Stabilising Experience Replay for Deep Multi-Agent Reinforcement Learning

Jakob Foerster [*1] **Nantas Nardelli** [*1] **Gregory Farquhar** [1] **Triantafyllos Afouras** [1]
Philip. H. S. Torr [1] **Pushmeet Kohli** [2] **Shimon Whiteson** [1]

Abstract

Many real-world problems, such as network packet routing and urban traffic control, are naturally modeled as multi-agent *reinforcement learning* (RL) problems. However, existing multi-agent RL methods typically scale poorly in the problem size. Therefore, a key challenge is to translate the success of deep learning on single-agent RL to the multi-agent setting. A major stumbling block is that *independent Q-learning*, the most popular multi-agent RL method, introduces nonstationarity that makes it incompatible with the *experience replay memory* on which deep Q-learning relies. This paper proposes two methods that address this problem: 1) using a multi-agent variant of importance sampling to naturally decay obsolete data and 2) conditioning each agent's value function on a *fingerprint* that disambiguates the age of the data sampled from the replay memory. Results on a challenging decentralised variant of *StarCraft unit micromanagement* confirm that these methods enable the successful combination of experience replay with multi-agent RL.

1. Introduction

Reinforcement learning (RL), which enables an agent to learn control policies on-line given only sequences of observations and rewards, has emerged as a dominant paradigm for training autonomous systems. However, many real-world problems, such as network packet delivery (Ye et al., 2015), rubbish removal (Makar et al., 2001), and urban traffic control (Kuyer et al., 2008; Van der Pol & Oliehoek, 2016), are naturally modeled as cooperative multi-agent systems. Unfortunately, tackling such problems with traditional RL is not straightforward.

If all agents observe the true state, then we can model a cooperative multi-agent system as a single meta-agent. However, the size of this meta-agent's action space grows exponentially in the number of agents. Furthermore, it is not applicable when each agent receives different observations that may not disambiguate the state, in which case decentralised policies must be learned.

A popular alternative is *independent Q-learning* (IQL) (Tan, 1993), in which each agent independently learns its own policy, treating other agents as part of the environment. While IQL avoids the scalability problems of centralised learning, it introduces a new problem: the environment becomes nonstationary from the point of view of each agent, as it contains other agents who are themselves learning, ruling out any convergence guarantees. Fortunately, substantial empirical evidence has shown that IQL often works well in practice (Matignon et al., 2012).

Recently, the use of deep neural networks has dramatically improved the scalability of single-agent RL (Mnih et al., 2015). However, one element key to the success of such approaches is the reliance on an *experience replay memory*, which stores experience tuples that are sampled during training. Experience replay not only helps to stabilise the training of a deep neural network, it also improves sample efficiency by repeatedly reusing experience tuples. Unfortunately, the combination of experience replay with IQL appears to be problematic: the nonstationarity introduced by IQL means that the dynamics that generated the data in the agent's replay memory no longer reflect the current dynamics in which it is learning. While IQL without a replay memory can learn well despite nonstationarity so long as each agent is able to gradually track the other agents' policies, that seems hopeless with a replay memory constantly confusing the agent with obsolete experience.

To avoid this problem, previous work on deep multi-agent RL has limited the use of experience replay to short, recent buffers (Leibo et al., 2017) or simply disabled replay altogether (Foerster et al., 2016). However, these workarounds limit the sample efficiency and threaten the stability of multi-agent RL. Consequently, the incompatibility of ex-

[*]Equal contribution [1]University of Oxford, Oxford, United Kingdom [2]Microsoft Research, Redmond, USA. Correspondence to: Jakob Foerster <jakob.foerster@cs.ox.ac.uk>, Nantas Nardelli <nantas@robots.ox.ac.uk>.

Proceedings of the 34^{th} International Conference on Machine Learning, Sydney, Australia, PMLR 70, 2017.

perience replay with IQL is emerging as a key stumbling block to scaling deep multi-agent RL to complex tasks.

In this paper, we propose two approaches for effectively incorporating experience replay into multi-agent RL. The first approach interprets the experience in the replay memory as *off-environment* data (Ciosek & Whiteson, 2017). By augmenting each tuple in the replay memory with the probability of the joint action in that tuple, according to the policies in use at that time, we can compute an importance sampling correction when the tuple is later sampled for training. Since older data tends to generate lower importance weights, this approach naturally decays data as it becomes obsolete, preventing the confusion that a nonstationary replay memory would otherwise create.

