Detect, Explore and Neutralize New Threats using Reinforcement Learning

Chapman Siu
Faculty of Engineering and Information Technology
University of Technology Sydney
chapman.siu@student.uts.edu.au

Richard Yi Da Xu
Faculty of Engineering and Information Technology
University of Technology Sydney
yida.xu@uts.edu.au

Abstract

Reinforcement learning has been shown to perform a range of complex tasks through interaction with an environment or collected leveraging experience. However, many of these approaches presume a consistent environment or (near) optimal experiences. In this report, we examine approaches to build reinforcement learning policies which are can robustly adapt to new scenarios and strategies for agents to effectively share experiences. We demonstrate that these approaches can outperform several strong baseline models in different contexts across single-agent continuous environments, and our structured exploration approach can be applied to multi-agent environments to effectively explore allowing policies to be trained in an more efficient manner.

1 Introduction

Reinforcement learning typically focuses on accumulating the maximum cumulative discounted reward as part the environment. Through reinforcement learning, near-optimal behavioral skills represented by policies have been developed, which has allowed for excellent results across a wide variety of problem domains [13, 27]. In the typical reinforcement learning scenario where agents have access to their environment, active data collection is encouraged, and in doing so, algorithms are developed which discard or ignore sub-optimal behavior of an agent [29, 1]. Instead, a rather greedy approach is typically used in reinforcement learning, which focuses on mode-seeking behavior, and over modeling the density of the whole space [11, 28]. To this end, approaches generally have a bias towards high rewards, whilst penalizing and even outright 'forgetting' about bad experiences or rewards. This is often formulated as *optimism* in reinforcement learning, which is typically formulated in the regret settings [17]. This becomes an even greater challenge in multi-agent reinforcement learning (MARL) scenarios, as the nature of the joint state action space leads to complexity in estimation, as the size of the joint action space grows exponentially with the number of agents. Structured exploration techniques have been explored through credit assignment [44] or through learning a latent policy conditional on global state information [26]. However both of these approaches require learning explicit cooperative structure which we argue is inefficient and not necessary.

We are interested in developing methods which are capable of detecting new, or unseen states, which can still develop robust policy by remaining within its distribution of experiences. This approach can be used across a variety of reinforcement learning problems, whether it be in an adversarial, offline,

or partially observable setting. That is, can we develop methods which can be trained offline, and online but still remain competitive to approaches which generally approach one or the other?

The goal of this article is to provide the reader with perspectives on how to tackle these two challenges; building robust policies in the face of new threats, and develop structured exploration approaches. This this end we present an opinionated approach and highlight important challenges, which can serve as a starting point for future endeavours. We will cover several approaches for each challenge, and discuss initial steps taken to mitigate these challenges, and conclude with some perspectives on future work in this space.

1.1 Preliminaries

1.1.1 Markov Games

The goal in reinforcement learning is to maximize the expected cumulative reward in a Markov decision process, through optimizing a learned policy. We define the tuple $\langle \mathcal{N}, G, \{S_{\mathsf{a}_i}\}_{\mathsf{a}_i \in \mathcal{N}}, \mathcal{A}, T, r, \gamma \rangle$ to represent partially observable Markov game for agents $\mathsf{a}_i \in \mathcal{N} = \{\mathsf{a}_1, \ldots, \mathsf{a}_N\}$ [23], global state space G, and joint action space $\mathcal{A} = A^1 \times \cdots \times A^N$. Each agent a_i only perceives $s_{\mathsf{a}_i} \in S_{\mathsf{a}_i}$ which depends on the current global state, $T(s'_{\mathsf{a}_i}|s_{\mathsf{a}_i}, a_{\mathsf{a}_i})$ and $r(s_{\mathsf{a}_i}, a_{\mathsf{a}_i}$ represent the dynamics and reward function, with $\gamma \in (0, 1)$ being the discount factor. Let $\langle s, a, r, s' \rangle \sim \mathcal{D}_{\mathsf{a}_i}$ represent the dataset of experiences stored as a transition tuple for agent a_i, with s' representing the state observed after taking action a in state s and receiving reward r. In this report we are particularly interested in multi-agent algorithms which only use each agent's local observation during training and execution, which is known as the *centralized training decentralized execution* paradigm.

1.1.2 Q-Learning

Q-learning is an off-policy RL algorithm based on dynamic programming, which uses the action-value function $Q(s, a)$. In Deep learning approaches parameters this Q function uses a deep neural network [27, 8]. Q-learning methods apply the Bellman operator $\mathcal{B}^* Q(s, a) = r(s, a) + \gamma \mathbb{E}_{s' \sim P(s'|s,a)}[\max_{a'} Q(s', a')]$. Given the tuple $\langle s, a, r, s' \rangle \sim \mathcal{D}_{\mathsf{a}_i}$, the iterative updating for training a particular agent a_i with tradeoff factor α:

$$\hat{Q}^{k+1} \leftarrow \arg\min_{Q} \alpha \cdot \mathbb{E}_{s \in \mathcal{D}, a \sim \pi(a|s)}[Q(s, a)] + \frac{1}{2} \mathbb{E}_{s,a,rs' \sim \mathcal{D}} \left[\left(Q(s, a) - \mathcal{B}^* \hat{Q}^k(s, a) \right)^2 \right] \quad (1)$$

Where $\hat{Q}$ is typically a *target network* where the parameters kept constant for a number of iterations and periodically copied.

1.1.3 Actor-Critic

An extension to the Q-learning approach is the actor-critic framework. In this setting, once we an estimation for $\hat{Q}$ (called the critic), we can leverage this to construct the policy (or the actor). These methods general leverage training the actor through gradient methods through the total discounted reward objective $J = \mathbb{E}_{[} \sum_{t=0}^{T} R_t]$, where R_t is the discounted reward, with discount rate $\gamma \in (0, 1]$ defined by $R_t = \sum_{l=0}^{\infty} \gamma^l r_{t+l}(s, a)$. If we approximate the objective with the Q function, this leads to DDPG algorithm, that is if the actor, π is parameterised by θ, $\max_\theta \mathbb{E}_{s \sim \mathcal{D}}[Q(s, \pi_\theta(s)]$ [22].

2 Detecting Distributional Shifts in Reinforcement Learning

In order to detect and adapting for distributional shifts in reinforcement learning, we turn our attention to methods which constrains the learning of the policy to a *data-driven* approach, one that optimizes a policy with minimal or no on-policy interaction with the environment. This is because on key focus area is answering *counterfactual* queries. This is where we form hypothesis about what *might* happen if agents perform an action different from one that is seen in the data [21]. In these scenarios, the agent might be trained under one distribution, but evaluated under a different one. This can be addressed in different ways, such as constraining the learning process such as the distributional shift is bounded [19, 10, 43, 33], or through providing conservative lower bounds to our estimation

[20], or through providing algorithms which generate more robust policies through estimation of a distribution or the addition of noise [13, 22, 8, 41].

2.1 Improving through Self-Imitation and Hindsight

One approach to improving the policy without collecting new experiences is to leverage it in different ways.

Self-Imitation Learning (SIL) [29] builds on top of SAC, whereby at every time step, an additional update is performed using the clipped advantage to update the actor-critic model. The clipped advantage is in the form $(R^{sil} - V_\theta(s))_+$, where $R^{sil} = \sum_{k=0}^{\infty} \gamma^k r_k$ is the discounted sum of rewards, where r_k and γ is the reward at time k and the discount rate respectively, $(\cdot)_+ = \max(\cdot, 0)$ and V_θ is the value function of an actor-critic policy parameterized by θ. In comparison, the clipped advantage used in ARM is in the form $(Q_\theta(s,a) - V_\theta(s))_+$, with the cumulative clipped advantage at training iteration T being $A_T^+ = (A_{T-1}^+)_+ + (Q_\theta(s,a) - V_\theta(s))$, where the key difference is ARM uses the Q function instead of the cumulative discounted reward. ARM does not require a separate update step in training the policy, but instead uses the clipped advantage directly to train the policy. However if this approach was extended to policy gradient approaches, as the formulation of the policy update can yield $\log(0)$, ARM models are restricted to Q-learning approaches in the discrete action space [17]. This is not a restriction for SIL approach, as the any update which would have yielded a $\log(0)$ is simply discarded and not used in the additional update step that is performed.

2.2 Towards More Robust Reinforcement Learning Approaches through Distributional and Entropy Considerations

Using off-policy reinforcement learning approaches without considering distribution shift has lead to poor generalisation when evaluating on states-actions outside of its observed space.

