Manipulating the Distributions of Experience used for Self-Play Learning in Expert Iteration

Dennis J. N. J. Soemers, Éric Piette, Matthew Stephenson, and Cameron Browne
Department of Data Science and Knowledge Engineering
Maastricht University
Maastricht, the Netherlands
{dennis.soemers,eric.piette,matthew.stephenson,cameron.browne}@maastrichtuniversity.nl

***Abstract*—Expert Iteration (ExIt) is an effective framework for learning game-playing policies from self-play. ExIt involves training a policy to mimic the search behaviour of a tree search algorithm -— such as Monte-Carlo tree search -— and using the trained policy to guide it. The policy and the tree search can then iteratively improve each other, through experience gathered in self-play between instances of the guided tree search algorithm. This paper outlines three different approaches for manipulating the distribution of data collected from self-play, and the procedure that samples batches for learning updates from the collected data. Firstly, samples in batches are weighted based on the durations of the episodes in which they were originally experienced. Secondly, Prioritized Experience Replay is applied within the ExIt framework, to prioritise sampling experience from which we expect to obtain valuable training signals. Thirdly, a trained exploratory policy is used to diversify the trajectories experienced in self-play. This paper summarises the effects of these manipulations on training performance evaluated in fourteen different board games. We find major improvements in early training performance in some games, and minor improvements averaged over fourteen games.**

***Index Terms*—reinforcement learning, self-play, games**

I. Introduction

Over the past few years, many state-of-the-art results in automated learning of policies for game-playing have been obtained by training policies using experience generated from self-play [1]–[4]. In the case of board games, the strongest results to date have been obtained using *Expert Iteration* (ExIt) [1]–[3], which is a self-play training framework in which an expert policy and an apprentice policy iteratively improve each other. The apprentice policy typically takes the form of a parameterised policy that can be trained, such as a neural network that outputs probability distributions over actions for given states. The expert policy is typically a search algorithm, such as *Monte-Carlo tree search* (MCTS) [5]–[7], enhanced to use the apprentice policy to bias its search behaviour. This bias allows the apprentice policy to improve the expert policy. The expert policy subsequently improves the apprentice policy by using the searching behaviour of the expert as a training target for the apprentice policy.

In ExIt, it is customary to generate training experience by running self-play games between instances of the expert policy, where the agents select moves proportionally to the visit counts of the search processes of MCTS. In contrast to greedy move selection, selecting moves proportionally to visit counts increases the diversity of experience that can be used for training. Note that in some cases agents only select moves proportionally to visit counts in the initial portions of games to increase diversity, and switch to greedy selection in the latter parts of training games [1].

There have been numerous attempts at analysing and improving the performance of ExIt-based training procedures [8]–[10]. This includes, for example, modifications to the search behaviour or architecture of the function approximator used for the policy, modification of the loss function, introduction of auxiliary targets, or other changes to the training target, and game-specific improvements (often for the game of Go) [10]. Modifications to the search behaviour – such as introducing different exploration mechanisms in the root node of MCTS – typically lead to changes in the distribution of states that we experience, but they also affect the visit-count-based training targets. However, there has been little investigation of the role played by the distribution of data (game states encountered in self-play) that we generate, or the procedure used to sample from that experience. The most notable exceptions are publications describing state-of-the-art results in various video games [4], [11], which involved extending the notion of self-play learning to use a larger, diverse menagerie [12] of different agents to generate experience.

In the literature on *reinforcement learning* (RL) in standard single-agent settings, *off-policy* RL [13] is a major area of research that allows for trajectories of experience to be generated by a different *behaviour policy* than the *target policy* that we aim to optimise or learn something about. Among other applications, this is commonly used to generate more valuable experience to learn from through directed exploration [14], or to bias the probabilities with which batches of experience are sampled based on how valuable of a training signal they are estimated to provide [15]. Similar applications may turn out to be valuable in the ExIt setting as well.

We explore three different ideas related to the manipulation of either the distribution of data, or how we sample from data, for training in ExIt – without extending the pool of agents that generate experience to a large and diverse set [4], [11]. In all three cases, we use importance sampling (IS) [16], [17] to correct for changes in distributions. First, we use IS in a manner that downweights samples of experience

978-1-7281-4533-4/20/$31.00 ©2020 IEEE

generated in longer episodes during self-play, and upweights samples of experience generated in shorter episodes. Intuitively, this makes every *episode* equally "important" for the training objective, rather than making every *game state* equally important. Second, we explore the application of *Prioritized Experience Replay* [15] in ExIt. Samples of experience that are estimated to provide a valuable training signal are sampled more frequently than they would under uniform sampling, and IS is used to correct for the changed sampling strategy. Third, we train a simple policy to navigate towards game states in which the apprentice policy deviates significantly from the expert policy, and mix this policy with the standard policy that samples moves proportionally to visit counts for the purpose of move selection in self-play. This changes the distribution of data that we expect to see in the experience buffer, and we investigate the use of IS to correct for this change.

An empirical evaluation using fourteen different board games reveals major effects on training performance in individual games – in particular improvements in early stages of training. In later stages of training, there are some games where performance degrades, but the average performance over all games is still improved.

A formalisation of the problem setting, and background information on MCTS and IS, are provided in Section II. Section III explains implementation details of ExIt. Section IV discusses the use of IS in ExIt. Sections V, VI, and VII describe the three proposed extensions. The experimental setup and results are explained in Section VIII, and discussed in Section IX. Finally, Section X concludes the paper.

II. Background

In this section, we formalise the standard framework of Markov decision processes and related concepts used throughout the paper. We use bold symbols – typically lowercase ($\boldsymbol{\pi}$, $\boldsymbol{\theta}$), but sometimes uppercase ($\boldsymbol{\mathcal{M}}$) – to denote vectors.

A. Markov decision processes

Markov decision processes (MDPs) are a standard framework for modelling problems in which an agent perceives and acts in an environment, and is awarded rewards depending on the states it reaches and/or the actions it takes. It is commonly used throughout RL literature [13].