The second approach is inspired by *hyper Q-learning* (Tesauro, 2003), which avoids the nonstationarity of IQL by having each agent learn a policy that conditions on an estimate of the other agents' policies inferred from observing their behaviour. While it may seem hopeless to learn Q-functions in this much larger space, especially when each agent's policy is a deep neural network, we show that doing so is feasible as each agent need only condition on a low-dimensional *fingerprint* that is sufficient to disambiguate where in the replay memory an experience tuple was sampled from.

We evaluate these methods on a decentralised variant of *StarCraft unit micromanagement*,[1] a challenging multi-agent benchmark problem with a high dimensional, stochastic environment that exceeds the complexity of many commonly used multi-agent testbeds. Our results confirm that, thanks to our proposed methods, experience replay can indeed be successfully combined with multi-agent Q-learning to allow for stable training of deep multi-agent value functions.

2. Related Work

Multi-agent RL has a rich history (Busoniu et al., 2008; Yang & Gu, 2004) but has mostly focused on tabular settings and simple environments. The most commonly used method is independent Q-learning (Tan, 1993; Shoham & Leyton-Brown, 2009; Zawadzki et al., 2014), which we discuss further in Section 3.2.

Methods like hyper Q-learning (Tesauro, 2003), also discussed in Section 3.2, and AWESOME (Conitzer & Sandholm, 2007) try to tackle nonstationarity by tracking and conditioning each agent's learning process on their teammates' current policy, while Da Silva et al. (2006) propose detecting and tracking different classes of traces on which to condition policy learning. Kok & Vlassis (2006) show that coordination can be learnt by estimating a global Q-function in the classical distributed setting supplemented with a coordination graph. In general, these techniques have so far not successfully been scaled to high-dimensional state spaces.

Lauer & Riedmiller (2000) propose a variation of distributed Q-learning, a coordination-free method. However, they also argue that the simple estimation of the value function in the standard model-free fashion is not enough to solve multi-agent problems, and coordination through means such as communication (Mataric, 1998) is required to ground separate observations to the full state function.

More recent work tries to leverage deep learning in multi-agent RL, mostly as a means to reason about the emergence of inter-agent communication. Tampuu et al. (2015) apply a framework that combines DQN with independent Q-learning to two-player pong. Foerster et al. (2016) propose DIAL, an end-to-end differentiable architecture that allows agents to learn to communicate and has since been used by Jorge et al. (2016) in a similar setting. Sukhbaatar et al. (2016) also show that it is possible to learn to communicate by backpropagation. Leibo et al. (2017) analyse the emergence of cooperation and defection when using multi-agent RL in mixed-cooperation environments such as the wolfpack problem. He et al. (2016) address multi-agent learning by explicitly marginalising the opponents' strategy using a mixture of experts in the DQN. Unlike our contributions, none of these papers directly aim to address the nonstationarity arising in multi-agent learning.

Our work is also broadly related to methods that attempt to allow for faster convergence of policy networks such as prioritized experience replay (Schaul et al., 2015), a version of the standard replay memory that biases the sampling distribution based on the TD error. However, this method does not account for nonstationary environments and does not take into account the unique properties of the multi-agent setting.

Wang et al. (2016) describe an importance sampling method for using off-policy experience in a single-agent actor-critic algorithm. However, to calculate policy-gradients, the importance ratios become products over potentially lengthy trajectories, introducing high variance that must be partially compensated for by truncation. By contrast, we address *off-environment* learning and show that the multi-agent structure results in importance ratios that are simply products over the agents' policies.

Finally, in the context of StarCraft micromanagement, Usunier et al. (2016) learn a centralised policy using standard single-agent RL. Their agent controls all the units owned by the player and observes the full state of the game.

[1] StarCraft and its expansion StarCraft: Brood War are trademarks of Blizzard Entertainment™.

By contrast, we consider a decentralised task in which each unit has only partial observability.

3. Background

We begin with background on single-agent and multi-agent reinforcement learning.

3.1. Single-Agent Reinforcement Learning

In a traditional RL problem, the agent aims to maximise its expected discounted return $R_t = \sum_{t=0}^{\infty} \gamma^t r_t$, where r_t is the reward the agent receives at time t and $\gamma \in [0, 1)$ is the discount factor (Sutton & Barto, 1998). In a fully observable setting, the agent observes the true state of the environment $s_t \in S$, and chooses an action $u_t \in U$ according to a policy $\pi(u|s)$.

