then a zero-regret policy of the player with such value function is a Nash equilibrium.

4.4.1 Matrix Games. Matrix games allow us to investigate the setup in its simplest form. This is because a matrix game corresponds to a simple sequential game, and we can perform one-step lookahead search for the first player, while the value function solves a single-state subgame for the second player. This approach is visualized in Figure 2. Note that as we update the search tree policy, the inputs to the subgame change, and so does the subgame optimal policy and the values as well.

4.4.2 Sequential Games. The situation gets more complicated in a sequential game, where one needs to minimize the counterfactual regret [40]. These games are typically solved using counterfactual regret minimization (CFR), which decomposes the full regret into regrets at each infostate and minimizes them individually [39]. Again, we will test our algorithms against a best-responding opponent.

5 EXPERIMENTS

For both NOA and NPRM, the neural network architecture we use is a two layer LSTMs. For NOA, these two layers are followed by a fully-connected layer with the softmax activation. For NPRM, we additionally scale all outputs by $\Delta_{\max}$, ensuring any reward vector can be represented by the network. We minimize Eq. (2) for $T = 32$ iterations over 1024 epochs, using mini-batch of 4 games[2] and Adam optimizer with a learning rate 0.001. Other hyperparameters[3] were found using a small grid search and only best results are presented. For evaluation, we run the algorithm for $2T = 64$ iterations to see whether our networks can generalize outside of the horizon T and keep minimizing regret as the search continues. We train and evaluate both NOA and NPRM and compare our methods against (RM) and (PRM). Our results are presented in Figure 3.

5.1 Matrix Games

We first evaluate our method on matrix games, allowing us to focus on a single-step search and single-state value function. Specifically, we use `rock_paper_scissors`, perturbing the "rock-scissors" payoff with $X \sim \mathcal{U}(-\varepsilon, \varepsilon)$, see Appendix A.1 for more details. We present results for $\varepsilon \in \{0, 1/4\}$, for both NOA and NPRM in Figures 3.

	$expl_i(\overline{\sigma}_T)$	Speed-up	$expl_i(\overline{\sigma}_{2T})$	Speed-up
RM	$5.07 \cdot 10^{-2}$	1	$2.86 \cdot 10^{-2}$	1
PRM	$4.38 \cdot 10^{-2}$	1.28	$2.44 \cdot 10^{-2}$	1.19
NOA	$\mathbf{2.96 \cdot 10^{-3}}$	**24.19**	$\mathbf{1.91 \cdot 10^{-3}}$	**20.33**
NPRM	$2.11 \cdot 10^{-2}$	3.12	$9.56 \cdot 10^{-3}$	3.56

(a) `rock_paper_scissors` **($\varepsilon = 1/4$).**

	$expl_i(\overline{\sigma}_T)$	Speed-up	$expl_i(\overline{\sigma}_{2T})$	Speed-up
RM	$3.9 \cdot 10^{-2}$	1	$2.76 \cdot 10^{-2}$	1
PRM	$5.06 \cdot 10^{-2}$	0.56	$3.92 \cdot 10^{-2}$	0.5
NOA	$\mathbf{6.14 \cdot 10^{-3}}$	**46.53**	$\mathbf{3.44 \cdot 10^{-3}}$	**73.7**
NPRM	$1.19 \cdot 10^{-2}$	9.72	$7.15 \cdot 10^{-3}$	16

(b) `kuhn_poker` **($\varepsilon = 1/4$).**

Table 1: Relative performance of NOA, NPRM, and PRM compared to RM in different games. We show the exploitability of the average strategy $\overline{\sigma}$ of each algorithm after T and $2T$ steps, as well as the speed-up over RM, i.e. how many more steps would RM require to achieve lower exploitability.

As one would expect, in the case without the perturbations, NOA can simply learn the least-exploitable strategy, which leads to significantly faster convergence compared to (P)RM, in our case by several orders of magnitude. Notice that while the exploitability of the average strategy is low, the average regret remains high. While surprising at first, this can in fact happen; see Appendix B for more details. NPRM produces much higher exploitability in this setting compared to NOA. We hypothesize that a reason is the vanishing, resp. exploding gradient of the PRM when the cumulative regret is large[4], resp. small.

However, our methods outperform both RM and PRM even in the perturbed setting. At the end of the evaluation, i.e. after $2T = 64$ iterations, NOA achieve almost an order of magnitude lower exploitability. This translate to RM needing 20.3 and 3.6 times more iterations than NOA and NPRM respectively to achieve the same exploitability, see Table 1a. Finally, we can see that NPRM is indeed able to minimize regret far above the number of iterations it has been trained to do so (see Theorem 1). Perhaps surprisingly, the same is true for NOA.