Every MDP consists of a finite set of states $\mathcal{S}$, a finite set of actions $\mathcal{A}$, a transition function $\mathcal{P}$, and a reward function $\mathcal{R}$. At discrete time steps $t = 0, 1, \dots$, the agent observes the current state $S_t \in \mathcal{S}$, selects an action $A_t \in \mathcal{A}$, transitions into a new state S_{t+1}, and observes a reward R_{t+1}. The transition function $\mathcal{P}$ gives the probability $\mathcal{P}(s, a, s') = \Pr\{S_{t+1} = s' \mid S_t = s, A_t = a\}$ for the agent to transition into any new state s' given a previous state $S_t = s$ and a selected action $A_t = a$. Similarly, the reward function $\mathcal{R}$ gives the probability $\mathcal{R}(s, a, s', r) = \Pr\{R_{t+1} = r \mid S_t = s, A_t = a, S_{t+1} = s'\}$ for any arbitrary real number $r \in \mathbb{R}$ to be observed as a reward in that time step.

Because it simplifies notation, we assume that every episode starts in the same initial state $S_0 = s_0$, but all the theory can trivially be extended to the case where the initial state is sampled from some fixed distribution. We are primarily interested in domains with episodes of finite length, but use sums $\sum_{t=0}^{\infty}$ over infinite numbers of time steps throughout most of the paper – which covers infinite-duration episodes. Finite-duration episodes, of length T, are still covered by setting all rewards $R_{t+1>T}$ after T time steps passed to 0.

A policy π is a function that, given a state s and an action a, produces a probability $0 \leq \pi(s, a) \leq 1$ for the policy to choose to execute a in s. Note that we require policies to yield probability distributions over all actions; $\sum_{a \in \mathcal{A}} \pi(s, a) = 1 \quad \forall s \in \mathcal{S}$. We use $\boldsymbol{\pi}(s)$ to denote a vector of probabilities for all possible entries $a \in \mathcal{A}$. We assume that all policies automatically set probabilities of any illegal actions to 0.

The *value* of a state s under a policy π, denoting the (discounted) cumulative rewards that we expect to obtain when sampling actions from π after reaching s, is given by;

$$V^{\pi}(s) \doteq \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R_{t+1} \mid A_{t' \leq t} \sim \pi\right], \tag{1}$$

where $0 \leq \gamma \leq 1$ denotes a discount factor. In infinitely long episodes, we require $\gamma < 1$ to guarantee that all states have finite value. In the practical implementations and experiments described in this paper, we only use finite-duration episodes and simply take $\gamma = 1$. The notation $A_{t' \leq t}$ denotes that all actions $A_{t'}$ for $t' \leq t$ are sampled from π. Note that this expectation, and various other expectations throughout the paper, formally also depend on the choice of initial state $s_0 = S_0$. This dependence is left implicit for notational brevity.

When applying this framework to multi-player, adversarial games, we generally do so from the "perspective" of a single player at a time, which is oblivious to the presence of other agents and simply treats them as a part of the "environment". This means that states in which other players are to move are skipped over, and the influence of other agents on the probabilities with which we reach states (through their policies) is merged with the environment's transition dynamics $\mathcal{P}$.

B. Monte-Carlo tree search

Monte-Carlo tree search (MCTS) [5]–[7] is a tree search algorithm that gradually builds up (typically in an asymmetric fashion) its search tree over multiple iterations. During every iteration, MCTS traverses the tree that has been built up so far, using a *selection* strategy that balances *exploitation* of parts of the search tree that appear promising so far, and *exploration* of parts of the search tree that have not yet been sufficiently explored in previous iterations. The search tree is typically expanded by a single node in the area reached by this selection strategy. A fast, (semi-)random *play-out* strategy is typically used to roll out all the way to a terminal game state, which then yields a (highly noisy) estimate of the value of all states traversed in the current iteration. This value is backpropagated through the tree, and may be used to inform the selection strategy in subsequent iterations. The number of iterations that traversed through any given node during the search process is referred to as the *visit count* of that node. Note that MCTS is not restricted to the MDP framework, and can account for the actions that other agents with opposing interests may take.

C. Importance sampling

Importance sampling (IS) [16], [17] is a standard technique to correct for differences between two distributions when using samples from one distribution to estimate expectations from another distribution. Suppose that we collect a set of n samples $\{x_k \mid 1 \leq k \leq n\}$ from a distribution μ, and wish to estimate the expected value $\mathbb{E}_{\pi}\left[x\right]$ under a different distribution π. Let $\mu(x_k)$ denote the probability of observing x_k under μ, and $\pi(x_k)$ the probability of observing x_k under π. Then, the *importance sampling ratios* $\rho_k = \frac{\pi(x_k)}{\mu(x_k)}$ can be used to compute an estimator $\hat{x}$ for $\mathbb{E}_{\pi}\left[x\right]$:

$$\hat{x} = \frac{\sum_{k=1}^{n} \rho_k x_k}{n} \approx \mathbb{E}_{\pi}\left[x\right]. \tag{2}$$

This estimator is unbiased, but often exhibits high variance. This becomes particularly problematic in off-policy RL applications [18], [19], where sequences of multiple IS ratios – correcting for differences between policies across sequences of multiple time steps – are often all multiplied together. An alternative to this estimator, referred to as the *weighted importance sampling* (WIS) estimator, is given by:

$$\hat{x} = \frac{\sum_{k=1}^{n} \rho_k x_k}{\sum_{k=1}^{n} \rho_k}. \tag{3}$$

Estimators of this form are not unbiased, but have substantially lower variance and are often found to perform better in practice – also in off-policy RL applications [18], [20].

III. Expert Iteration

Expert Iteration (ExIt) [1], [2] is the self-play training framework for which an intuitive description was provided in Section I. This section provides a few implementation details that are particularly important for the remainder of this paper.