The action-value function Q of a policy π is $Q^\pi(s, u) = \mathbb{E}\left[R_t | s_t = s, u_t = u\right]$. The Bellman optimality equation,

$$\begin{aligned} Q^*(s, u) &= \mathcal{T} Q^*(s, u) \\ &= r(s, u) + \gamma \sum_{s'} P(s'|s, u) \max_{u'} Q^*(s', u'), \end{aligned} \tag{1}$$

recursively represents the optimal Q-function $Q^*(s, u) = \max_\pi Q^\pi(s, u)$ as a function of the expected immediate reward $r(s, u)$ and the transition function $P(s'|s, u)$, which in turn yields an optimal greedy policy $\pi^*(u|s) = \delta(\arg\max_{u'} Q(s, u') - u)$. Q-learning (Watkins, 1989) uses a sample-based approximation of $\mathcal{T}$ to iteratively improve the Q-function. In deep Q-learning (Mnih et al., 2015), the Q-function is represented by a neural network parameterised by θ. During training, actions are chosen at each timestep according to an exploration policy, such as an ϵ-greedy policy that selects the currently estimated best action $\arg\max_u Q(s, u)$ with probability $1 - \epsilon$, and takes a random exploratory action with probability ϵ. The reward and next state are observed, and the tuple $\langle s, u, r, s' \rangle$ is stored in a *replay memory*. The parameters θ are learned by sampling batches of b transitions from the replay memory, and minimising the squared TD-error:

$$\mathcal{L}(\theta) = \sum_{i=1}^{b} [(y_i^{DQN} - Q(s, u; \theta))^2], \tag{2}$$

with a target $y_i^{DQN} = r_i + \gamma \max_{u'_i} Q(s'_i, u'_i; \theta^-)$, where θ^- are the parameters of a target network periodically copied from θ and frozen for a number of iterations. The replay memory stabilises learning, prevents the network from overfitting to recent experiences, and improves sample efficiency. In partially observable settings, agents must in general condition on their entire action-observation history, or a sufficient stastistic thereof. In deep RL, this is accomplished by modelling the Q-function with a recurrent neural network (Hausknecht & Stone, 2015), utilising a gated architecture such as LSTM (Hochreiter & Schmidhuber, 1997) or GRU (Chung et al., 2014).

3.2. Multi-Agent Reinforcement Learning

We consider a fully cooperative multi-agent setting in which n agents identified by $a \in A \equiv \{1, ..., n\}$ participate in a stochastic game, G, described by a tuple $G = \langle S, U, P, r, Z, O, n, \gamma \rangle$. The environment occupies states $s \in S$, in which, at every time step, each agent takes an action $u_a \in U$, forming a joint action $\mathbf{u} \in \mathbf{U} \equiv U^n$. State transition probabilities are defined by $P(s'|s, \mathbf{u}) : S \times \mathbf{U} \times S \to [0, 1]$. As the agents are fully cooperative, they share the same reward function $r(s, \mathbf{u}) : S \times \mathbf{U} \to \mathbb{R}$.

Each agent's observations $z \in Z$ are governed by an observation function $O(s, a) : S \times A \to Z$. For notational simplicity, this observation function is deterministic, i.e., we model only perceptual aliasing and not noise. However, extending our methods to noisy observation functions is straightforward. Each agent a conditions its behaviour on its own action-observation history $\tau_a \in T \equiv (Z \times U)^*$, according to its policy $\pi_a(u_a|\tau_a) : T \times U \to [0, 1]$. After each transition, the action u_a and new observation $O(s, a)$ are added to τ_a, forming τ'_a. We denote joint quantities over agents in bold, and joint quantities over agents other than a with the subscript $-a$, so that, e.g., $\mathbf{u} = [u_a, \mathbf{u}_{-a}]$.

In *independent Q-learning* (IQL) (Tan, 1993), the simplest and most popular approach to multi-agent RL, each agent learns its own Q-function that conditions only on the state and its own action. Since our setting is partially observable, IQL can be implemented by having each agent condition on its action-observation history, i.e., $Q_a(\tau_a, u_a)$. In deep RL, this can be achieved by having each agent perform DQN using a recurrent neural network trained on its own observations and actions.