In an attempt to stablise learning without explicit distributional control has lead to distributional and entropy considerations in order to create sample-efficient algorithms. One approach in the discrete action space is to learn the value distribution instead of the value function which is used in discrete action spaces [8].

Another approach called *Soft Actor Critic* is to include an entropy regulariser in the actor-critic framework to induce a stochastic actor which leads to greater robustness and stability in learner [13]. This changes the actor objective to the form $\max_\theta \mathbb{E}_{s\sim\mathcal{D}}[Q(s, \pi_\theta(s)] + \alpha H(\pi(\cdot|s))$.

This idea of leverage an entropy regulariser has been extended to the DQN space through reward augmentation approaches [41]. In this setting, the reward $r(s,a)$ is replaced by $r'(s,a) := r(s,a) + \alpha \log \pi(a|s)$.

Conservative Q-Learning (CQL) extends approaches taken in DQN, which leverages information from the empirical behavioral distribution. Let $\hat{\pi}_\mathcal{D}(a|s) := \frac{\sum_{s,a\in\mathcal{D}} \mathbf{1}[s=s,a=a]}{\sum_{s\in\mathcal{D}} \mathbf{1}[s=s]}$ denote the empirical behavioral distribution at the state s. Then by adding an additional Q-value *maximization* term (in red) under the data $\hat{\pi}_\mathcal{D}$, we have the resulting iterative update for CQL:

$$\hat{Q}^{k+1} \leftarrow \arg\min_Q \alpha \cdot \left(\mathbb{E}_{s\sim\mathcal{D},a\sim\pi(a|s)}[Q(s,a)] - \mathbb{E}_{s\sim\mathcal{D},a\sim\hat{\pi}(a|s)}[Q(s,a)]\right) + \frac{1}{2}\mathbb{E}_{s,a,rs'\sim\mathcal{D}}\left[(Q(s,a) - \mathcal{B}^*\hat{Q}^k(s,a))^2\right] \tag{2}$$

This approach provides lower bound on the value of the current policy which provides theoretical improvement guarantees. In our work, we are interested in extensions of this approach through choices of regularizers $\mathcal{R}(\pi)$, in the multi-agent setting, when additional considerations are made with respective to leveraging other agent experiences or inducing a more structured exploration strategy. This is described in the form:

$$\hat{Q}^{k+1} \leftarrow \arg\min_{Q} \alpha \cdot \left(\mathbb{E}_{s\sim\mathcal{D},a\sim\pi(a|s)}[Q(s,a)] - \mathbb{E}_{s\sim\mathcal{D},a\sim\hat{\pi}(a|s)}[Q(s,a)]\right) + \frac{1}{2}\mathbb{E}_{s,a,rs'\sim\mathcal{D}}\left[\left(Q(s,a) - \mathcal{B}^*\hat{Q}^k(s,a)\right)^2\right] + \mathcal{R}(\pi) \quad (3)$$

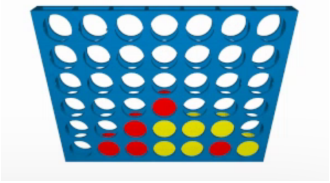
CQL presents several variations for the regularizer $\mathcal{R}$ with closed form solutions, including entropy approach $\mathcal{R}(\pi) = H(\pi)$ and a KL-divergence approach to a prior distribution ρ in the form $\mathcal{R}(\pi) = -D_{\text{KL}}(\pi\|\rho)$. These approaches have been empirically demonstrated to be more stable in high-dimensional action spaces [20].

2.3 Behaviour Constrained Reinforcement Learning

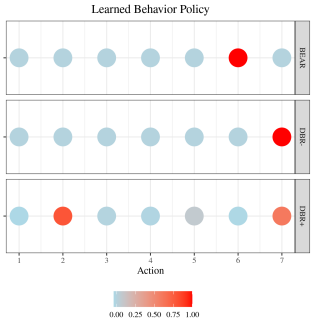
All the previous algorithms operate in an off-policy scenario, where we aim to be more sample efficient in our training algorithm. In this section, we examine algorithms which more explicitly constrains the policy update according to the behaviour policy.

One DQN extension is to *Batch Constrained Q-learning* (BCQ). This approach constrains the temporal difference update to be conditional only on observed state, action combinations; thereby avoiding state and action combinations which have not been observed[10]. If the offline data is wholly defined by $\mathcal{D}$, then the Bellman operator in this scenario is $\mathcal{B}^*_{\text{BCQ}}Q(s,a) = r(s,a) + \gamma\mathbb{E}_{s'\sim P(s'|s,a)}[\max_{a' \text{ s.t. } (s',a')\sim\mathcal{D}} Q(s',a')]$, where items in red demonstrates the batch constraint that is included.

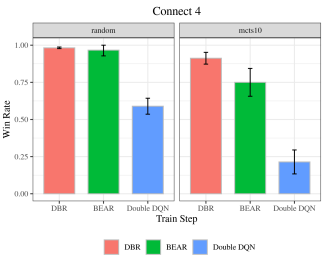
Another group of approaches is to model the behaviour policy itself directly, and constrain updates through regularising the policy with the behaviour policy. This approach modifies SAC model by replacing the entropy regularizer $\max_\theta \mathbb{E}_{s\sim\mathcal{D}}[Q(s,\pi_\theta(s)] + \alpha D(\pi(\cdot|s)\|\beta(\cdot|s))$, where β is the behavioural policy, and D is the choice of divergence function. *Bootstrap Error Accumulation Reduction* (BEAR) explores the usage of max mean discrepancy divergence function [19], whereas *Behavior Regularised Actor Critic* (BRAC) leverages Kullback-Leibler Divergence [43]. *Dual Behavior Regularised* is a similar approach which reweighs the learned behavioural policy according to the clipped advantage, which allows learning positive behaviours whilst avoiding negative behaviours [33].



(a) Connect-4 Game with Red to Play. Optimal action is to play the right-most file block Yellow's winning move. Image taken from "gameplay of Connect-4", source: Wikipedia - https://en.wikipedia.org/wiki/Connect_Four



(b) Learned behavior policy (normalized) from BEAR, DBR (positive and negative) based on the board presented in Fig 1(a).



(c) Evaluation performance against an (unseen) adversarial agent with MCTS depth 10, with Double DQN as the baseline approach. Error bars represent the standard deviation over these rollouts.

Figure 1: In our Connect-4 environment our agents is trained and only have access against an opponent which makes random moves, and is evaluated on a separate unseen agent trained using Monte-Carlo Tree search with varying depths, all policies were trained for 2 million rollouts against a random agent. DBR β_- successfully identifies the importance of the right most move whereby *not playing* will result in a loss, notice that leveraging β_+ alone in a greedy setting would still fail to identify this action, as it has a preference for a move on the left hand side, whereas BEAR unsuccessfully identifies the correct action.

Dual Behavior Regularized Reinforcement Learning

In constructing Dual Behavior Regularized Reinforcement Learning (DBR), we address some limitations of BEAR and ARM. As part of the BEAR formulation the level of constraint is a fixed constant ϵ which is set to allow the rate of which the BEAR policy can be updated during fine tuning. For ARM, the update formulation is restricted to Q-learning and discrete action spaces. To address these deficiencies we use split the approximation of the behavioral policy into two components, being the "clipped advantage", $(R - V_\theta(s))_+$ and the "*negative* clipped advantage", $(R - V_\theta(s))_-$, where the functions are defined as $(\cdot)_+ = \max(\cdot, 0)$ and $(\cdot)_- = \min(\cdot, 0)$. We define $\hat{\beta}_+ := \text{argmax}_{\hat{\pi}} \mathbb{E}_{(s,a,r,s')\sim\mathcal{D}_+}[\log \hat{\pi}(a, |s)]$ and $\hat{\beta}_- := \text{argmax}_{\hat{\pi}} \mathbb{E}_{(s,a,r,s')\sim\mathcal{D}_-}[\log \hat{\pi}(a, |s)]$, where D_+, D_- are subsets of the replay buffer where $R \geq V_\theta(s)$ and $R \leq V_\theta(s)$ respectively. This changes the formulation in the BRAC framework to be

$$V^\pi(s) = \sum_{t=0}^{\infty} \gamma^t \mathbb{E}_{s_t \sim P_t^\pi(s)}[R^\pi(s_t) - \alpha D(\pi(\cdot|s_t), \hat{\beta}_+(\cdot|s_t))] \tag{4}$$