Notice that RM often outperforms PRM, even in sequential games. While surprising at first, the reason is that we minimize regret against a value function rather than against self-play opponent. PRM performs well in self-play settings, where the last-observed reward is a good prediction of the next one as the opponent does not radically change their strategy between iterations. This is no longer the case when the values are coming from value functions,

[2]We perform the update using all infostates of each of the games.
[3]Specifically the size of the LSTM layer, L2 penalty, and gradient clipping constant.
[4]Actually, the gradient is zero if the regret is positive for at most one action.

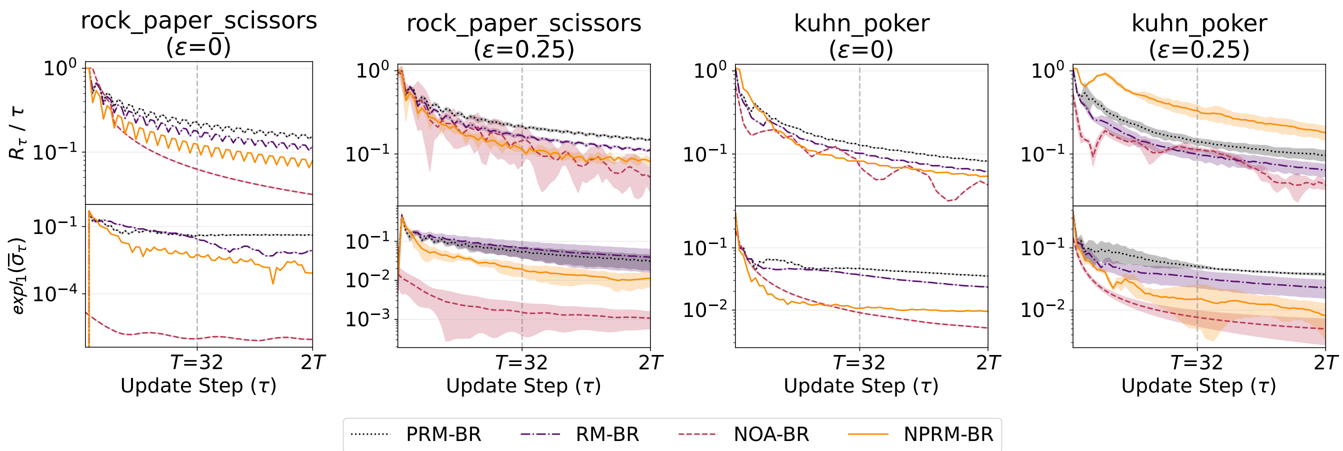


Figure 3: Comparison of regret minimization algorithms against best responding opponent. The y axis are in logaritmic scale and are aligned where possible. The top row shows the average regret at time τ and the bottom row shows exploitability of the average strategy $\overline{\sigma}_\tau$. Vertical dashed line separates two regimes: training (up to T steps) and generalization (from T to $2T$ steps). Colored areas show standard error for non-zero-noise settings.

where arbitrarily small modification of the input can lead to large changes of the output [26].

5.2 Sequential Games

To evaluate our methods in sequential setting, we use a small standard benchmark `kuhn_poker`. The distribution $\mathcal{F}$ is generated by changing terminal rewards of certain fixed terminal histories, see Appendix A.2 for more details.

Our setup allows NOA and NPRM to learn a contextualized regret minimizer, resulting in per-state specific regret minimization. This is achieved by adding features to the input of the network. These can encode cards in games like poker, or current board representation in other games. As the games in our experiments are relatively small, we opt for a simple one-hot encoding. As described in Section 4.4, we utilize the setup of best responding opponent, resulting in CFR-BR [19].

Our results, presented in Figure 3 show our method can outperform (P)RM even in a sequential setting. Specifically, even in the perturbed setting after $2T$ steps, it would take 73.7, resp. 16-times more steps for RM to reach the final exploitability of NOA, resp. NPRM (see Table 1b).

6 CONCLUSION

We introduced two new algorithms to meta-learn regret minimizers for a distribution of games in a new "learning not to regret" framework. One algorithm has better empirical performance, while the other guarantees converge on any game. We evaluated our algorithms in a general setting of playing with a value function. Our experiments showed both outperform previous state-of-the-art methods, sometimes by several orders of magnitude.

In the future, we plan to use our approach within the continual resolving framework [21], and extend our work to general-sum games or apply it within to framework of hindsight rationality [22] for games that change over time.