IQL is appealing because it avoids the scalability problems of trying to learn a joint Q-function that conditions on $\mathbf{u}$, since $|\mathbf{U}|$ grows exponentially in the number of agents. It is also naturally suited to partially observable settings, since, by construction, it learns decentralised policies in which each agent's action conditions only on its own observations.

However, IQL introduces a key problem: the environment becomes nonstationary from the point of view each agent, as it contains other agents who are themselves learning, ruling out any convergence guarantees. On the one hand, the conventional wisdom is that this problem is not severe in practice, and substantial empirical results have demonstrated success with IQL (Matignon et al., 2012). On the other hand, such results do not involve deep learning.

As discussed earlier, deep RL relies heavily on experience replay and the combination of experience replay with IQL appears to be problematic: the nonstationarity introduced by IQL means that the dynamics that generated the data in the agent's replay memory no longer reflect the current dynamics in which it is learning. While IQL without a replay memory can learn well despite nonstationarity so long as each agent is able to gradually track the other agents' policies, that seems hopeless with a replay memory constantly confusing the agent with obsolete experience. In the next section, we propose methods to address this problem.

4. Methods

To avoid the difficulty of combining IQL with experience replay, previous work on deep multi-agent RL has limited the use of experience replay to short, recent buffers (Leibo et al., 2017) or simply disabled replay altogether (Foerster et al., 2016). However, these workarounds limit the sample efficiency and threaten the stability of multi-agent RL. In this section, we propose two approaches for effectively incorporating experience replay into multi-agent RL.

4.1. Multi-Agent Importance Sampling

We can address the non-stationarity present in IQL by developing an importance sampling scheme for the multi-agent setting. Just as an RL agent can use importance sampling to learn *off-policy* from data gathered when its own policy was different, so too can it learn *off-environment* (Ciosek & Whiteson, 2017) from data gathered in a different environment. Since IQL treats other agents' policies as part of the environment, off-environment importance sampling can be used to stabilise experience replay. In particular, since we know the policies of the agents at each stage of training, we know exactly the manner in which the environment is changing, and can thereby correct for it with importance weighting, as follows. We consider first a fully-observable multi-agent setting. If the Q-functions can condition directly on the true state s, we can write the Bellman optimality equation for a single agent given the policies of all other agents:

$$Q_a^*(s,u_a|\boldsymbol{\pi}_{-a}) = \sum_{\mathbf{u}_{-a}} \boldsymbol{\pi}_{-a}(\mathbf{u}_{-a}|s)\Big[r(s,u_a,\mathbf{u}_{-a}) + \gamma \sum_{s'} P(s'|s,u_a,\mathbf{u}_{-a}) \max_{u'_a} Q_a^*(s',u'_a)\Big]. \quad (3)$$

The nonstationary component of this equation is $\boldsymbol{\pi}_{-a}(\mathbf{u}_{-a}|s) = \Pi_{i\in -a}\pi_i(u_i|s)$, which changes as the other agents' policies change over time. Therefore, to enable importance sampling, at the time of collection t_c, we record $\boldsymbol{\pi}_{-a}^{t_c}(\mathbf{u}_{-a}|s)$ in the replay memory, forming an augmented transition tuple $\langle s, u_a, r, \pi(\mathbf{u}_{-a}|s), s'\rangle^{(t_c)}$.

At the time of replay t_r, we train off-environment by minimising an importance weighted loss function:

$$\mathcal{L}(\theta) = \sum_{i=1}^{b} \frac{\boldsymbol{\pi}_{-a}^{t_r}(\mathbf{u}_{-a}|s)}{\boldsymbol{\pi}_{-a}^{t_i}(\mathbf{u}_{-a}|s)}[(y_i^{DQN} - Q(s,u;\theta))^2], \quad (4)$$

where t_i is the time of collection of the i-th sample.

The derivation of the non-stationary parts of the Bellman equation in the partially observable multi-agent setting is considerably more complex as the agents' action-observation histories are correlated in a complex fashion that depends on the agents' policies as well as the transition and observation functions.

