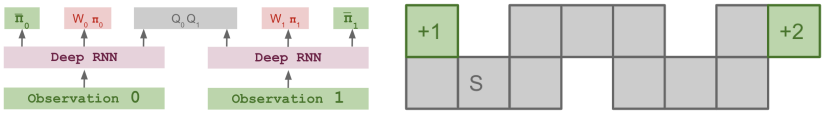


(a) (b)

Figure 5: The (a) Multi-headed network architecture, and (b) Exploration example.

Appendices

A Baseline Details and Hyperparameters

For MC-RCFR, we sweep over all combinations of the exploration parameter, using a (learned) state-action baseline ([27]), and learning rate $(\epsilon, b, \alpha) \in \{0.25, 0.5, 0.6, 0.7, 0.9, 0.95, 1.0\} \times \{\text{True}, \text{False}\} \times \{0.0001, 0.00005, 0.00001\}$, where each combination is averaged over five seeds. We found that higher exploration values worked consistently better, which matches the motivation of the robust sampling technique (corresponding to $\epsilon = 1$) presented in ([22]) as it leads to reduced variance since part of the correction term is constant for all histories in an information state. The baseline helped significantly in the larger game with more variable-length episodes.

For NFSP, we keep a set of hyperparameters fixed, in line with ([21]) and ([16]): anticipatory parameter $\eta = 0.1$, ϵ-greedy decay duration $20M$ steps, reservoir buffer capacity $2M$ entries, replay buffer capacity $200k$ entries, while sweeping over a combination of the following hyperparameters: ϵ-greedy starting value $\{0.06, 0.24\}$, RL learning rate $0.1, 0.01, 0.001$, SL learning rate $\{0.01, 0.001, 0.005\}$, DQN target network update period of $\{1000, 19200\}$ steps (the later is equivalent to 300 network-parameter updates). Each combination was averaged over three seeds. Agents were trained with the ADAM optimizer, using MSE loss for DQN and one gradient update step using mini-batch size 128, every 64 steps in the game.

Finally, note that there are at least four difference in the results, experimental setup, and assumptions between MC-RCFR and OS-DeepCFR reported in ([31]):

1. ([31]) uses domain expert input features which do not generalize outside of poker. The neural network architecture we use is a basic MLP with raw input representations, whereas ([31]) uses a far larger network. Our empirical results on benchmark games compare the convergence properties of knowledge-free algorithms across domains.
2. The amount of training per iteration is an order of magnitude larger in OS-DeepCFR than our training. In ([31]), every 346 iterations, the Q-network is trained using 1000 minibatches of 512 samples (512000 examples), whereas every 346 iterations we train 346 batches of 128 samples, 44288 examples.
3. MC-RCFR uses standard outcome sampling rather than Linear CFR ([7]).
4. MC-RCFR's strategy is approximated by predicting the OS's average strategy increment rather than sampling from a buffer of previous models.

Our NFSP also does not use any extra enhancements.

A.1 Single-Agent Environments

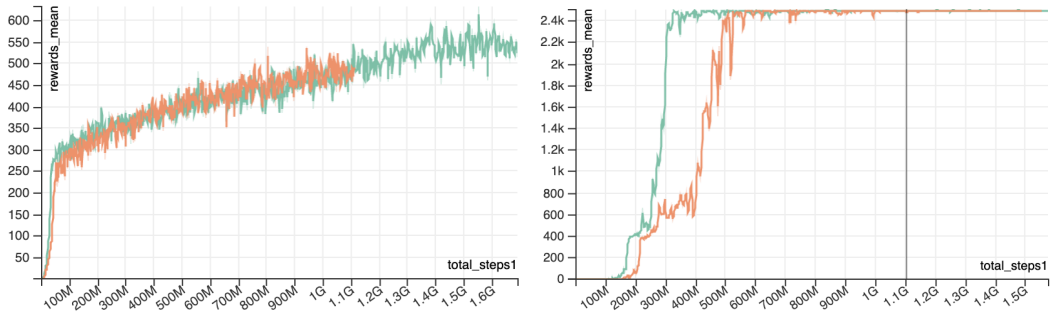
Despite ARMAC being based on commonly-used multiagent algorithms, it has properties that may be desirable in the single-agent setting. First, similar to policy gradient algorithms in the common "short corridor example" ([32], Example 13.1), stochastic policies are representable by definition, since they are normalized positive mean regrets over the actions. This could have a practical effect that entropy bonuses typically have in policy gradient methods, but rather than simply adding arbitrary entropy, the relative regret over the set of past policies is taken into account.