We aim to train a parameterised policy $\pi_{\boldsymbol{\theta}}$, with parameters $\boldsymbol{\theta}$. These are often the parameters of a deep neural network [1]–[3], [8]–[10], but in the empirical evaluation in this paper we focus on simpler linear function approximators. This makes it computationally feasible to perform our evaluations in *general game playing* settings, using a wide variety of games as test domains. The theoretical aspects of this paper are written to facilitate either form of function approximation. Let $\boldsymbol{\phi}(s, a)$ denote a feature vector for the state-action pair (s, a). For every such pair, in any given game state s, we compute a logit $z(s, a) = \boldsymbol{\theta}^{\top} \boldsymbol{\phi}(s, a)$. The policy's probabilities $\pi_{\boldsymbol{\theta}}(s, a)$ are then given by a softmax over all the action logits:

$$\pi_{\boldsymbol{\theta}}(s, a) = \frac{\exp z(s, a)}{\sum_{a' \in \mathcal{A}} \exp z(s, a')}. \tag{4}$$

Experience is generated by playing games of self-play between identical MCTS agents, which use $\pi_{\boldsymbol{\theta}}$ to guide their search. We use the same selection strategy as AlphaGo Zero [1], which traverses the tree by traversing edges that correspond to actions a_{PUCT} selected according to:

$$a_{PUCT} = \operatorname*{argmax}_{a} \hat{Q}(s, a) + C_{PUCT} \frac{\pi_{\boldsymbol{\theta}}(s, a) \sqrt{\sum_{a'} N(s, a')}}{1 + N(s, a)}, \tag{5}$$

where s denotes the state of the current node, $\hat{Q}(s, a)$ denotes the current value estimate of executing a in s as estimated by the MCTS process so far, and $N(s, a)$ denotes the visit count of the edge that is traversed by executing a in s. Contrary to most related work with ExIt, we do not use a state-value function approximator, and only backpropagate values resulting from playouts executed using $\pi_{\boldsymbol{\theta}}$. This eliminates the need for learning a strong state-value function.

In the self-play games, agents select moves proportional to the visit counts along edges from the root node after executing an MCTS search process for a fixed number of iterations. Suppose that we built up a search tree by running MCTS from a root node with a root state s. Then, we can formally define a policy $\mathcal{M}_s$, that assigns probabilities $\mathcal{M}_s(s, a)$ as follows:

$$\mathcal{M}_s(s, a) = \frac{N(s, a)}{\sum_{a'} N(s, a')}, \tag{6}$$

where $N(s, a)$ denotes the final visit counts after searching.

Experience in self-play is generated by, for every encountered state s, running an MCTS process rooted in s, and selecting an action by sampling from $\boldsymbol{\mathcal{M}}_s(s)$. A tuple containing s, $\boldsymbol{\mathcal{M}}_s(s)$, and any other data required for training, is stored in a limited-capacity experience buffer that discards the oldest entries first when the maximum capacity is reached.

Training is typically done by uniformly sampling batches of experience tuples with states s from the buffer, and taking gradient descent steps to minimise the cross-entropy between apprentice policy $\boldsymbol{\pi}_{\boldsymbol{\theta}}(s)$ and expert policy $\boldsymbol{\mathcal{M}}_s(s)$. The loss, estimated by averaging over a batch of size N, is given by:

$$\mathcal{L}_{CE} \approx \frac{1}{N} \sum_{i=1}^{N} \boldsymbol{\mathcal{M}}_{s_i}(s_i)^{\top} \log \boldsymbol{\pi}_{\boldsymbol{\theta}}(s_i). \tag{7}$$

It is common to include an L_2 regularisation term [1], [3], but this is omitted in this paper, as our use of relatively simple function approximators and significantly lower training times reduces the risk of overfitting.

IV. Importance Sampling in ExIt

Suppose that an experience buffer is filled with tuples of experience corresponding to all states s_i encountered in self-play, as described above. If the MCTS agent used to generate experience remains fixed, the weightings $d^{\mathcal{M}}(s)$ with which we expect to observe states s in the buffer is then given by:

$$d^{\mathcal{M}}(s) \doteq \sum_{t=0}^{\infty} \Pr\{S_t = s \mid A_{t'<t} \sim \mathcal{M}_{S_{t'}}\}. \tag{8}$$

The standard approach of sampling batches to estimate the gradients for gradient descent updates uniformly from this buffer then yields an expected probability of $p(s) = \frac{d^{\mathcal{M}}(s)}{\sum_{s'} d^{\mathcal{M}}(s')}$ for a tuple containing any particular state s to be sampled. Note that the assumption that the MCTS agent used to generate experience remains fixed is a simplifying assumption. In practice, the agent's behaviour is gradually modified by updating the apprentice policy $\pi_{\boldsymbol{\theta}}$, while retaining old experience generated using older versions of the policy in the experience buffer until they are discarded due to the limited capacity of the buffer.

Sampling states according to these probabilities $p(s)$ implies that, in expectation, the cross-entropy loss that we estimate using Equation (7) – and therefore optimise – is given by:

$$\mathcal{L}_{CE} = \sum_{s \in \mathcal{S}} p(s) \left(\boldsymbol{\mathcal{M}}_s(s)^\top \log \boldsymbol{\pi}_{\boldsymbol{\theta}}(s) \right). \tag{9}$$

A. Optimising for a different data distribution

Generating data (experience) as described above is the most common procedure, and has produced state-of-the-art results empirically [1], [3], but it is not certain that the optimal loss function is one that weights states s by $p(s)$ as in Equation (9). It is possible that different weightings may perform better. If we have *target probabilities* $\omega(s)$ that we expect to work better than $p(s)$ in Equation (9), we may use IS ratios $\rho(s) = \frac{\omega(s)}{p(s)}$ (as described in Subsection II-C) to estimate appropriate gradients – without requiring a change in how ExIt generates experience.