To make progress, we can define an augmented state space $\hat{s} = \{s, \boldsymbol{\tau}_{-a}\} \in \hat{S} = S \times T^{n-1}$. This state space includes both the original state s and the action-observation history of the other agents $\boldsymbol{\tau}_{-a}$. We also define a corresponding observation function $\hat{O}$ such that $\hat{O}(\hat{s},a) = O(s,a)$. With these definitions in place, we define a new reward function $\hat{r}(\hat{s},u) = \sum_{\mathbf{u}_{-a}} \boldsymbol{\pi}_{-a}(\mathbf{u}_{-a}|\boldsymbol{\tau}_{-a}) r(s,\mathbf{u})$ and a new transition function,

$$\hat{P}(\hat{s}'|\hat{s},u) = P(s',\tau'|s,\tau,u) = \sum_{\mathbf{u}_{-a}} \boldsymbol{\pi}_{-a}(\mathbf{u}_{-a}|\boldsymbol{\tau}_{-a}) P(s'|s,\mathbf{u}) p(\boldsymbol{\tau}'_{-a}|\boldsymbol{\tau}_{-a},\mathbf{u}_{-a},s'). \quad (5)$$

All other elements of the augmented game $\hat{G}$ are adopted from the original game G. This also includes T, the space of action-observation histories. The augmented game is then specified by $\hat{G} = \langle \hat{S}, U, \hat{P}, \hat{r}, Z, \hat{O}, n, \gamma\rangle$. We can now write a Bellman equation for $\hat{G}$:

$$Q(\tau,u) = \sum_{\hat{s}} p(\hat{s}|\tau)\Big[\hat{r}(\hat{s},u) + \gamma \sum_{\tau',\hat{s}',u'} \hat{P}(\hat{s}'|\hat{s},u)\pi(u',\tau')p(\tau'|\tau,\hat{s}',u)Q(\tau',u')\Big]. \quad (6)$$

Substituting back in the definitions of the quantities in $\hat{G}$, we arrive at a Bellman equation of a form similar to (3), where the righthand side is multiplied by $\boldsymbol{\pi}_{-a}(\mathbf{u}_{-a}|\boldsymbol{\tau}_{-a})$:

$$Q(\tau,u) = \sum_{\hat{s}} p(\hat{s}|\tau) \sum_{\mathbf{u}_{-a}} \boldsymbol{\pi}_{-a}(\mathbf{u}_{-a}|\boldsymbol{\tau}_{-a})\Big[r(s,\mathbf{u}) + \gamma \sum_{\tau',\hat{s}',u'} P(s'|s,\mathbf{u}) p(\boldsymbol{\tau}'_{-a}|\boldsymbol{\tau}_{-a},\mathbf{u}_{-a},s') \cdot \pi(u',\tau')p(\tau'|\tau,\hat{s}',u)Q(\tau',u')\Big]. \quad (7)$$

This construction simply allows us to demonstrate the dependence of the Bellman equation on the same nonstationary term $\boldsymbol{\pi}_{-a}(\mathbf{u}_{-a}|s)$ in the partially-observable case.

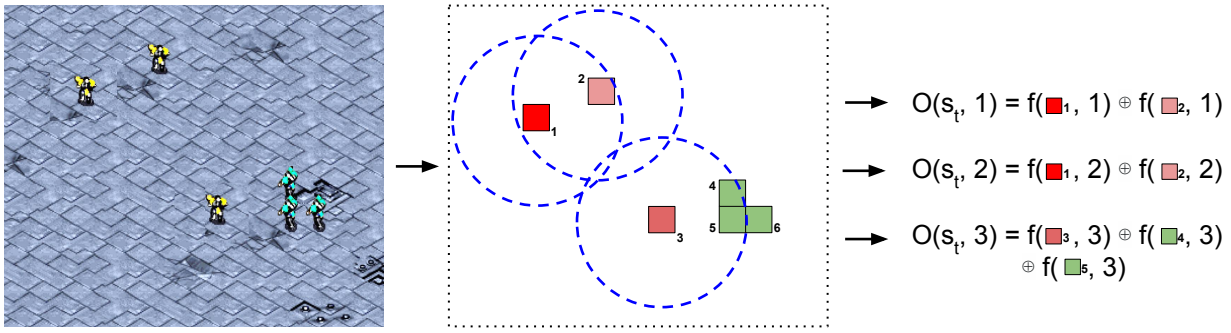


Figure 1. An example of the observations obtained by all agents at each time step t. The function f provides a set of features for each unit in the agent's field of view, which are concatenated. The feature set is {`distance`, `relative x`, `relative y`, `health points`, `weapon cooldown`}. Each quantity is normalised by its maximum possible value.

However, unlike in the fully observable case, the righthand side contains several other terms that indirectly depend on the policies of the other agents and are to the best of our knowledge intractable. Consequently, the importance ratio defined above, $\frac{\boldsymbol{\pi}^{t_r}_{-a}(\mathbf{u}_{-a}|s)}{\boldsymbol{\pi}^{t_i}_{-a}(\mathbf{u}_{-a}|s)}$, is only an approximation in the partially observable setting.