Second, a retrospective agent uses a form of *directed exploration* of different exploration policies ([2]). Here, this is achieved by the simulation (μ_i^T, π_{-i}^t), which could be desirable whenever there is overlapping structure in successive tasks. μ_i^T here is an exploratory policy, which consists of a mixture of all past policies (plus random uniform) played further modulated with different amounts of random uniform exploration (more details are given in Section 3.2). Consider a gridworld illustrated in Fig. 5(b). Green squares illustrate positions where the agent i gets a reward and the game terminates. Most of RL algorithms would find the reward of $+1$ first as it is the closest to the origin S. Once this reward is found, a policy would quickly learn to approach it, and finding reward $+2$ would be problematic. ARMAC, in the meantime, would keep re-running old policies, some of which would pre-date finding reward $+1$, and thus would have a reasonable chance of finding $+2$ by random exploration. This behaviour may also be useful if instead of terminating the game, reaching one of those two rewards would start next levels, both of which would have to be explored.

These properties are not necessarily specific to ARMAC. For example, Politex (another retrospective policy improvement algorithm ([1])) has similar properties by keeping its past approximators intact. Like Politex, we show an initial investigation of ARMAC in Atari in Appendix B. Average strategy sampling MCCFR ([13]) also uses exploration policies that are a mixture of previous policies and uniform random to improve performance over external and outcome sampling variants. However, this exact sampling method cannot be used directly in ARMAC as it requires a model of the game.

B Initial Investigation of ARMAC in the Atari Learning Environment

While performance on Atari is not the main contribution, it should be treated as a health check of the algorithm. Unlike previously tested multiplayer games, many Atari games have a long term credit assignment problem. Some of them, like Montezuma's Revenge, are well-known hard exploration problems. It is interesting to see that ARMAC was able to consistently score 2500 points on Montezuma's Revenge despite not using any auxiliary rewards, demonstrations, or distributional RL as critic. We hypothesize that regret matching may be advantageous for exploration, as it provides naturally stochastic policies which stay stochastic until regrets for other actions becomes negative. We also tested the algorithm on Breakout, as it is a fine control problem. We are not claiming that out results on Atari are state of art - they should be interpreted as a basic sanity check showing that ARMAC could in principle work in this domain.



(a) Breakout

(b) Montezuma Revenge

Figure 6: Performance on Breakout (left) and Montezuma Revenge (right). Results are shown for two seeds.

C Training

Training is done by processing a batch of 64 of trajectories of length 32 at a time. In order to implement a full recall, all unfinished episodes will be continued on the next training

iteration by propagating recurrent network states forward. Each time when one episode finishes at a particular batch entry, a new one is sampled and started to be unrolled from the beginning.

Adam optimized with $\beta_1 = 0.0$ and $\beta_2 = 0.999$ was used for optimization. Hyperparameter selection was done by trying only two learning rates: $5 \cdot 10^{-5}$ and $2 \cdot 10^{-4}$. The results reported use $5 \cdot 10^{-5}$ in all games, including Atari.

D Proof of Lemma 1

Proof. First, let us notice that

$$W_i^T(s,a) = \sum_{h \in s} \frac{\sum_{t=1}^T \eta^{\pi^t}(h)}{\sum_{t=1}^T \eta^{\pi^t}(s)} A_{\pi^t,i}(h,a), \tag{5}$$

$$= \sum_{h \in s} \frac{\sum_{t=1}^T \eta_{-i}^{\pi^t}(h)}{\sum_{t=1}^T \eta_{-i}^{\pi^t}(s)} A_{\pi^t,i}(h,a) \tag{6}$$

$$= \frac{1}{w^T(s)} \sum_{t=1}^T \sum_{h \in s} \eta_{-i}^{\pi^t}(h) A_{\pi^t,i}(h,a), \tag{7}$$

where we used the perfect recall assumption in the first derivation, and we define $w^T(s) = \sum_t \eta_{-i}^{\pi^t}(s)$. Notice that $w^T(s)$ depends on the state only (and not on h). Now the cumulative regret is:

$$\begin{aligned}
R_i^T(s,a) &= \sum_{t=1}^K q_{\pi^t,i}^c(s,a) - v_{\pi^t,i}^c(s) \\
&= \sum_{t=1}^T \eta_{-i}^{\pi^t}(s)\big(q_{\pi^t,i}(s,a) - v_{\pi^k,i}(s)\big) \\
&= \sum_{t=1}^T \eta_{-i}^{\pi^t}(s) \sum_{h \in s} \frac{\eta_{-i}^{\pi^t}(h)}{\eta_{-i}^{\pi^t}(s)} \big(q_{\pi^t,i}(h,a) - v_{\pi^t,i}(h)\big) \\
&= \sum_{t=1}^T \sum_{h \in s} \eta_{-i}^{\pi^t}(h) A_{\pi^t,i}(h,a) \\
&= w^T(s) W_i^T(s,a).
\end{aligned}$$

Finally, noticing that regret matching is not impacted by multiplying the cumulative regret by a positive function of the state, we deduce

$$\frac{R_i^{T,+}(s,a)}{\sum_b R_i^{T,+}(s,b)} = \frac{\big(w^T(s) W_i^T(s,a)\big)^+}{\sum_b \big(w^T(s) W_i^T(s,b)\big)^+} = \frac{W_i^{T,+}(s,a)}{\sum_b W_i^{T,+}(s,b)}.$$

□

E Unbiasedness of $\hat{W}_i^T(s,a)$

Lemma 2. *Consider the case of a tabular representation and define the estimate $\hat{W}_i^T(s,a)$ as the minimizer (over W) of the empirical loss defined over N trajectories*

$$\hat{\mathcal{L}}_{(s,a)}(W) = \frac{1}{N} \sum_{n=1}^N \big[W - A_{\pi^{j_n},i}(h,a)\big]^2 \mathbb{I}\{(h,a) \in \rho^{j_n} \text{ and } h \in s\},$$

where ρ^{j_n} is the n-th trajectory generated by the policy $(\mu_i^T, \pi_{-i}^{j_n})$ where $j_n \sim \mathcal{U}(\{1,\dots,T\})$. Define $N(s,a) = \sum_{n=1}^N \mathbb{I}\{(h,a) \in \rho^{j_n} \text{ and } h \in s\}$ to be the number of trajectories going

through (s,a). *Then* $\hat{W}_i^T(s,a)$ *is an unbiased estimate of* $W_i^T(s,a)$ *conditioned on* (s,a) *being traversed at least once:*

$$\mathbb{E}\big[\hat{W}_i^T(s,a)|N(s,a)>0\big] = W_i^T(s,a).$$

Proof. The empirical loss being quadratic, under the event $\{N(s,a)>0\}$, its minimum is well defined and reached for

$$\hat{W}_i^T(s,a) = \frac{1}{N(s,a)} \sum_{n=1}^{N(s,a)} A_{\pi^{j_n},i}(h_n,a),$$

where $h_n \in s$ is the history of the n-th trajectory traversing s. Let us use simplified notations and write $A_n = A_{\pi^{j_n},i}(h,a)\mathbb{I}\{(h,a)\in\rho^{j_n} \text{ and } h\in s\}$ and $b_n = \mathbb{I}\{(h,a)\in\rho^{j_n} \text{ and } h\in s\}$. Thus

$$\begin{aligned}
\mathbb{E}\Big[\hat{W}_i^T(s,a)\mathbb{I}\Big\{\sum_{m=1}^N b_m>0\Big\}\Big] &= \mathbb{E}\left[\frac{\sum_{n=1}^N A_n\mathbb{I}\big\{\sum_{m=1}^N b_m>0\big\}}{\sum_{m=1}^N b_m}\right] \\
&= \sum_{n=1}^N \mathbb{E}\left[\mathbb{E}\left[\frac{A_n\mathbb{I}\big\{\sum_{m=1}^N b_m>0\big\}}{\sum_{m=1}^N b_m}\Big|\sum_{m=1}^N b_m\right]\right] \\
&= \sum_{n=1}^N \mathbb{E}\left[\mathbb{E}\Big[A_n\Big|\sum_{m=1}^N b_m\Big]\frac{\mathbb{I}\big\{\sum_{m=1}^N b_m>0\big\}}{\sum_{m=1}^N b_m}\right]
\end{aligned}$$

.