Whereby we restrict the policy improvement step to be dynamic, based on the behavior policy trained on the negative clipped advantage $\hat{\beta}_-$, which creates the dynamic constraint: $\mathbb{E}_{s\sim\mathcal{D}}[D(\pi(\cdot|s_t), \hat{\beta}_+(\cdot|s_t))] < \max(\epsilon, \mathbb{E}_{s\sim\mathcal{D}}[D(\pi(\cdot|s_t), \hat{\beta}_-(\cdot|s_t))])$. The intuition behind this update in a fine-tuning example, is that as the replay buffer updates with new experiences, we want the divergence between the behavior policies based experiences which yield higher advantage scores to be closer to our trained policy, than the corresponding behavior policy based on experiences which yield lower advantage scores. In this formulation, we avoid the $\log(0)$ issue present in ARM without discarding experiences as in SIL through training via the behavioral policies $\hat{\beta}_+, \hat{\beta}_-$, allowing all experiences to be used in a counterfactural regret minimization setting by optimizing over a surrogate objective $\hat{\beta}_+$, rather than the clipped advantage directly, that is we want to ensure the update is in the form $\text{argmin}_\pi \mathbb{E}_{s\sim\mathcal{D}}[D(\pi(\cdot|s_t), \beta_+(\cdot|s_t)]$ [17, 13]. This allows the clipped advantage to be approximated through the behavioral policy β_+, that is $\beta_+ \approx A_+$

As such our approach is essentially ARM with different design choices around the surrogate objective in order to facilitate ARM-like policy gradient, which was deficient in the original ARM algorithm [17].

Compared with BEAR, this approach intentionally avoids of regions indicated by β_- which is important particularly in environments with severe degenerative states. This is demonstrated in empirical experiences using Connect-4 game, shown in Fig. 1, where it demonstrates DBR's ability to both learn key moves, and avoid key moves over BEAR and DQN.

It is natural to consider the scenarios where the dynamic constraint using β_- is removed, and thereby using a fixed constraint as in BEAR: $\mathbb{E}_{s\sim\mathcal{D}}[D(\pi(\cdot|s_t), \hat{\beta_+}(\cdot|s_t))] < \epsilon$. This would form a variation of the BEAR algorithm where the behavior policy is trained using the "clipped advantage" like the ARM model [17]. Alternatively, if we train using the cumulative reward, rather than the Q value function, with a separate update step, we would have a SIL variation [29]. In constraining the actor policy, we also explore the Munchhausen Reinforcement Learning (MRL) approach, whereby we leverage the entropy regularization during the training of the policy as in SAC and MRL, however when we construct the Munchhausen reward, we instead use $r_t^{mrl} := r_t + \tau \log \beta_+(a|s)$, where $\tau \in [0, 1]$ is the scaling factor.

3 Structured Exploration in Multi-Agent Reinforcement Learning

Attempting to leverage experience of other agents in Q-Learning set up does not greatly improve performance as off-policy samples would not have been corrected from the underlying policy distribution [5], whereas using maximum entropy RL on an individual agent may not guarantee diverse exploration across all agents, requiring a network to learn the appropriate credit assignment [44]. Structured explorations have been explored through to explicitly learning a latent policy conditional on global state information [26]. MARQ aims to address these limitations in the MARL context without explicit cooperative structures through applying regularization constraints which can correct bias in off-policy out-of-distribution agent experiences and promote diverse exploration [35].

Greedy UnMix

The construction of Greedy UnMixing (GUM) takes into account the manner in which conservative QMIX is constructed, with both regularizer on the mixer network and individual network. In determining the updates of the mixer network or the individual policy networks, we consider the nature in which they are updated. Can we bias the weights towards considering individual information only and *purposely* neglect the coordination aspect of the mixer network, thereby "unmixing" the network?

In order to unmix, we use a learnable weight to control the regularization strength for the independent and mixer networks. Intuitively, the "better" the estimate the stronger the strength of the regularizer. In order to construct this, we apply a soft constraint on the regularizer trade-off factor, such that the larger the relative regularizer strength, the smaller its corresponding component has on the neural network weights update.

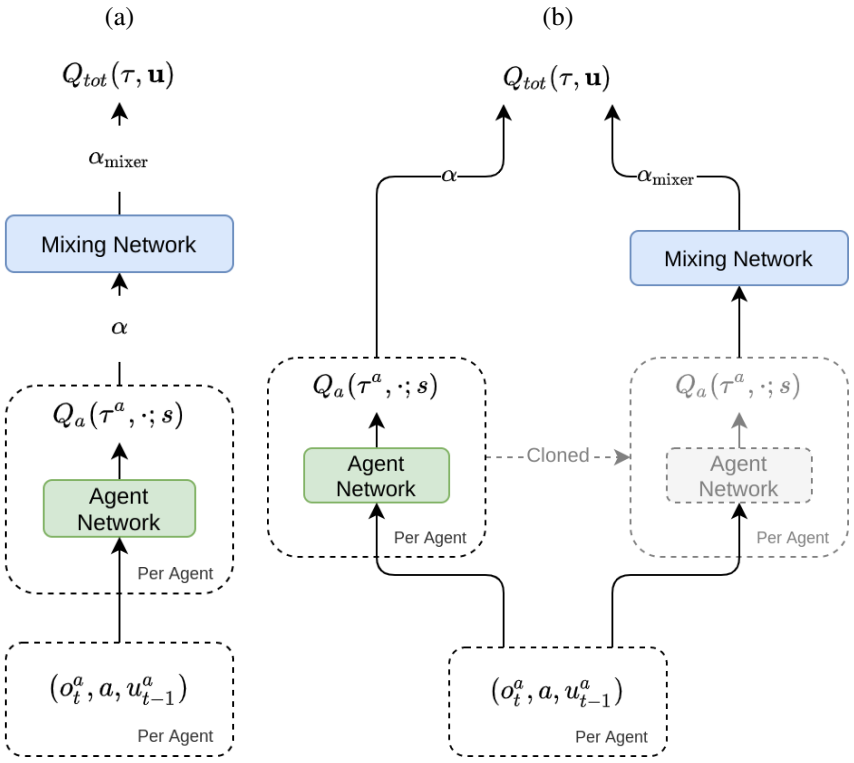


Figure 2: Architecture for the GUM algorithm. (Left, a) The model is implicitly unmixed between the independent policies and cooperative training used in the mixer network, through weighting the regularizer strength through a learned α and α_{mixer}. This allows for learning the assigning learning weights to the independent policy and mixer network in an appropriate manner to avoid overestimation. (Right, b) Architecture for the GUM algorithm can be interpreted as an ensemble of neural network modules in an AdaBoost setting with the candidate network being included is the mixer network. This interpretation allows for explicit unmixing between the independent policies and the cooperative training used in the mixer network.

We explored GUM, in which regularizer trade-off weights were used to implicitly to unmix the cooperative and independent Q functions.

The construction and choice of neural network architectures have been explored as a boosting problem with learning guarantees. AdaNet framework explores neural architecture selection through iteratively proposing candidate networks with hard selection proposals [7], or soft selection proposals through application of a learnable trade-off factor [32]. Through the learnable trade-off α, the GUM algorithm can be interpreted as a neural network architecture selection problem at training time between the mixer network and individual agent networks. This ensemble can be applied implicitly or explicitly

as a stacking ensemble [42] of the individual agent policies and the mixer function. Rather than leveraging a learned parameter for neural network selection problem, an alternative approach is to simply take the minimum ensemble as the solution, which leads to a hard assignment in the choice of the Q function ensemble rather than a soft assignment. This approach is a variety of ensemble Q-learning approaches [22, 10, 19], which is used to stabilize training in the context of homogeneous target networks, as opposed to this scenario where it is between the individual agent policies and the mixer network. Then the Adanet mixer is a linear combination of the individual agent Q function, and the monotonic function factorization, $Q_{\text{adanet}} = \alpha[Q_1, \ldots, Q_n] + \alpha_{\text{mixer}} Q_{\text{mixer}}$, where α and α_{mixer} are learnable parameters by gradient ascent.