B. Optimising with a different data distribution

Even if we expect the cross-entropy loss function in Equation (9), where states s are weighted by $p(s)$, to be the optimal one to optimise. It is still possible that approaches leading to experience buffers with different data distributions, or approaches that sample from it in a different (non-uniform) manner, may be expected to perform more successfully. By using $\mu(s)$ to denote the new probability for any state s to be sampled due to a modified data-generating or sampling procedure, we can specify IS ratios $\rho(s) = \frac{p(s)}{\mu(s)}$ to estimate appropriate gradients for the optimisation of Equation (9). This holds even if ExIt has been modified to store (or sample from) experience in a different way.

V. Weighting According to Episode Durations

One of the original publications on ExIt [2] describes only storing *a single state* s in the experience buffer *for every full episode* experienced in self-play. The primary motivation for this was to break correlations in the data, because states that occurred in the same episode may be highly correlated. For a similar reason, the value network of AlphaGo [21] was trained from data containing only one state per game of self-play. In contrast, AlphaGo Zero [1] and AlphaZero [3] were trained using buffers that contained all states observed in self-play. Presumably, the improvements in sample efficiency were found to outweigh possible detriments due to correlated data.

Aside from the observation that storing only a single state per episode breaks correlations, it also has a different effect on the data distribution; it ensures that every episode is represented "equally" by a single state. When storing all states, longer-duration episodes may be argued to be "over-represented" due to having more states. When storing all states in experience buffers, and therefore preserving sample efficiency, we can treat the data distribution where every episode – regardless of duration – is equally represented as target distribution, and use IS ratios to correct for the potential issue of overrepresentation of states from long episodes.

Let T denote the duration of one particular episode. If we were to only include a single state from this episode in the experience buffer, the probability for any particular state s to be selected would be $\frac{1}{T}$. Recall that $d^{\mathcal{M}}(s)$ denotes the relative weightings with which we expect to observe states s when storing every state per episode, leading to probabilities $p(s)$ after dividing by $\sum_{s'} d^{\mathcal{M}}(s')$ for normalisation. The relative weightings $d^{\mathcal{M}}_{single}(s)$ with which states would be observed if we only stored a single state per episode are given by $d^{\mathcal{M}}_{single}(s) = \frac{1}{\mathbb{E}[T \mid s \text{ observed}]} d^{\mathcal{M}}(s)$, where $\mathbb{E}\left[T \mid s \text{ observed}\right]$ denotes the expected duration of episodes in which s is observed. Normalising to probabilities leads to the following target probabilities $\omega(s)$:

$$\begin{aligned} \omega(s) &= \frac{\mathbb{T}}{\mathbb{E}\left[T \mid s \text{ observed}\right]} \frac{d^{\mathcal{M}}(s)}{\sum_{s'} d^{\mathcal{M}}(s')} \\ &= \frac{\mathbb{T}}{\mathbb{E}\left[T \mid s \text{ observed}\right]} p(s), \end{aligned} \tag{10}$$

where $\mathbb{T}$ denotes the expected duration of any episode in ExIt.

As described in Subsection IV-A, this means that we can use IS ratios $\rho(s)$ given by:

$$\begin{aligned} \rho(s) = \frac{\omega(s)}{p(s)} &= \frac{\mathbb{T}}{\mathbb{E}\left[T \mid s \text{ observed}\right]} p(s) \frac{1}{p(s)} \\ &= \frac{\mathbb{T}}{\mathbb{E}\left[T \mid s \text{ observed}\right]}. \end{aligned} \tag{11}$$

In practice, the empirical duration T of the episode in which any particular state s was observed can be stored in the experience buffer along with s, and used as an unbiased estimator of $\mathbb{E}\left[T \mid s \text{ observed}\right]$. We keep track of a moving average $\hat{\mathbb{T}}$ of episode durations during self-play as an estimator for $\mathbb{T}$. Recent episodes are given a higher weight than old episodes in this moving average, because our MCTS agent is not stationary in practice due to its use of the apprentice policy (which is trained over time). More concretely, after completing the i^{th} episode with a duration T_i, we update $\hat{\mathbb{T}}$ as follows:

$$\begin{aligned} u_i &\leftarrow 0.95 u_{i-1} + 1 \qquad (u_0 \doteq 0) \\ \hat{\mathbb{T}} &\leftarrow \hat{\mathbb{T}} + \frac{1}{u_i}(T_i - \hat{\mathbb{T}}). \end{aligned} \tag{12}$$

VI. Prioritized Experience Replay

Prioritized Experience Replay (PER) [15] is an approach that samples batches of experience in a non-uniform manner. Elements from a larger replay buffer are sampled more frequently if they are expected to perform a valuable training signal, and less frequently if a trained model already appears to provide accurate predictions for them. It is commonly used in value-based RL approaches, where it has been found to be one of the most valuable extensions [22] for DQN [23].

In PER, tuples of experience i in a replay (or experience) buffer are assigned *priority levels* p_i. When sampling batches from the buffer for training, tuples i are sampled with probability $P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$. The exponent α is a hyperparameter, where $\alpha = 0$ leads to uniform sampling, and $\alpha > 0$ causes tuples with higher priority levels to be sampled more frequently. Sampling according to these probabilities can be implemented efficiently using a binary tree structure [15].

When applied to value-based RL, priority levels are typically assigned based on the absolute values of the *temporal difference* errors, which may intuitively be interpreted as the magnitudes of the mistakes made by a value function approximator for given tuples of experience. For the optimisation of the cross entropy loss (Equation (9)) considered in this paper, we similarly assign priorities based on the differences between apprentice and expert distributions.