4.2. Multi-Agent Fingerprints

While importance sampling provides an unbiased estimate of the true objective, it often yields importance ratios with large and unbounded variance (Robert & Casella, 2004). Truncating or adjusting the importance weights can reduce the variance but introduces bias. Consequently, we propose an alternative method that embraces the nonstationarity of multi-agent problems, rather than correcting for it.

The weakness of IQL is that, by treating other agents as part of the environment, it ignores the fact that such agents' policies are changing over time, rendering its own Q-function nonstationary. This implies that the Q-function could be made stationary if it conditioned on the policies of the other agents. This is exactly the philosophy behind *hyper Q-learning* (Tesauro, 2003): each agent's state space is augmented with an estimate of the other agents' policies computed via Bayesian inference. Intuitively, this reduces each agent's learning problem to a standard, single-agent problem in a stationary, but much larger, environment.

The practical difficulty of hyper Q-learning is that it increases the dimensionality of the Q-function, making it potentially infeasible to learn. This problem is exacerbated in deep learning, when the other agents' policies consist of high dimensional deep neural networks. Consider a naive approach to combining hyper Q-learning with deep RL that includes the weights of the other agents' networks, $\boldsymbol{\theta}_{-a}$, in the observation function. The new observation function is then $O'(s) = \{O(s), \boldsymbol{\theta}_{-a}\}$. The agent could in principle then learn a mapping from the weights $\boldsymbol{\theta}_{-a}$, and its own trajectory τ, into expected returns. Clearly, if the other agents are using deep models, then $\boldsymbol{\theta}_{-a}$ is far too large to include as input to the Q-function.

However, a key observation is that, to stabilise experience replay, each agent does not need to be able to condition on any possible $\boldsymbol{\theta}_{-a}$, but only those values of $\boldsymbol{\theta}_{-a}$ that actually occur in its replay memory. The sequence of policies that generated the data in this buffer can be thought of as following a single, one-dimensional trajectory through the high-dimensional policy space. To stabilise experience replay, it should be sufficient if each agent's observations disambiguate where along this trajectory the current training sample originated from.

The question then, is how to design a low-dimensional *fingerprint* that contains this information. Clearly, such a fingerprint must be correlated with the true value of state-action pairs given the other agents' policies. It should typically vary smoothly over training, to allow the model to generalise across experiences in which the other agents execute policies of varying quality as they learn. An obvious candidate for inclusion in the fingerprint is the training iteration number e. One potential challenge is that after policies have converged, this requires the model to fit multiple fingerprints to the same value, making the function somewhat harder to learn and more difficult to generalise from.

Another key factor in the performance of the other agents is the rate of exploration ϵ. Typically an annealing schedule is set for ϵ such that it varies smoothly throughout training and is quite closely correlated to performance. Therefore, we further augment the input to the Q-function with ϵ, such that the observation function becomes $O'(s) = \{O(s), \epsilon, e\}$. Our results in Section 6 show that even this simple fingerprint is remarkably effective.

5. Experiments

In this section, we describe our experiments applying experience replay with fingerprints (XP+FP), with importance sampling (XP+IS), and with the combination (XP+IS+FP), to the StarCraft domain. We run experiments with both feedforward (FF) and recurrent (RNN) models, to test the hypothesis that in StarCraft recurrent models can use trajectory information to more easily disambiguate experiences from different stages of training.

5.1. Decentralised StarCraft Micromanagement

StarCraft is an example of a complex, stochastic environment whose dynamics cannot easily be simulated. This differs from standard multi-agent settings such as Packet World (Weyns et al., 2005) and simulated RoboCup (Hausknecht et al., 2016), where often entire episodes can be fully replayed and analysed. This difficulty is typical of real-world problems, and is well suited to the model-free approaches common in deep RL. In StarCraft, *micromanagement* refers to the subtask of controlling single or grouped units to move them around the map and fight enemy units. In our multi-agent variant of StarCraft micromanagement, the centralised player is replaced by a set of agents, each assigned to one unit on the map. Each agent observes a subset of the map centred on the unit it controls, as shown in Figure 1, and must select from a restricted set of durative actions: `move[direction]`, `attack[enemy_id]`, `stop`, and `noop`. During an episode, each unit is identified by a positive integer initialised on the first time-step.