Another interpretation of the AdaNet neural network selection setting, is that the base learner is the independent Q-learning policies, which are then boosted (ensemble) from the subsequent mixer network. This interpretation, also addresses the overestimation concerns in reinforcement learning, but penalizes the additional weight inversely to the regularizer strength. There have been several approaches leveraging ensemble of Q functions which assume independence and their analysis results would not apply here either [10, 19]. In the boosting setting, our approach can be interpreted as a variation of AdaNet [7], in the bushy construction, whereby the independent Q-learning model is the base policy, and the mixer is the boosted network module. Rather than having hard update or stopping rule, we leverage a soft choice as a shrinkage parameter [32]. In this setting, the networks are explicitly combined rather than implicitly through the regularizer.

We present the two variations of GUM, one which allows for implicit unmixing through learning the trade-off factors through an inverse weighting scheme, and the other approach in which the combination of the mixer and independent policies are explicitly learned.

Multi-Agent Regularized Q-learning

In order to have individual agents in a MARL framework to explore effective, we explicitly regularize using KL divergence to encourage structured exploration. This approach was explored as part of Conservative Q-learning [20] in single-agent framework and *Learning Implicit Credit Assignment* (LICA) within the multi-agent framework, however the structured exploration was only through the entropy by treating each agent independently. The *adaptive entropy regularization*, $\mathcal{H}$ is created through dynamically controlling the magnitudes of the entropy gradients so that they are inversely proportional to the policy entropy during training

$$\begin{aligned} \mathcal{H}_i &:= \alpha H(\pi_{\mathsf{a}_i}) \\ \partial \mathcal{H}_i &:= -\alpha \cdot \frac{\log \pi_{\mathsf{a}_i} + 1}{H(\pi_{\mathsf{a}_i})} \end{aligned}$$

where H is the entropy, π_{a_i} is the policy of agent a_i, and α is a constant controlling the regularization strength. In this particularly instance, as we are not operating in an actor critic framework, we are regularizing in maximum entropy RL framework using distributional RL. This approach is used to ensure there is sufficient sustained exploration through training. To extend this to our approach where there is no mixing network or explicit cooperation, we use cross-entropy penalty to encourage agents to explore diverse trajectories.

In order to address this, we first observe the definition of cross entropy $H(p, q) = H(p) + D_{\text{KL}}(p \| q)$. Then we construct Q-learning update based on Equation 3 with the pairwise adaptive cross entropy regularization penalty (in red) to be:

$$\begin{aligned} \hat{Q}^{k+1}_{\mathsf{a}_i} \leftarrow \arg\min_Q \alpha \cdot \Big(\mathbb{E}_{s \sim \mathcal{D}_{\mathsf{a}_i}, a \sim \pi_{\mathsf{a}_i}(a|s)}[Q(s,a)] - \mathbb{E}_{s \sim \mathcal{D}_{\mathsf{a}_i}, a \sim \hat{\pi}_{\mathsf{a}_i}(a|s)}[Q(s,a)] \Big) \\ + \frac{1}{2} \mathbb{E}_{s,a,rs' \sim \mathcal{D}_{\mathsf{a}_i}} \left[\left(Q(s,a) - \mathcal{B}^* \hat{Q}^k(s,a) \right)^2 \right] + \lambda \sum_{\mathsf{a}_j \in \mathcal{N}} \left(H(\pi_{\mathsf{a}_i}) + D_{\text{KL}}(\pi_{\mathsf{a}_i} \| \pi_{\mathsf{a}_j}) \right) \end{aligned} \tag{5}$$

Where λ is the regularization strength. Under this framework, in addition to the adaptive regularizer used in LICA, the regularizer would leverage pairwise experience information. Since cross-entropy

is defined as $H(p, q) = H(p) + D_{\text{KL}}(p\|q)$, then the nonnegativity of the KL divergence on the righthand side implies that a agent policies which behave in the same manner induces a regularizer penalty, encouraging diversity, which is offset by entropy. Not only does our approach extend the LICA adaptive entropy, it is a natural extension to a MARL variation of CQL models.

4 Neutralize through Coordination and Cooperation

Reinforcement learning (RL) has increasingly greater real-world applications, ranging from autonomous vehicle coordination [3], and controlling robotic swarms [15] which can be framed as multi-agent cooperative problems. In these settings, agents act only on their partial observability and/or communication constraints leading to learning *decentralized policies*, which only rely on each agent's local state-action history. Although single-agent RL algorithms can be applied to these environments [39], this approach is not inherently scalable nor does it allow for agents to leverage each others experience or encourage structured exploration in the cooperative multi-agent setting.

In Multi-Agent Reinforcement Learning (MARL), the nature of the joint state action space leads to complexity in estimation, as the size of the joint action spaces grow exponentially with the number of agents. Leveraging traditional Reinforcement Learning approaches which model the combined joint action and state space is often impractical or does not appropriate reflect constraints which may be present in the evaluation environment such as partial observability and/or communication constraints leading to learning *decentralized policies*.

4.1 Multi-Agent Coordination

A common approach used to cope with large action spaces is to decentralize the decision policy and/or value function, whereby an agent policy is conditioned on only their own local action-observation history. In order to facilitate training and communication amongst agents, learning may occur in a centralized fashion to provide additional state information and to remove inter-agent communication constraints. This paradigm is known as *centralized training with decentralized execution* [30, 18]. Value Decomposition Networks (VDN) [38] perform central Q-learning with a value function that is the sum of independent utility functions for each agent. QMIX [31, 34] extends VDN through employing a mixing network for the agent utilities with monotonic constraints, which allows different mixtures in different states so that the central value can be maximized independently. These two methods condition on an agent's utility on its history, i.e. the past observations and actions and share the parameters of all utility functions as shown in Figure 3(a). DCG [2] is another decentralized agent execution approach that uses the addition of one extra linear layer which is used to approximate a coordination graph's (CG) *pairwise payoffs*. For DCG, parameter sharing is employed to approximate *all* payoff functions in the same neural network as shown in 3(b). QTRAN [37] is another approach which attempts to calculate the central Q-learning through applying linear constraints between agent utilities and joint action values, which removes structural constraints of factorizable MARL tasks. Actor-critic approaches for MARL problems have also been explored [24, 44, 9].

QCGraph

We introduce the Q-value Coordination Graph (QCGraph) [36], which learns the utility and payoff functions of a coordination graph. This approach aims to unify other CG approaches (QTRAN and DCG) in an overall framework, as well as remove constraints on having fixed pre-determined coordination structures. Figure 4 illustrates the overall setup for QCGraph.

The agent network (shown in green, or bottom-left side of Figure 4) describes the agent utility network. Each agent network represents its individual utility function $q^i(\tau^i, u^i)$. At each time step, each agent network receives a local observation as input followed by a GRU recurrent layer and a fully-connected layer with $|U|$ outputs. This setup is the same one used in QMIX, LICA, and DCG [31, 44, 2]. The output of the utility network is both the utility value itself, q^i and also the utility network's hidden feature layer. This approach of emitting both the utility value and recurrent network's hidden state, h^i to the mixing network for centralized learning is used in both QTRAN and DCG [2, 37].

The mixing network (shown in blue, on right side of Figure 4). The first "module" is the graph node sorting operation. For each agent i, as q^i can be interpreted as a 1D projection of the hidden state h^i,

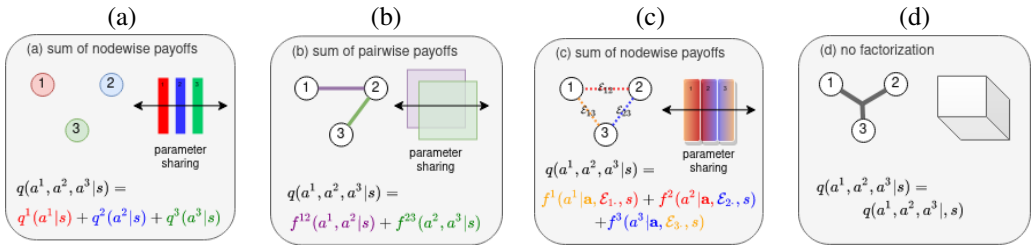


Figure 3: Examples of value factorization are shown in (a, b, c, d), examples of coordination graph factorization shown in items (b, c, d): (a) sum of independent utilities as used in VDN, QMIX; (b) sum of pairwise payoffs as used in DCG; (c) sum of node-wise payoffs based on dynamically generated coordination graph (our approach); (d) single hyper-edge with no factorization as used in QTRAN.