Let s_i denote a state that occurs in our experience buffer, with an expert distribution $\boldsymbol{\mathcal{M}}_s(s)$ over all actions, and an apprentice distribution $\boldsymbol{\pi_\theta}(s)$. As in the original PER implementation [15], the priority level is simply set equal to the maximum priority level across all existing tuples of experience if s_i is newly entered (i.e. if we have not yet used it for even a single update). After using s_i for an update, its new priority level p_i is set by summing up the absolute differences between the distributions for all actions:

$$p_i = \sum_{a \in \mathcal{A}} \left| \mathcal{M}_{s_i}(s_i, a) - \pi_{\boldsymbol{\theta}}(s_i, a) \right|. \tag{13}$$

We also considered using only the maximum absolute error, rather than the sum, or simply using the cross-entropy loss $\boldsymbol{\mathcal{M}}_s(s)^\top \log \boldsymbol{\pi_\theta}(s)$ as a priority level. We decided against using the maximum absolute error, because that tends to be a (decreasing) function of the number of legal actions in a state, more so than an indication of how well a policy performs. The cross-entropy loss was not used because its absolute value may be arbitrarily large, which can lead to instability.

As in the original PER [15], we compute IS ratios $\rho(s_i)$ for sampled states s_i using:

$$\rho(s_i) = \left(\frac{1}{N} \frac{1}{P(i)} \right)^\beta, \tag{14}$$

where N is the total number of tuples in the experience buffer. The exponent β is a hyperparameter, where $\beta = 0$ leads to no corrections for bias, and $\beta = 1$ fully corrects for the changes in sampling probabilities as described in Subsection IV-B. For improved stability, we also divide all IS ratios $\rho(s_i)$ in any batch by the maximum IS ratio across that batch [15].

Note that the original PER publication [15] describes multiplying the IS ratios with the temporal-difference errors in Q-learning updates, which yields WIS estimators [20]. In the case of the cross-entropy losses considered in this paper, we multiply the IS ratios with the full cross-entropy loss. Obtaining a WIS estimator still requires explicitly constructing an estimator of the form in Equation (3).

VII. Cross-entropy Exploration

The intuition behind PER is that states s for which the apprentice policy's distribution $\boldsymbol{\pi_\theta}(s)$ does not yet approximate the expert policy's distribution $\boldsymbol{\mathcal{M}}_s(s)$ may be especially valuable to learn from. This intuition does not only have to apply to the stage where we sample collected experience from a buffer, but may also inform how we should collect experience in the first place. It may be beneficial for learning to actively seek out states in self-play that lead to large differences between the two policies. We refer to this idea as Cross-Entropy Exploration (CEE).

More concretely, we train an additional policy μ using REINFORCE [24]. At every time step t in an episode, μ obtains the sum of absolute differences between probabilities assigned to all actions by the expert and apprentice as a reward:

$$R_{t+1} = \sum_{a \in \mathcal{A}} \left| \mathcal{M}_{S_t}(S_t, a) - \pi_{\boldsymbol{\theta}}(S_t, a) \right|. \tag{15}$$

This means that μ is trained to navigate towards states that (eventually) lead to large errors for the apprentice distribution. Note that – unlike typical rewards used in games such as "wins" or "losses" – these rewards are invariant to the state's current mover. This means that we can collect rewards from *all* encountered states, rather than only from those corresponding to a specific player. This policy is trained using a discount factor $\gamma = 0.99$.

In self-play, we no longer sample actions proportionally to the visit counts of MCTS, but we sample actions from a mixed distribution with action-probabilities $0.9\mathcal{M}_s(s, a) + 0.1\mu(s, a)$. A correction for the modified probabilities for a single step requires an IS ratio $\rho(S_t) = \frac{\mathcal{M}_{S_t}(S_t, A_t)}{0.9\mathcal{M}_{S_t}(S_t, A_t) + 0.1\mu(S_t, A_t)}$. As in multi-step off-policy RL settings [18], [19], longer trajectories of multiple time steps with a modified behaviour policy require a product of many such IS ratios. For improved stability – and to avoid cases where large portions of entire episodes become completely useless when $\mathcal{M}_{S_t}(S_t, A_t) = 0$ but $\mu(S_t, A_t) > 0$ – we truncate these (products of) IS ratios to always lie in $[0.1, 2]$. This comes at the cost of some bias.

VIII. Experiments

This section describes experiments used to empirically evaluate the effects of weighting states according to episode durations (WED), Prioritized Experience Replay (PER), and Cross-Entropy Exploration (CEE) on the performance of agents with policies trained using ExIt.

A. Setup

We use fourteen different board games, implemented in the Ludii general game system [25]; Amazons, Ard Ri, Breakthrough, English Draughts, Fanorona, Gomoku, Hex, Knightthrough, Konane, Pentalath, Reversi, Surakarta, Tablut, and Yavalath. These are all two-player, deterministic, perfect information board games, but otherwise varied in mechanics and goals. Ard Ri and Tablut are highly asymmetric games.

For each of WED, PER, and CEE, we run a training run of ExIt for 200 games of self-play. We also include a standard ExIt run (without any of the extensions discussed in this paper), an additional run of CEE without performing any IS corrections, and a training run that uses WED, PER, and CEE (without IS) simultaneously. Policies use local patterns [26] as binary features for state-action pairs. We start every training run with a limited set of "atomic" features, and add one feature to every feature set after every full game of self-play [27]. Because we include asymmetric games, we use separate feature sets, separate experience buffers, and train separate feature weights, per player number (or colour). Experience buffers have a maximum capacity of 2500 states. Policies are trained by taking a gradient descent step at the end of every time step in self-play, using a centred variant [28] of RMSProp as optimiser, and batches of 30 states to estimate gradients.

PER uses $\alpha = \beta = 0.5$ for its hyperparameters. These are the default values for PER in the Dopamine framework [29]. In all cases where IS is used for WED, PER, or CEE, we use WIS estimators of the form in Equation (3) to estimate gradients. The unbiased, higher-variance ordinary IS estimators were found not to perform as well in preliminary experiments.

For every training run, we store checkpoints of feature sets and trained weights after 1, 51, 101, 151, and 200 games of self-play, leading to five different versions of each of the following: **ExIt** (no extensions), **WED**, **PER**, **CEE**, **CEE (No IS)**, and **WED + PER + CEE (No IS)**, for a total of 30 trained agents. In evaluation games, we also add two more non-learning agents as benchmarks: **UCT** (a standard UCT [7] implementation), and **MC-GRAVE** (an implementation of GRAVE [30] without exploration term in the selection phase), for a total of 32 agents participating in evaluation games.