All units are *Terran Marines*, ground units with a fixed range of fire about the length of four stacked units. Reward is the sum of the damage inflicted against opponent units during that timestep, with an additional terminal reward equal to the sum of the health of all units on the team. This is a variation of a naturally arising battle signal, comparable with the one used by Usunier et al. (2016). A few timesteps after the agents are spawned, they are attacked by opponent units of the same type. Opponents are controlled by the game AI, which is set to attack all the time. We consider two variations: 3 marines vs 3 marines (m3v3), and 5 marines vs 5 marines (m5v5). Both of these require the agents to coordinate their movements to get the opponents into their range of fire with good positioning, and to focus their firing on each enemy unit so as to destroy them more quickly. Skilled human StarCraft players can typically solve these tasks.

We build our models in Torch7 (Collobert et al., 2011), and run our StarCraft experiments with TorchCraft (Synnaeve et al., 2016), a library that provides the functionality to enact the standard reinforcement learning step in *StarCraft: BroodWar*, which we extend to enable multi-agent control.

5.2. Architecture

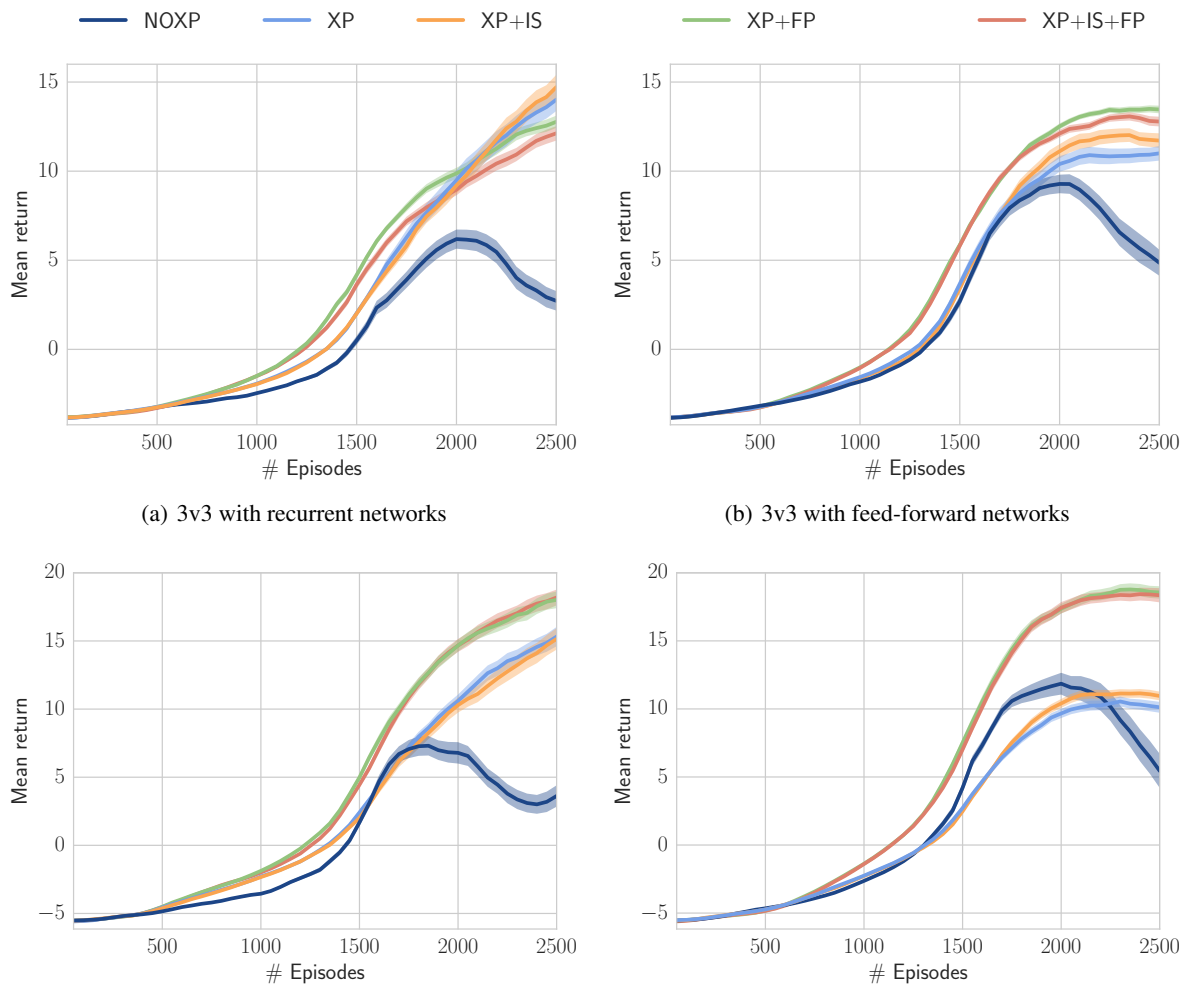
We use the recurrent DQN architecture described by Foerster et al. (2016) with a few modifications. We do not consider communicating agents, so there are no message connections. As mentioned above, we use two different different models: one with a feed-forward model with two fully connected hidden layers, and another with a single-layer GRU. For both models, every hidden layer has 128 neurons.