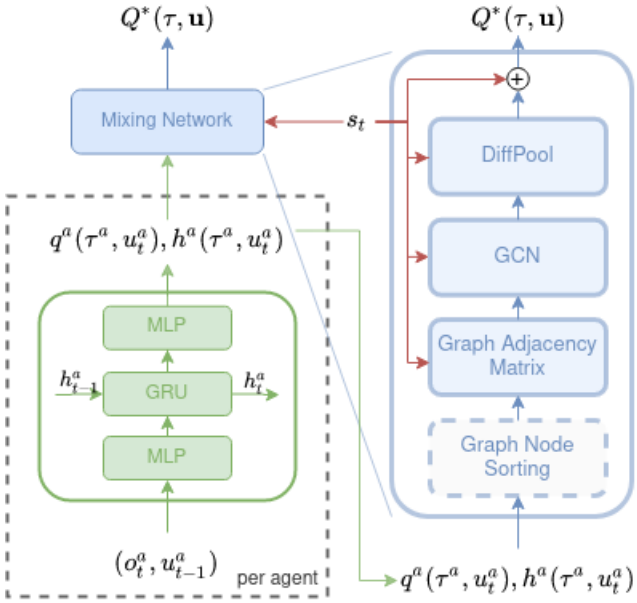


Figure 4: QCGraph network structure. In blue is the mixing network, and in green is the agent network structure. On the left is the overall network structure, on the right is the mixing network, where h represents the hidden layer of the agent utility network. The dotted outline around "Graph Node Sorting" indicates that there are no learnable parameters as part of this operation.

this can be used to pre-sort the node feature matrix as required for permutation invariance. The next hypernetwork generates the graph adjacency matrix. The learnable parameters consist of a single linear layer and the state bias, as the node feature matrix project is subsumed in the first module of the mixing network. The final state bias is constructed through a two layer hypernetwork with ReLU non-linearity, in the same manner used by QMIX. Through the use of graph neural network techniques in its setup, QCGraph generally would have comparable number of parameters, or in most cases less than QMIX[1].

The agents and mixing network are trained in a centralized manner, and each agent uses its own utility function q^i to take an action during decentralized execution. QCGraph is trained end-to-end to minimize the loss as defined by QTRAN [37].

In general CG approaches are used to prevent *relative overgeneralization* during the exploration phase of agents. DCG and QTRAN both enable estimating a value function over a joint action space.

[1]In QMIX, the state bias is projected to every agent in the hypernetwork. This is replaced with graph neural network modules instead. More details are provided in supplementary materials.

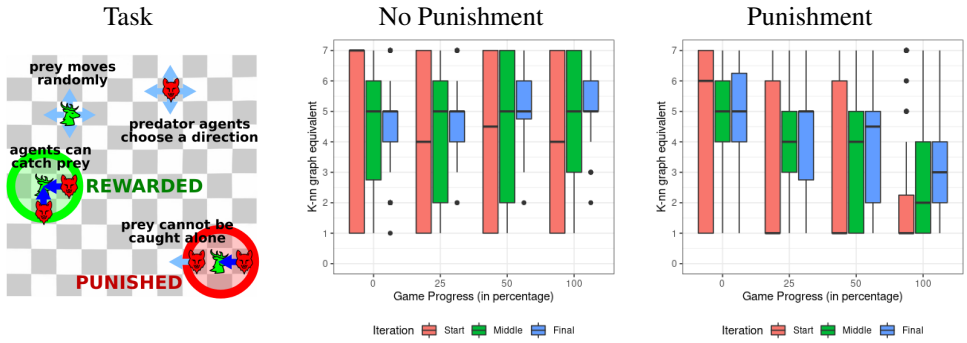


Figure 5: Visualized k-nearest neighbor adjacency matrix representation in QCGraph in the modified predatory prey task (left) over two environments where there is no penalty for catching a prey alone (middle), and where there is a penalty (right). The box plots represents the distribution of the dynamic k-nearest neighbor adjacency matrix representation generated during training at the start, middle and end of training for the task. The k is determined by the number of agents deemed *Relevant*. In the scenario where task emits no punishment, QCGraph tends towards QTRAN-style model (i.e. higher k) (middle), on the other hand, when the task emits punishment, QCGraph favors coordination with lower number of agents, tending towards DCG-style model.

This approach is useful in certain environments where cooperation is important. For example, in the predator-prey environment, the predators may need to cooperate together in order to capture large or dangerous prey. From the perspective of each predator, the expected reward will depend on the action of the other agent. An example of this is shown in Figure 5, which demonstrates QCGraph's representational ability to adapt and build structures which transition towards more rewarding joint behaviour, which existing CG approaches would struggle to accomplish. In models which do not take into account joint actions (QMIX, IQL) it is not possible to learn the optimal policy. Furthermore QCGraph has the ability to dynamically adapt the level of coordination graph over various states which becomes important in war game scenarios, such as predator-prey environments, where the level of appropriate cooperation may differ according to the formation or number of agents available at a particular point in time.

5 Application and Evaluation

Dual Behavior Regularized Reinforcement Learning Experiments

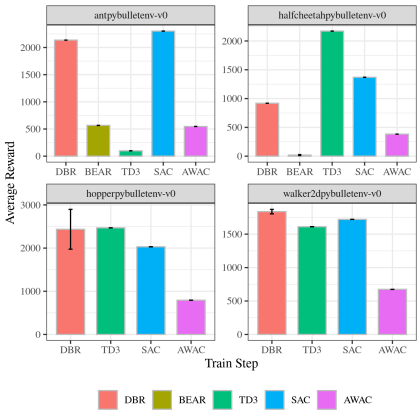
We hypothesise that DBR should perform well in both offline reinforcement learning and other reinforcement learning settings where it is forced to adapt. We conduct our experiments on the standard PyBullet environments and in the offline reinforcement learning, where the agents are trained on static dataset of experiences.

Connect-4

Connect-4 is a non-trivial board game of medium-high complexity, for which we constructed a heuristic agent using Monte-Carlo Tree Search [4] as an agent to determine our approaches' ability to generalize beyond random actions and explore its representative power. We use Deep Reinforcement Learning with Double Q-learning (Double DQN) [14] as our benchmark approach due to the discrete nature of the environment. We modify both BEAR and DBR using discrete variation of SAC [6]. Figure 1 demonstrates the ability for the various approaches to generalize, as we observe the level of degeneration when the agents move from the training environment to the evaluation environment.

PyBullet Environments

We train our agents on PyBullet environments *tabula rasa*. Although algorithms such as BEAR, AWAC are not necessarily designed to be used without the presence of offline data, these experiments demonstrate DBR's extensions to BEAR to enable dynamic control for exploration in this



(a)

Figure 6: Performance over a range of PyBullet environments, when trained from scratch without the presence of offline dataset. We report the mean over the last 10 evaluation steps, where each evaluation point is average over 1000 episodes. Error bars represent the standard deviation over these rollouts.

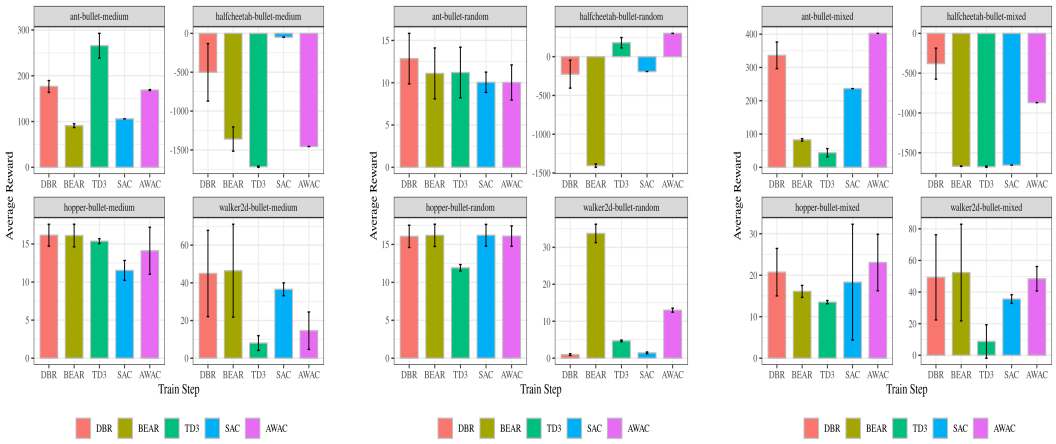
scenario, and explore the performance of AWAC where training is performed from the start; without the presence of offline datasets. This demonstrate DBR is superior to the offline reinforcement learning approaches (BEAR, AWAC), whilst comparable to baseline approaches (TD3, SAC). This demonstrates the reliance AWAC has in the implicit constrained on observed data, as it needs to build up the experience itself, which suggests its limitations in learning *tabula rasa*. In comparison, DBR demonstrates this compromise between the offline reinforcement learning techniques and the off-policy techniques, and able to balance between the two competing priorities.