UCT uses a value of $\sqrt{2}$ for its exploration constant. All of the trained agents use $C_{PUCT} = 2.5$ in Equation (5). All variants of MCTS re-use relevant parts of search trees from previous searches, and run 800 iterations per move – in training as well as evaluation games. The use of 800 iterations is consistent with AlphaZero [3]. Value estimates in all variants of MCTS lie in $[-1, 1]$. Unvisited nodes are always estimated to have a value equal to the value estimate of their parent, except in MC-GRAVE where unvisited nodes get a value estimate of $10,000$. In evaluation games, all agents select the action that maximises the visit count (breaking ties randomly).

For every game, we run 120 evaluation matches for every possible (unordered) pair of agents that could be sampled – with replacement – from the total pool of 32 agents. Every agent plays each side of its matchup in half of the evaluation games (i.e. 60 out of 120).

B. Results

The thick lines in Fig. 1 depict the average win percentages of each of the 30 different (checkpoints of) learning agents across all games against all 31 possible opponents. Different checkpoints of the same training run are connected, forming learning curves. The two non-learning agents (UCT and MC-GRAVE) are drawn as horizontal lines. The fourteen thin lines depict similar learning curves for WED + PER + CEE (No IS) for individual games (i.e., not averaged over all games), and only use ExIt at equal training checkpoints as opponent (i.e., not averaged over all opponents).

While these win percentages offer some insight into relative playing strengths, a shortcoming of this metric is that every possible opponent is considered equally important. Suppose that there are three agents A, B, and C. If A outperforms the other two by a small margin, we may consider it to be the strongest agent. But if B (outperformed by A) more aggressively exploits the weakest agent C, it may be ranked as the top agent by average win percentages. Therefore, we also evaluate our agents using α-rank [31], [32]. This ranking approach, based on evolutionary game theory, would find that agent A is dominated and would be eliminated from the population of agents in the example above.

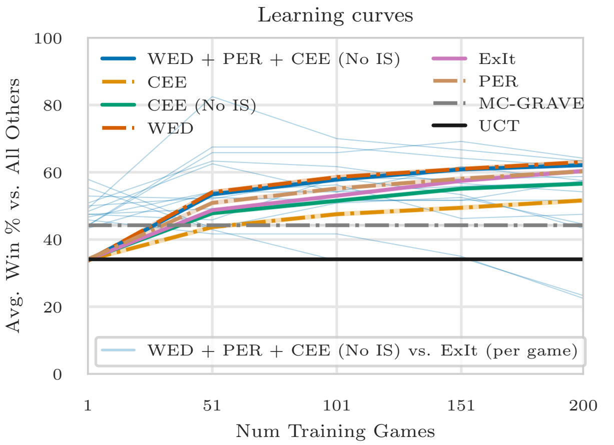

Fig. 1. Thick lines depict progression of win percentages, averaged over all fourteen games and all 31 possible opponent agents, including 95% Agresti-Coull confidence intervals for the mean over all those data points. The fourteen thin lines depict win percentages in individual games of **WED + PER + CEE (No IS)**, against only **ExIt** with equal numbers of training games.

We use tables of pairwise win rates as payoff tables for α-rank, conducting a sweep over its ranking-intensity hyperparameter α to find sufficiently high values [31] for every game. We treat all games as asymmetric games, meaning that α-rank does not generate rankings of agents, but rankings of *pairs* of agents corresponding to the two player indices in 2-player games. In some games the same agent is the top-performing agent for both player numbers, but there are also cases where one agent performs best as Player 1 and another as Player 2.

Table I shows the results of the α-rank evaluations. For every agent, we count how often it is present in the top-ranked strategy across all games. There is a total of 28 top ranks available across fourteen games. For every agent, we also compute the strategy mass of that agent in α-rank's stationary distribution over strategies – averaged over the fourteen games. These two metrics are often correlated, but can still provide different insights. When a single agent clearly outperforms all the others, it achieves the top ranks as well as gaining all the strategy mass in a game. When multiple closely-matched agents outperform each other (e.g., pure strategies in Rock-Paper-Scissors), the strategy mass is more evenly distributed among these agents.

For the trained agents, we add up the top ranks and strategy masses for all the different checkpoints of the same training

TABLE I

RESULTS FOR α-RANK EVALUATIONS IN FOURTEEN GAMES. THE SECOND COLUMN SHOWS THE NUMBER OF TOP RANKS THAT EVERY AGENT HAS OBTAINED ACROSS THE GAMES (WITH 2 TOP RANKS AVAILABLE PER GAME, FOR THE 2 PLAYERS). THE THIRD COLUMN SHOWS THE STRATEGY MASSES OF THE DIFFERENT AGENTS IN THE STATIONARY DISTRIBUTION OF α-RANK (AVERAGED OVER GAMES).

Agent	Num. Top Ranks	Avg. Strategy Mass
UCT	0	0.010
MC-GRAVE	4	0.145
ExIt	3	0.085
WED	9	0.304
PER	2	0.118
CEE	1	0.048
CEE (No IS)	4	0.146
WED + PER + CEE (No IS)	5	0.144
Total	28	1.0

run. There were only few cases where the final checkpoints were not definitively the strongest agents of their run.

IX. DISCUSSION

In Fig. 1, we see WED and the combination of extensions WED + PER + CEE (No IS) outperforming the ExIt baseline on average, especially for the early checkpoints of 51 and 101 training games, but also in later checkpoints to a lesser extent. PER on its own also has a small positive impact in the initial stages of learning. Both variants of CEE are detrimental for performance on average, with the variant that uses IS corrections performing significantly worse than the variant that ignores IS corrections.