A BENCHMARK GAMES

A.1 `rock_paper_scissors`

To obtain a distribution of matrix games, we perturb the "rock-scissors" payoff. The utility is thus given by

$$u_1 = -u_2 = \begin{pmatrix} 0 & -1 & 1+X \\ 1 & 0 & -1 \\ -1 & 1 & 0 \end{pmatrix},$$

where $X \sim \mathcal{U}(-\varepsilon, \varepsilon)$. The equilibrium of the unperturbed game is uniform, making it a particularly simple problem to learn. For $X > 0$, playing scissors more and rock less is beneficial.

A.2 `kuhn_poker`

We generate a distribution of games by changing the payoff agents get when the first player gets the king, and the second got the queen. This corresponds to changing 5 out of the 30 terminal utilities. Just like before, the payoffs are modified by $X \sim \mathcal{U}(-\varepsilon, \varepsilon)$ each of the payoffs independently.

B REGRET VS EXPLOITABILITY

We have seen in Section 5 that the average regret can remain high even if exploitability is small. While surprising at first, we present one example where this happens while playing agains the best response. Consider a sequence of strategies

$$\sigma_n^t(\epsilon) = \left(\frac{1}{3} + 2\epsilon, \frac{1}{3} - \epsilon, \frac{1}{3} - \epsilon\right).$$

In `rock_paper_scissors`, this strategy prefers rock too much, and thus the best response opponent will always choose paper, resulting in external regret

$$\boldsymbol{r}_t(\epsilon) = (-1, 0, 1) - \epsilon \mathbf{1} \Rightarrow \boldsymbol{R}_\tau = \tau(1 - \epsilon).$$

However, the exploitability $expl_n(\sigma_n^\tau) = \epsilon$ can be arbitrarily small.

REFERENCES

[1] Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando De Freitas. 2016. Learning to learn by gradient descent by gradient descent. *Advances in neural information processing systems* 29 (2016).

[2] Mohammadjavad Azizi, Branislav Kveton, Mohammad Ghavamzadeh, and Sumeet Katariya. 2022. Meta-Learning for Simple Regret Minimization. *arXiv preprint arXiv:2202.12888* (2022).

[3] David Blackwell et al. 1956. An analog of the minimax theorem for vector payoffs. *Pacific J. Math.* 6, 1 (1956), 1–8.

[4] Avrim Blum and Yishay Mansour. 2007. From external to internal regret. *Journal of Machine Learning Research* 8, 6 (2007).

[5] Michael Bowling, Neil Burch, Michael Johanson, and Oskari Tammelin. 2015. Heads-up limit hold'em poker is solved. *Science* 347, 6218 (2015), 145–149.

[6] Noam Brown, Anton Bakhtin, Adam Lerer, and Qucheng Gong. 2020. Combining Deep Reinforcement Learning and Search for Imperfect-Information Games. *arXiv preprint arXiv:2007.13544* (2020).

[7] Noam Brown, Anton Bakhtin, Adam Lerer, and Qucheng Gong. 2020. Combining deep reinforcement learning and search for imperfect-information games. *Advances in Neural Information Processing Systems* 33 (2020), 17057–17069.

[8] Noam Brown and Tuomas Sandholm. 2018. Superhuman AI for heads-up no-limit poker: Libratus beats top professionals. *Science* 359, 6374 (2018), 418–424.

[9] Noam Brown and Tuomas Sandholm. 2018. Superhuman AI for heads-up no-limit poker: Libratus beats top professionals. *Science* 359, 6374 (2018), 418–424.

[10] Noam Brown and Tuomas Sandholm. 2019. Superhuman AI for multiplayer poker. *Science* 365, 6456 (2019), 885–890.

[11] Murray Campbell, A Joseph Hoane Jr, and Feng-hsiung Hsu. 2002. Deep blue. *Artificial intelligence* 134, 1-2 (2002), 57–83.

[12] B Jack Copeland. 2004. *The essential Turing*. Clarendon Press.

[13] Meta Fundamental AI Research Diplomacy Team (FAIR)†, Anton Bakhtin, Noam Brown, Emily Dinan, Gabriele Farina, Colin Flaherty, Daniel Fried, Andrew Goff, Jonathan Gray, Hengyuan Hu, et al. 2022. Human-level play in the game of Diplomacy by combining language models with strategic reasoning. *Science* 378, 6624 (2022), 1067–1074.

[14] Gabriele Farina, Christian Kroer, and Tuomas Sandholm. 2019. Online convex optimization for sequential decision processes and extensive-form games. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 1917–1925.

[15] Gabriele Farina, Christian Kroer, and Tuomas Sandholm. 2021. Faster game solving via predictive blackwell approachability: Connecting regret matching and mirror descent. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35. 5363–5371.

[16] James Hannan. 1957. Approximation to Bayes risk in repeated play. *Contributions to the Theory of Games* 3 (1957), 97–139.