Offline Reinforcement Learning

We train our agents on PyBullet Offline Reinforcement Learning datasets[2]. These datasets are a frozen set of experiences generated from a SAC agent under different setups. The offline datasets used for each of the competing agents are identical. Furthermore the performance shown below is based on the agent interacting directly with the environment for evaluation purposes, and is not used for training. We compare the efficacy of our approach on three different sets of imperfect demonstration data, these being (1) completely random behavior policy, (2) a partially trained, medium scoring policy, and (3) a samples of experiences generated when constructed the partially trained agent. We trained behavior policies using Soft Actor-Critic algorithm [13]. In all cases, random data was generated by running a uniform at random policy in the environment. Mediocre data was generated by training SAC agent for 1 million timesteps for each of the environment to yield "medium" agents, at which case the agent policy was frozen, and the data was generated against this agent. Mixed data was generated by leveraging the experiences created in the training of the mediocre agent. We used the same datasets for evaluating different algorithms to maintain uniformity across results.

We provide the average return and the standard deviation across all training runs in Table 1.

[2]the generated PyBullet offline reinforcement learning examples are available from: https://github.com/takuseno/d4rl-pybullet



(a) "Medium" Offline Datasets (b) "Random" Offline Datasets (c) "Mixed" Offline Datasets

Figure 7: Comparison over PyBullet Control environments trained using offline datasets only, using (a) partially trained, medium scoring policy (b) randomly generated actions, (c) mixture of agent experiences gathered during training.

Table 1: Results in all tasks for the different environments. The average return and the standard deviation during evaluation is presented. Discrete environments in Connect-4 for BEAR and DBR were implemented using Soft-Actor Critic discrete variations [6]

Tasks/Algorithms	DBR	BEAR	TD3	SAC	AWAC	DDQN
Connect 4 – Random	0.982 ± 0.0158	0.967 ± 0.116	–	–	–	0.589 ± 0.161
Connect 4 – MCTS 10	0.912 ± 0.0793	0.75 ± 0.187	–	–	–	0.214 ± 0.161
Ant-bullet (Offline – Medium)	175 ± 2	91 ± 18.25	266 ± 53.9	106 ± 2	169 ± 2	–
Halfcheetah-bullet (Offline – Medium)	-1539 ± 8.2	-1359 ± 312	-1716 ± 14.7	-49.9 ± 9.2	1456 ± 6.4	–
Hopper-bullet (Offline – Medium)	17.3 ± 6	16.1 ± 2.97	15.4 ± 0.65	11.5 ± 2.6	14.1 ± 6.17	–
Walker2d-bullet (Offline – Medium)	33.3 ± 4.8	46.5 ± 49.3	8.03 ± 7.83	36.6 ± 6.81	14.6 ± 19.8	–
Ant-bullet (Offline – Mixed)	292 ± 0.255	82.4 ± 7.32	43.7 ± 24.9	236 ± 0.153	403 ± 0.119	–
Halfcheetah-bullet (Offline – Mixed)	-1694 ± 9.3	-1669 ± 14.78	-1675 ± 19.2	-1654 ± 8.7	-870 ± 3.1	–
Hopper-bullet (Offline – Mixed)	12.1 ± 3.38	16.1 ± 2.88	13.5 ± 0.783	18.3 ± 27.9	23.1 ± 13.6	–
Walker2d-bullet (Offline – Mixed)	39.3 ± 7.99	52.3 ± 61.2	8.74 ± 21.2	35.6 ± 5.51	48.4 ± 15.5	–
Ant-bullet (Offline – Random)	13.6 ± 5.73	11.1 ± 6.02	11.2 ± 5.99	10.1 ± 2.4	10.0 ± 4.15	–
Halfcheetah-bullet (Offline – Random)	-226 ± 359	-1405 ± 42.7	180 ± 133	-192 ± 8.2	301 ± 4.6	–
Hopper-bullet (Offline – Random)	16.6 ± 2.88	16.2 ± 2.93	11.9 ± 0.856	16.2 ± 2.86	16.1 ± 2.68	–
Walker2d-bullet (Offline – Random)	0.822 ± 0.490	33.7 ± 4.97	4.69 ± 0.426	1.45 ± 0.465	13.0 ± 1.08	–
Ant-pybullet-v0	2136 ± 6.5	567 ± 9.3	98.9 ± 3.4	2302 ± 7.5	547 ± 9.2	-
Halfcheetah-pybullet-v0	919 ± 9.4	19.3 ± 18	2172 ± 4.5	1370 ± 3.2	383 ± 2.8	–
Hopper-pybullet-v0	2435 ± 923	33.4 ± 1.83	2470 ± 9.3	2031 ± 7.4	794 ± 6.1	–
Walker2d-pybullet-v0	1837 ± 758	47.0 ± 16.9	1611 ± 117	1721 ± 7.39	6.75 ± 5.82	–

5.0.1 Performance on Medium-Quality Data

We first discuss the evaluation of condition with "mediocre" data, as this is resembles settings where training on offline data would be the most useful. One million transitions from a partially trained policy were collected, to simulate imperfect demonstration data or data from a mediocre prior policy. Compared with BEAR, it consistently performed similarly or noticeably stronger as shown in Figure 7(a). This scenario is the most relevant, as random data may not have adequate exploration to learn a good policy. We observe in the two scenarios where SAC or TD3 outperformed DBR, DBR was the clear second-best algorithm in the "ant-bullet" and "halfcheetah-bullet" environments respectively.

5.0.2 Performance on Random and Mixed-Quality Data

In Figure 7, we show the performance of each method when trained on data from a random policy (b) and a mixed policy (c). The "random" dataset was created from generating data made from an agent randomly sampling from the environment's action space, whereas "mixed" data was creating from sampling the experiences of an agent during training. In both cases, our method DBR, achieves good results, generally performing comparable or better than BEAR and other approaches, with the exception of "walker2d-bullet-random" and "hopper-bullet-mixed" environments. These results are generally consistent with the expectation that BEAR, DBR would be more robust to the dataset composition.

5.1 Multi-agent Experiments

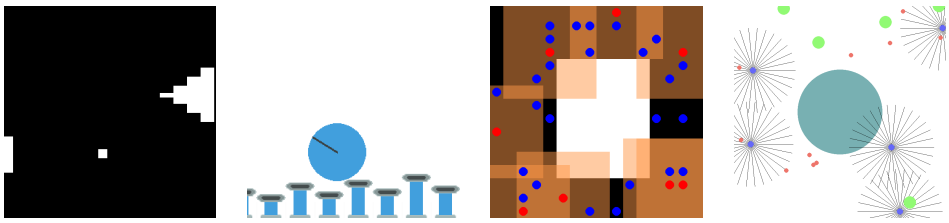


Figure 8: Left to right: Pong, Pistonball, Pursuit, Waterworld environments.

In this section we describe experiments we use to empirically justify our algorithm and approach. We use several reinforcement learning benchmarks [40, 24, 12], shown in Figure 8.

Cooperative Pong is a multi-agent game of pong from "Butterfly" environments [40], where the goal is for both agents (paddles) to keep the ball in play. The game is over if the ball goes out of bounds from either the left or right edge of the screen. In this setting the action space is discrete, and the paddles can either move up or down. To make learning more challenging the right paddle is tiered cake-shaped by default, and the observations of each agent are its own half of the screen. The agents receive a positive, fixed, combined reward based on successful completion of a game frame, otherwise a negative reward is given, and the game ends.

Pistonball is a physics based cooperative game from "Butterfly" environments [40] where the goal is move the a ball to the left wall of the game border by activating vertically moving pistons. Each piston agent's observation is an RGB image of is local surrounding area. The agents have a discrete action space, and can choose to stay still, move down or move up. Pistons must learn highly coordinated emergent behavior to achieve an optimal policy for the environment. Each agent gets a reward based on the overall movement of the ball, and how much the ball moved in a left-wards direction if it was in the agent's local observation.