The thin learning curves in Fig. 1 show that the combination of extensions leads to major improvements in playing strength in the early stages of training in multiple games, with win percentages between 60% and 85% against ExIt with the same amount of training in five out of fourteen games after 51 training episodes. For other games, the playing strength tends to be closer to even. After 200 training episodes, there are two games where the regular ExIt has a major advantage in playing strength, but on average the extensions still lead to a minor advantage. For other extensions, we similarly observed that there can be major effects – both positive and negative – in individual games, but we omit these plots for visual clarity.

The α-rank evaluations in Table I show particularly dominant results for WED, in terms of its number of achieved top ranks as well as average presence in the stationary distributions over agents. This is interesting considering it is also the simplest of all the evaluated extensions of ExIt. PER achieves only two top ranks, but has a high average strategy mass relative to this number of top ranks. This suggests that PER has a relatively stable level of performance; it rarely leads to the best agent, but it is also rarely entirely dominated by other strategies. In contrast, ExIt without any extensions has a relatively low average strategy mass.

X. CONCLUSION

This paper explores three different extensions for the Expert Iteration (ExIt) self-play training framework, all three of which involve manipulations of the distribution of data that we learn from – either by modifying the distribution of data that we collect, or by modifying how we sample from it.

Firstly, we investigated applying importance sampling (IS) corrections based on the durations of episodes in which samples of experience were observed, such that – in expectation – we optimise the cross-entropy loss for the distribution of states that we would have collected if we only stored one state for every full game of self-play. We still retain sample efficiency because we do in practice retain *all* states – IS corrects for this discrepancy between the distribution of collected data, and distribution of data for which we optimise. This is referred to as weighting according to episodes durations (WED).

Secondly, we apply Prioritized Experience Replay (PER) [15] to the ExIt training framework. The impact that experienced states may have on our training process is estimated by the differences between expert and apprentice policies for these states, and states that are estimated to be more informative are sampled more frequently. IS ratios are used to correct for bias introduced by this non-uniform sampling.

Thirdly, we use REINFORCE [24] to train an additional exploratory policy that is rewarded for navigating to states in which there is a large mismatch between expert and apprentice policies. This exploratory policy is mixed with the standard visit-count-based policy when selecting actions during self-play training. This is referred to as Cross-Entropy Exploration (CEE). We evaluate the introduction of this exploration mechanism both with and without applying IS corrections to correct for the modified distribution of experienced states.

An empirical evaluation across fourteen different 2-player games shows that – on average – WED, and a combination of WED + PER + CEE (No IS), lead to policies with stronger performance levels in terms of average win percentage against a pool of 31 other agents. This difference is primarily noticeable in the early stages of training. This pool of other agents includes earlier and later checkpoints of the same training run, all checkpoints of all other training runs, and two non-training agents (UCT and MC-GRAVE). PER on its own also appears to have a minor advantage in early training stages. Either variant of CEE on its own appears to be detrimental.

An additional evaluation using the α-rank [31] method from evolutionary game theory provides additional evidence for some of these conclusions. The α-rank evaluation is particularly favourable for WED, but also for other extensions proposed in the paper.

From these results, we conclude that it is worth examining the distributions of experience for which we optimise cross-entropy losses in self-play training processes such as ExIt more closely. Various extensions that maniupulate these distributions show improvements in playing strength when averaged over fourteen games. WED, which is arguably the simplest modification examined in this paper, also appears to have one of the most noticeable impacts on training performance. Effects averaged over all games tend to be small, but we observe major effects in individual games.

For CEE, in this paper we focused on training a policy to explore trajectories that leads to large cross-entropy losses. In future work, it would also be interesting to investigate other forms of targeted exploration [14]. For example, a policy that has already been trained in one game may be directly used to diversify the experience collected – and speed up learning – in a second game [33]. Finally, it would be interesting to investigate if there are certain patterns to which extensions provide positive or negative effects in which games.

Acknowledgment.

This research was conducted as part of the European Research Council-funded Digital Ludeme Project (ERC Consolidator Grant #771292), run by Cameron Browne at Maastricht University's Department of Data Science and Knowledge Engineering (DKE). We thank Shayegan Omidshafiei for guidance on the α-rank evaluations.

References

[1] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis, "Mastering the game of Go without human knowledge," *Nature*, vol. 550, pp. 354–359, 2017.

[2] T. Anthony, Z. Tian, and D. Barber, "Thinking fast and slow with deep learning and tree search," in *Adv. in Neural Inf. Process. Syst. 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 5360–5370.

[3] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis, "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play," *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018.

[4] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, J. Oh, D. Horgan, M. Kroiss, I. Danihelka, A. Huang, L. Sifre, T. Cai, J. P. Agapiou, M. Jaderberg, A. S. Vezhnevets, R. Leblond, T. Pohlen, V. Dalibard, D. Budden, Y. Sulsky, J. Molloy, T. L. Paine, C. Gulcehre, Z. Wang, T. Pfaff, Y. Wu, R. Ring, D. Yogatama, D. Wünsch, K. McKinney, O. Smith, T. Schaul, T. Lillicrap, K. Kavukcuoglu, D. Hassabis, C. Apps, and D. Silver, "Grandmaster level in StarCraft II using multi-agent reinforcement learning," *Nature*, vol. 575, pp. 350–354, 2019.

[5] L. Kocsis and C. Szepesvári, "Bandit based Monte-Carlo planning," in *Mach. Learn.: ECML 2006*, ser. LNCS, J. Fürnkranz, T. Scheffer, and M. Spiliopoulou, Eds. Springer, Berlin, Heidelberg, 2006, vol. 4212, pp. 282–293.

[6] R. Coulom, "Efficient selectivity and backup operators in Monte-Carlo tree search," in *Computers and Games*, ser. LNCS, H. J. van den Herik, P. Ciancarini, and H. H. L. M. Donkers, Eds., vol. 4630. Springer Berlin Heidelberg, 2007, pp. 72–83.