[17] Keegan Harris, Ioannis Anagnostides, Gabriele Farina, Mikhail Khodak, Zhiwei Steven Wu, and Tuomas Sandholm. 2022. Meta-Learning in Games. *arXiv preprint arXiv:2209.14110* (2022).

[18] Sergiu Hart and Andreu Mas-Colell. 2000. A simple adaptive procedure leading to correlated equilibrium. *Econometrica* 68, 5 (2000), 1127–1150.

[19] Michael Johanson, Nolan Bard, Neil Burch, and Michael Bowling. 2012. Finding optimal abstract strategies in extensive-form games. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 26. 1371–1379.

[20] Vojtěch Kovařík, Martin Schmid, Neil Burch, Michael Bowling, and Viliam Lisý. 2019. Rethinking formal models of partially observable multiagent decision making. *arXiv preprint arXiv:1906.11110* (2019).

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

[22] Dustin Morrill, Ryan D'Orazio, Reca Sarfati, Marc Lanctot, James R Wright, Amy R Greenwald, and Michael Bowling. 2021. Hindsight and sequential rationality of correlated play. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35. 5584–5594.

[23] Noam Nisan, Tim Roughgarden, Éva Tardos, and Vijay V. Vazirani. 2007. *Algorithmic Game Theory*. Cambridge University Press, New York, NY, USA.

[24] Julien Perolat, Bart De Vylder, Daniel Hennes, Eugene Tarassov, Florian Strub, Vincent de Boer, Paul Muller, Jerome T Connor, Neil Burch, Thomas Anthony, et al. 2022. Mastering the game of Stratego with model-free multiagent reinforcement learning. *Science* 378, 6623 (2022), 990–996.

[25] Arthur L Samuel. 1959. Some studies in machine learning using the game of checkers. *IBM Journal of research and development* 3, 3 (1959), 210–229.

[26] Martin Schmid. 2021. Search in Imperfect Information Games. *arXiv preprint arXiv:2111.05884* (2021).

[27] Martin Schmid, Matej Moravcik, Neil Burch, Rudolf Kadlec, Josh Davidson, Kevin Waugh, Nolan Bard, Finbarr Timbers, Marc Lanctot, Zach Holland, et al. 2021. Player of games. *arXiv preprint arXiv:2112.03178* (2021).

[28] Martin Schmid, Matej Moravcik, Neil Burch, Rudolf Kadlec, Josh Davidson, Kevin Waugh, Nolan Bard, Finbarr Timbers, Marc Lanctot, Zach Holland, et al. 2021. Player of games. *arXiv preprint arXiv:2112.03178* (2021).

[29] Jürgen Schmidhuber. 1992. Learning to control fast-weight memories: An alternative to dynamic recurrent networks. *Neural Computation* 4, 1 (1992), 131–139.

[30] Jürgen Schmidhuber. 1993. A neural network that embeds its own meta-levels. In *IEEE International Conference on Neural Networks*. IEEE, 407–412.

[31] Jack Serrino, Max Kleiman-Weiner, David C Parkes, and Josh Tenenbaum. 2019. Finding friend and foe in multi-agent games. *Advances in Neural Information Processing Systems* 32 (2019).

[32] Claude E Shannon. 1950. XXII. Programming a computer for playing chess. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 41, 314 (1950), 256–275.

[33] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. 2016. Mastering the game of Go with deep neural networks and tree search. *nature* 529, 7587 (2016), 484–489.

[34] Oskari Tammelin. 2014. Solving large imperfect information games using CFR+. *arXiv preprint arXiv:1407.5042* (2014).

[35] Gerald Tesauro. 1995. Temporal difference learning and TD-Gammon. *Commun. ACM* 38, 3 (1995), 58–68.

[36] Sebastian Thrun and Lorien Pratt. 2012. *Learning to learn*. Springer Science & Business Media.

[37] Ryan Zarick, Bryan Pellegrino, Noam Brown, and Caleb Banister. 2020. Unlocking the potential of deep counterfactual value networks. *arXiv preprint arXiv:2007.10442* (2020).

[38] Martin Zinkevich. 2003. Online convex programming and generalized infinitesimal gradient ascent. In *Proceedings of the 20th international conference on machine learning (icml-03)*. 928–936.

[39] Martin Zinkevich, Michael Johanson, Michael Bowling, and Carmelo Piccione. 2008. Regret minimization in games with incomplete information. In *Advances in neural information processing systems*. 1729–1736.

[40] Martin Zinkevich, Michael Johanson, Michael Bowling, and Carmelo Piccione. 2008. Regret minimization in games with incomplete information. In *Advances in neural information processing systems*. 1729–1736.