Pursuit is an environment from the "SISL" set of environments [12], where our agents are control of pursuer agents, which aim to capture randomly controlled evader agents. Evader agents are only removed from the environment when pursuers fully surround an evader. Agents receive a small reward when they touch an evader, and a large reward when they successfully surround and remove an evader agent. In this scenario, each pursuer agent observes a local RGB grid, and operate in a discrete action space.

Waterworld is a simulation of archea trying to navigate and surviving their environment from the "SISL" set of environments [12]. The goal of the agent is to control the archea to simultaneously avoid

poison and find food. The original environment consists of a continuous action space, indicating their horizontal and vertical thrust within the environment. We instead modify this to a discrete action space, in order to assess Q-learning approaches. Within this environment, the agent's observation is based on range limited sensors to detect neighboring entities, indicated by black lines in the right-most image in Figure 8.

Reference and *Spread* are environments consisting of agents and landmarks from the "MPE" set of environments [24]. In *Reference*, each agent wants to get closer to their target landmark, which is known only by other agents. The agents are rewarded by the distance to their target landmark. In *Spread* Agents must learn to cover each landmark without colliding with other agents. The agents are rewarded based on how far the closest agent is to each landmark, and are penalized if they collide with other agents. For both environments, the agent's observation space consists of distance to every landmark, and the action space consists of the movement action. In addition the *Reference* environment has an additional actions to communicate with other agents.

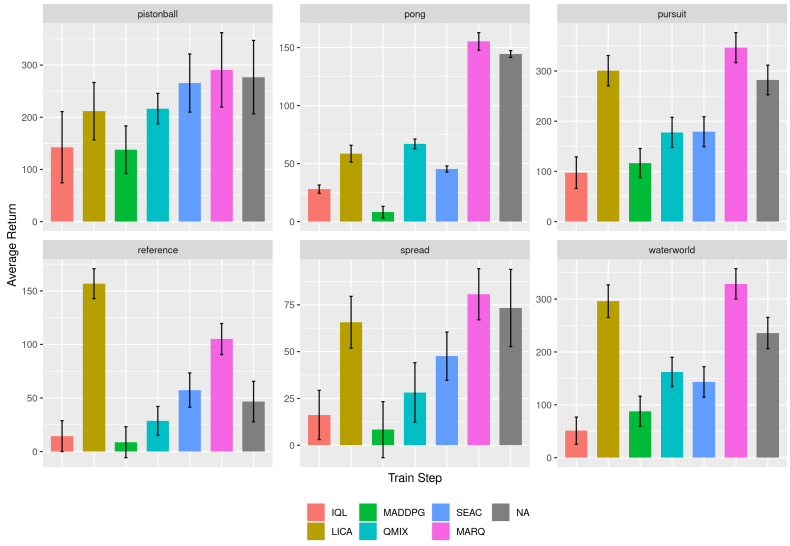


Figure 9: Performance over a variety of benchmark environments. The bar graph report the mean over the last 10 evaluation steps, where each evaluation point is average over 1000 episodes. Error bars for plots represent the standard deviation over these rollouts.

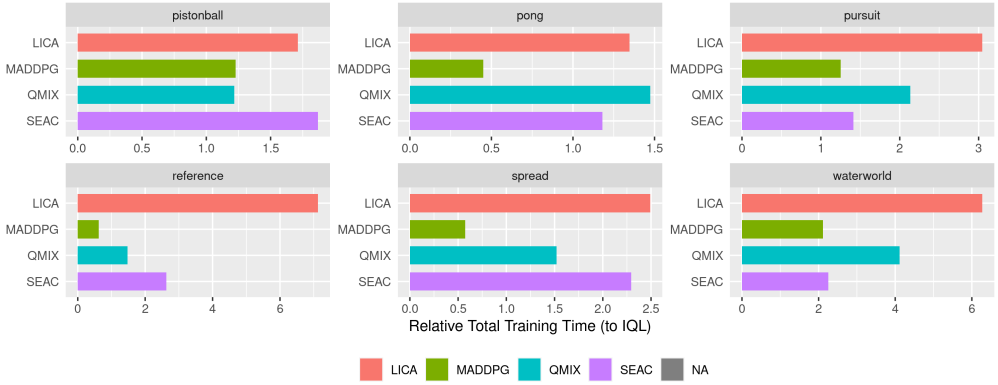


Figure 10: Comparison of the relative total time taken in training the different approaches, using IQL as the benchmark.

The policies leverage the default hyper-parameters based on the official implementation of LICA, QMIX, IQL and their accompanying code based on the pymarl library [31, 44, 37]. SEAC was

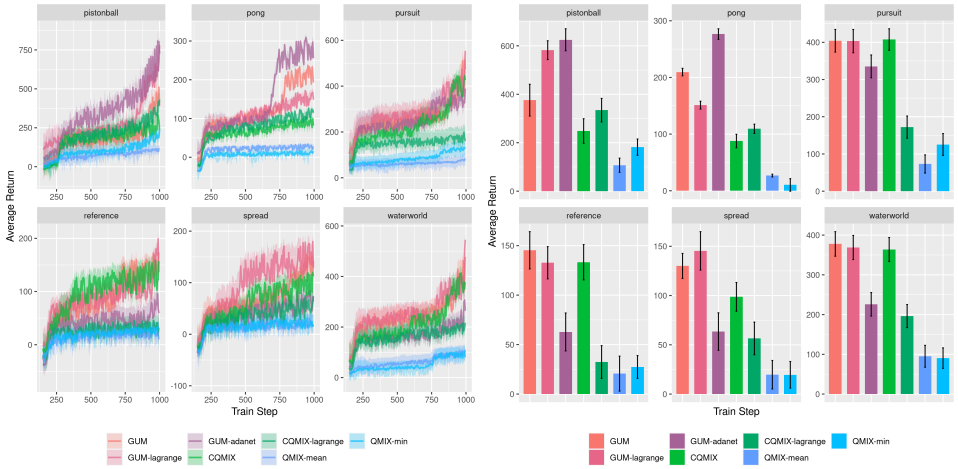


Figure 11: Performance over a variety of benchmark environments as part of our ablation studies for GUM algorithm. The bar graph reports the mean over the last 10 evaluation steps, where each evaluation point is average over 1000 episodes. Error bars for plots represent the standard deviation over these rollouts. This demonstrates the effectiveness of the GUM algorithm, as well as justifying the theoretical results, in particular the overestimation present in the explicit approach ("GUM-adanet").

re-implemented in rlkit framework with the same hyperparameters proposed in their paper, but using Soft Actor-Critic instead of A2C. Similarly, MADDPG extended the rlkit implementation of DDPG to support multi-agent setting. In order to facilitate discrete action spaces gumbel softmax [16, 25] is used as proposed by the approach taken by by SEAC [5]. Similar to the usage in the pymarl library, parameter sharing was used in all models in order to improve the general performance of the agents.

In this section, we explore the effects of different variations of our algorithm on the performance of the agent in order to determine which algorithm generalizes best. We list the "GUM" where the trade-off factor for the regularizer α is fixed, "GUM-lagrange" to be the GUM algorithm where the trade-off factor for the regularizer α is learned, "GUM-adanet" to use an explicit stacking with a fixed trade-off factor for the CQL regularizer. We contrast our approach CQMIX, which applies CQL over QMIX, which does not attempt to unmix the independent Q function and the mixing function, with a fixed trade-off factor for the regularizer and learned trade-off factor in algorithms labeled "CQMIX" and "CQMIX-lagrange" respectively. Finally to demonstrate the effectiveness of the algorithm is not related to the ensembling of agent Q policies with the mixing function, we use ensembling schemes involving the minimum or mean used in other RL approaches [22, 10, 19] labeled "QMIX-min" and "QMIX-mean" respectively.

From the results, we observe that the GUM family of algorithms are generally the best performing algorithms. Surprisingly, explicitly stacking the networks together has strong performance in half of the environments (pistonball, pong, pursuit) but is inferior to a naive application of CQL to QMIX for the other environments (reference, spread, waterworld). For the other algorithms, it is clear that the naive application of CQL to QMIX and ensembling of QMIX algorithms is inferior to our GUM algorithm across a wide range of MARL tasks. This justifies our neural architecture choices and construction empirically. We hypothesize this is due to the dependent nature of the mixer hypernetwork and the independent Q-learning networks. In the same way target networks or Polyak averaging is used to minimise the update rate across the networks. Similar approaches can be used to learn the ensemble weighting, which appears to lead to degenerative cases. The inverse weighting scheme explicitly tempers the weights directly, based on the greedy weighting scheme. We leave exploring variations of avoiding this, as an exercise for future, perhaps through this mechanism, the networks may learn to unmix themselves without the need for inverse weighing scheme.