Now, $\mathbb{E}\big[A_n\big|\sum_{m=1}^N b_m\big] = \mathbb{E}\big[A_n|b_n\big]\mathbb{E}\big[b_n|\sum_{m=1}^N b_m\big]$ since given b_n, A_n is independent of $\sum_{m=1}^N b_m$. Thus

$$\begin{aligned}
\mathbb{E}\Big[\hat{W}_i^T(s,a)\mathbb{I}\Big\{\sum_{m=1}^N b_m>0\Big\}\Big] &= \sum_{n=1}^N \mathbb{E}\big[A_n|b_n\big]\mathbb{E}\left[\mathbb{E}\Big[\frac{\mathbb{E}\big[b_n\big|\sum_{m=1}^N b_m\big]\mathbb{I}\big\{\sum_{m=1}^N b_m>0\big\}}{\sum_{m=1}^N b_m}\right] \\
&= \sum_{n=1}^N \mathbb{E}\big[A_n|b_n\big]\mathbb{E}\left[\frac{b_n\mathbb{I}\big\{\sum_{m=1}^N b_m>0\big\}}{\sum_{m=1}^N b_m}\right]
\end{aligned}$$

Since $\sum_{n=1}^N \mathbb{E}\Big[\frac{b_n\mathbb{I}\big\{\sum_{m=1}^N b_m>0\big\}}{\sum_{m=1}^N b_m}\Big] = \mathbb{E}\Big[\frac{\sum_{n=1}^N b_n}{\sum_{m=1}^N b_m}\mathbb{I}\big\{\sum_{m=1}^N b_m>0\big\}\Big] = \mathbb{P}\big(\sum_{m=1}^N b_m>0\big)$, by a symmetry argument we deduce $\mathbb{E}\Big[\frac{b_n\mathbb{I}\big\{\sum_{m=1}^N b_m>0\big\}}{\sum_{m=1}^N b_m}\Big] = \frac{1}{N}\mathbb{P}\big(\sum_{m=1}^N b_m>0\big)$ for each n. Thus

$$\begin{aligned}
\mathbb{E}\Big[\hat{W}_i^T(s,a)\Big|N(s,a)>0\Big] &= \mathbb{E}\Big[\hat{W}_i^T(s,a)\Big|\sum_{m=1}^N b_m>0\Big] \\
&= \frac{\mathbb{E}\Big[\hat{W}_i^T(s,a)\mathbb{I}\Big\{\sum_{m=1}^N b_m>0\Big\}\Big]}{\mathbb{P}\big(\sum_{m=1}^N b_m>0\big)} \\
&= \frac{1}{N}\sum_{n=1}^N \mathbb{E}[A_n|b_n] = \mathbb{E}[A_1|b_1]
\end{aligned}$$

which is the expectation of the advantage $A_{\pi^j,i}(h,a)$ conditioned on the trajectory ρ^j going through $h\in s$, i.e. $W_i^T(s,a)$ as defined in (3). □

F Neural Network Architecture

The following recurrent neural network was used for no-limit Texas Hold'em experiments. Two separate recurrent networks with shared parameters were used, consuming observations of each player respectively. Each of those networks consisted of a single linear layer mapping

input representation to a vector of size 256. This was followed by a double rectified linear unit, producing a representation of size 512 then followed by LSTM with 256 hidden units. This produced an information state representation for each player a_0 and a_1.

Define architecture $B(x)$, which will be reused several times. It consumes one of the information state representations produced by the previously mentioned RNN: $h_1 = Linear(128)(x)$, $h_2 = DoubleReLU(h_1)$, $h_3 = h1 + Linear(128)(h2)$, $B(a) = DoubleReLU(h_3)$.

The immediate regret head is formed by applying $B(s)$ on the information state representation followed by a single linear layer of the size of the number of actions in the game. The same is done for an average regret head and mean policy head. All those $B(s)$ do not share weights between themselves, but share weights with respective heads for another player.

The global critic $q(h)$ is defined in the following way. $n_A = Linear(128)$, $n_B = Linear(128)$, $a_0 = n_A(s_0) + n_B(s_1)$, $a_0 = n_B(s_0) + n_A(s_1)$, $h_1 = Concat(a_0, a_1)$, $h_2 = B(h_1)$ and finally $q_0(s_1, s_2)$ and $q_1(s_1, s_2)$ are evaluated by a two linear layers on top of h_2. $B(x)$ shares architecture but does not share parameters with the ones used previously.