[7] C. Browne, E. Powley, D. Whitehouse, S. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A survey of Monte Carlo tree search methods," *IEEE Trans. Comput. Intell. AI Games*, vol. 4, no. 1, pp. 1–49, 2012.

[8] Y. Tian, J. Ma, Q. Gong, S. Sengupta, Z. Chen, J. Pinkerton, and C. L. Zitnick, "ELF OpenGo: An analysis and open reimplementation of AlphaZero," in *Proc. 36th Int. Conf. Mach. Learn. (ICML)*, 2019, pp. 6244–6253.

[9] F. Morandin, G. Amato, R. Gini, C. Metta, M. Parton, and G.-C. Pascutto, "SAI: a sensible artificial intelligence that plays Go," in *Proc. 2019 Int. Joint Conf. Neural Networks (IJCNN)*. IEEE, 2019.

[10] D. J. Wu, "Accelerating self-play learning in Go," https://arxiv.org/abs/1902.10565v3, 2019.

[11] M. Jaderberg, W. M. Czarnecki, I. Dunning, L. Marris, G. Lever, A. G. Castañeda, C. Beattie, N. C. Rabinowitz, A. S. Morcos, A. Ruderman, N. Sonnerat, T. Green, L. Deason, J. Z. Leibo, D. Silver, D. Hassabis, K. Kavukcuoglu, and T. Graepel, "Human-level performance in 3D multiplayer games with population-based reinforcement learning," *Science*, vol. 364, no. 6443, pp. 859–865, 2019.

[12] D. Hernandez, K. Denamganaï, Y. Gao, P. York, S. Devlin, S. Samothrakis, and J. A. Walker, "A generalized framework for self-play training," in *IEEE Conf. on Games (CG)*. IEEE, 2019, pp. 586–593.

[13] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. Cambridge, MA: MIT Press, 2018.

[14] S. B. Thrun, *The role of exploration in learning control*. New York, NY: Van Nostrand Reinhold, 1992.

[15] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," in *Int. Conf. Learning Representations (ICLR)*, 2016.

[16] H. Kahn and A. W. Marshall, "Methods of reducing sample size in Monte Carlo computations," *Journal of the Operations Research Society of America*, vol. 1, no. 5, pp. 263–278, 1953.

[17] R. Y. Rubinstein, *Simulation and the Monte Carlo Method*. New York: Wiley, 1981.

[18] D. Precup, R. S. Sutton, and S. Singh, "Eligibility traces for off-policy policy evaluation," in *Proc. 17th Int. Conf. Mach. Learn. (ICML)*. Morgan Kaufmann, 2000, pp. 759–766.

[19] D. Precup, R. S. Sutton, and S. Dasgupta, "Off-policy temporal-difference learning with function approximation," in *Proc. 18th Int. Conf. Mach. Learn. (ICML)*. Morgan Kaufmann, 2001, pp. 417–424.

[20] A. R. Mahmood, H. van Hasselt, and R. S. Sutton, "Weighted importance sampling for off-policy learning with linear function approximation," in *Adv. in Neural Inf. Process. Syst. 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds., 2014.

[21] D. Silver, A. Huang, C. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[22] M. Hessel, J. Modayil, H. van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver, "Rainbow: Combining improvements in deep reinforcement learning," in *Proc. AAAI Conf. Artif. Intell. (AAAI)*. AAAI, 2018, pp. 3215–3222.

[23] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, 2015.

[24] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Mach. Learn.*, vol. 8, no. 3-4, pp. 229–256, 1992.

[25] É. Piette, D. J. N. J. Soemers, M. Stephenson, C. F. Sironi, M. H. M. Winands, and C. Browne, "Ludii - the ludemic general game system," in *Proc. 2020 Eur. Conf. Artif. Intell.*, 2020, to appear.

[26] C. Browne, D. J. N. J. Soemers, and E. Piette, "Strategic features for general games," in *Proc. 2nd Workshop Know. Extraction from Games (KEG)*, 2019, pp. 70–75.

[27] D. J. N. J. Soemers, É. Piette, and C. Browne, "Biasing MCTS with features for general games," in *Proc. 2019 IEEE Congr. Evol. Computation*. IEEE, 2019, pp. 442–449.

[28] A. Graves, "Generating sequences with recurrent neural networks," https://arxiv.org/abs/1308.0850v5, 2013.

[29] P. S. Castro, S. Moitra, C. Gelada, S. Kumar, and M. G. Bellemare, "Dopamine: A research framework for deep reinforcement learning," https://arxiv.org/abs/1812.06110, 2018.

[30] T. Cazenave, "Generalized rapid action value estimation," in *Proc. 24th Int. Joint Conf. Artif. Intell. (IJCAI)*, Q. Yang and M. Woolridge, Eds. AAAI Press, 2015, pp. 754–760.

[31] S. Omidshafiei, C. Papadimitriou, G. Piliouras, K. Tuyls, M. Rowland, J.-B. Lespiau, W. M. Czarnecki, M. Lanctot, J. Perolat, and R. Munos, "α-rank: Multi-agent evaluation by evolution," *Scientific Reports*, vol. 9, no. 9937, 2019.

[32] M. Lanctot, E. Lockhart, J.-B. Lespiau, V. Zambaldi, S. Upadhyay, J. Pérolat, S. Srinivasan, F. Timbers, K. Tuyls, S. Omidshafiei, D. Hennes, D. Morrill, P. Muller, T. Ewalds, R. Faulkner, J. Kramár, B. de Vylder, B. Saeta, J. Bradbury, D. Ding, S. Borgeaud, M. Lai, J. Schrittwieser, T. Anthony, E. Hughes, I. Danihelka, and J. Ryan-Davis, "OpenSpiel: A framework for reinforcement learning in games," http://arxiv.org/abs/1908.09453, 2019.

[33] M. Madden and T. Howley, "Transfer of experience between reinforcement learning environments with progressive difficulty," *Artif. Intell. Review*, vol. 21, no. 3–4, pp. 375–398, 2004.