6 Conclusion

In this report, we have examined how we can create robust reinforcement learning algorithms which cater for distributional shift, where the training environment and evaluation environment differ. We also examined cooperation and structured exploration approach in the multi-agent reinforcement learning scenario.

We have explored different approaches to build robust reinforcement learning algorithms using offline data. We have also assessed different ways in which multi-agent reinforcement learning problems can leverage its experiences in efficient ways, as well as learning to cooperate (and unlearn cooperation) to deal with threats effectively.

There are many different approaches in both single-agent and multi-agent reinforcement learning which is worth exploring concurrently.

Acknowledgements

This research is supported by ONRG research grant (N62909-19-1-2141).

References

[1] Abbas Abdolmaleki, Jost Tobias Springenberg, Yuval Tassa, Remi Munos, Nicolas Heess, and Martin Riedmiller. Maximum a posteriori policy optimisation. In *International Conference on Learning Representations*, 2018.

[2] Wendelin Boehmer, Vitaly Kurin, and Shimon Whiteson. Deep coordination graphs. In *ICML 2020: Proceedings of the Thirty-Seventh International Conference on Machine Learning*, 2020.

[3] Yongcan Cao, Wenwu Yu, Wei Ren, and Guanrong Chen. An overview of recent progress in the study of distributed multi-agent coordination. *IEEE Transactions on Industrial informatics*, 9(1):427–438, 2012.

[4] Guillaume Chaslot, Sander Bakkes, Istvan Szita, and Pieter Spronck. Monte-carlo tree search: A new framework for game ai. In *Proceedings of the Fourth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, AIIDE'08, page 216–217. AAAI Press, 2008.

[5] Filippos Christianos, Lukas Schäfer, and Stefano V Albrecht. Shared experience actor-critic for multi-agent reinforcement learning. *Advances in Neural Information Processing Systems*, 2020.

[6] Petros Christodoulou. Soft actor-critic for discrete action settings. *arXiv preprint arXiv:1910.07207*, 2019.

[7] Corinna Cortes, Xavier Gonzalvo, Vitaly Kuznetsov, Mehryar Mohri, and Scott Yang. AdaNet: Adaptive structural learning of artificial neural networks. In *International Conference on Machine Learning*, 2017.

[8] Will Dabney, Mark Rowland, Marc Bellemare, and Rémi Munos. Distributional reinforcement learning with quantile regression. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018.

[9] Jakob N Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[10] Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. In *International Conference on Machine Learning*, pages 2052–2062, 2019.

[11] Seyed Kamyar Seyed Ghasemipour, Richard Zemel, and Shixiang Gu. A divergence minimization perspective on imitation learning methods. In *Conference on Robot Learning*, pages 1259–1277. PMLR, 2020.

[12] Jayesh K Gupta, Maxim Egorov, and Mykel Kochenderfer. Cooperative multi-agent control using deep reinforcement learning. In *International Conference on Autonomous Agents and Multiagent Systems*, pages 66–83. Springer, 2017.

[13] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1861–1870, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

[14] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, page 2094–2100. AAAI Press, 2016.

[15] Maximilian Hüttenrauch, Adrian Šošić, and Gerhard Neumann. Guided deep reinforcement learning for swarm systems. *arXiv preprint arXiv:1709.06011*, 2017.

[16] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. In *International Conference on Learning Representations*, 2018.

[17] Peter Jin, Kurt Keutzer, and Sergey Levine. Regret minimization for partially observable deep reinforcement learning. In *International conference on machine learning*, pages 2342–2351. PMLR, 2018.

[18] Landon Kraemer and Bikramjit Banerjee. Multi-agent reinforcement learning as a rehearsal for decentralized planning. *Neurocomputing*, 190:82–94, 2016.

[19] Aviral Kumar, Justin Fu, Matthew Soh, George Tucker, and Sergey Levine. Stabilizing off-policy q-learning via bootstrapping error reduction. In *Advances in Neural Information Processing Systems*, pages 11784–11794, 2019.

[20] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1179–1191, 2020.

[21] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

[22] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *International Conference on Learning Representations*, 2016.

[23] Michael L. Littman. Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the Eleventh International Conference on International Conference on Machine Learning*, ICML'94, page 157–163, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc.

[24] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in Neural Information Processing Systems*, pages 6379–6390, 2017.

[25] Chris J. Maddison, Andriy Mnih, and Yee Whye Teh. The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables. In *International Conference on Learning Representations*, 2018.

[26] Anuj Mahajan, Tabish Rashid, Mikayel Samvelyan, and Shimon Whiteson. Maven: Multi-agent variational exploration. In *Advances in Neural Information Processing Systems 32*, pages 7613–7624. Curran Associates, Inc., 2019.

[27] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[28] Gerhard Neumann. Variational inference for policy search in changing situations. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ICML'11, page 817–824, Madison, WI, USA, 2011. Omnipress.

[29] Junhyuk Oh, Yijie Guo, Satinder Singh, and Honglak Lee. Self-imitation learning. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 3878–3887, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

[30] Frans A Oliehoek, Matthijs TJ Spaan, and Nikos Vlassis. Optimal and approximate q-value functions for decentralized pomdps. *Journal of Artificial Intelligence Research*, 32:289–353, 2008.

[31] Tabish Rashid, Mikayel Samvelyan, Christian Schroeder De Witt, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. Qmix: monotonic value function factorisation for deep multi-agent reinforcement learning. *International Conference on Machine Learning*, 2018.

[32] Chapman Siu. Residual networks behave like boosting algorithms. In *2019 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pages 31–40. IEEE, 2019.

[33] Chapman Siu, Jason Traish, and Richard Yi Da Xu. Dual behavior regularized reinforcement learning. *arXiv preprint arXiv:2109.09037*, 2021.

[34] Chapman Siu, Jason Traish, and Richard Yi Da Xu. Greedy unmixing for q-learning in multi-agent reinforcement learning. *arXiv preprint arXiv:2109.09034*, 2021.

[35] Chapman Siu, Jason Traish, and Richard Yi Da Xu. Regularize! don't mix: Multi-agent reinforcement learning without explicit centralized structures. *arXiv preprint arXiv:2109.09038*, 2021.

[36] Chapman Siu, Jason Traish, and Richard Yi Da Xu. Dynamic coordination graph for cooperative multi-agent reinforcement learning. *Proceedings of the 13th Asian Conference on Machine Learning*, 2021.

[37] Kyunghwan Son, Daewoo Kim, Wan Ju Kang, David Earl Hostallero, and Yung Yi. Qtran: Learning to factorize with transformation for cooperative multi-agent reinforcement learning. *International Conference on Machine Learning*, 2019.

[38] Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z Leibo, Karl Tuyls, et al. Value-decomposition networks for cooperative multi-agent learning based on team reward. *International Foundation for Autonomous Agents and Multiagent Systems*, pages 2085–2087, 2018.

[39] Ming Tan. Multi-agent reinforcement learning: independent versus cooperative agents. In *Proceedings of the Tenth International Conference on International Conference on Machine Learning*, pages 330–337. Morgan Kaufmann Publishers Inc., 1993.

[40] Justin K Terry, Benjamin Black, Mario Jayakumar, Ananth Hari, Luis Santos, Clemens Dieffendahl, Niall L Williams, Yashas Lokesh, Ryan Sullivan, Caroline Horsch, and Praveen Ravi. Pettingzoo: Gym for multi-agent reinforcement learning. *arXiv preprint arXiv:2009.14471*, 2020.

[41] Nino Vieillard, Olivier Pietquin, and Matthieu Geist. Munchausen reinforcement learning. *Advances in Neural Information Processing Systems*, 33, 2020.

[42] David H. Wolpert. Stacked generalization. *Neural Networks*, 5(2):241–259, 1992.

[43] Yifan Wu, George Tucker, and Ofir Nachum. Behavior regularized offline reinforcement learning. *arXiv preprint arXiv:1911.11361*, 2019.

[44] Meng Zhou, Ziyu Liu, Pengwei Sui, Yixuan Li, and Yuk Ying Chung. Learning implicit credit assignment for multi-agent actor-critic. *Advances in Neural Information Processing Systems*, 2020.