We linearly anneal ϵ from 1.0 to 0.02 over 1500 episodes, and train the network for $e_{max} = 2500$ training episodes. In the standard training loop, we collect a single episode and add it to the replay memory at each training step. We sample batches of $\frac{30}{n}$ episodes uniformly from the replay memory and train on fully unrolled episodes. In order to reduce the variance of the multi-agent importance weights, we clip them to the interval $[0.01, 2]$. We also normalise the importance weights by the number of agents, by raising them to the power of $\frac{1}{n-1}$. Lastly, we divide the importance weights by their running average in order to keep the overall learning rate constant. All other hyperparameters are identical to Foerster et al. (2016).

6. Results

In this section, we present the results of our StarCraft experiments, summarised in Figure 2. Across all tasks and models, the baseline without experience replay (NOXP) performs poorly. Without the diversity in trajectories provided by experience replay, NOXP overfits to the greedy policy once ϵ becomes small. When exploratory actions do occur, agents visit areas of the state space that have not had their Q-values updated for many iterations, and bootstrap off of values which have become stale or distorted by updates to the Q-function elsewhere. This effect can harm or destabilise the policy. With a recurrent model, performance simply degrades, while in the feed-forward case, it begins to drop significantly later in training. We hypothesise that full trajectories are inherently more diverse than single observations, as they include compounding chances for exploratory actions. Consequently, it is easier to overfit to single observations, and experience replay is more essential for a feed-forward model.

With a naive application of experience replay (XP), the model tries to simultaneously learn a best-response policy to every historical policy of the other agents. Despite the nonstationarity, the stability of experience replay enables XP to outperform NOXP in each case. However, due to limited disambiguating information, the model cannot appropriately account for the impact of any particular policy of the other agents, or keep track of their current policy. The experience replay is therefore used inefficiently,



(a) 3v3 with recurrent networks

(b) 3v3 with feed-forward networks

(c) 5v5 with recurrent networks

(d) 5v5 with feed-forward networks

Figure 2. Performance of our methods compared to the two baselines XP and NOXP, for both RNN and FF; (a) and (b) show the 3v3 setting, in which IS and FP are only required with feed-forward networks; (c) and (d) show the 5v5 setting, in which FP clearly improves performance over the baselines, while IS shows a small improvement only in the feedforward setting. Overall, the FP is a more effective method for resolving the nonstationarity and there is no additional benefit from combining IS with FP. Confidence intervals show one standard deviation of the sample mean.

and the model cannot generalise properly from experiences early in training.

6.1. Importance Sampling

The importance sampling approach (XP+IS) slightly outperforms XP when using feed-forward models. While mathematically sound in the fully observable case, XP+IS is only approximate for our partially observable problem, and runs into practical obstacles. Early in training, the importance weights are relatively well behaved and have low variance. However, as ϵ drops, the importance ratios become multi-modal with increasing variance. The large majority of importance weights are less than or equal to $\epsilon(1-\epsilon) \approx \epsilon$, so few experiences contribute strongly to learning. In a setting that does not require as strongly deterministic a policy as StarCraft, ϵ could be kept higher and the variance of the importance weights would be lower.

6.2. Fingerprints

Our results show that the simple fingerprint of adding e and ϵ to the observation (XP+FP) dramatically improves performance for the feed-forward model. This fingerprint provides sufficient disambiguation for the model to track the quality of the other agents' policies over the course of training, and make proper use of the experience buffer. The network still sees a diverse array of input states across which

to generalise but is able to modify its predicted value in accordance with the known stage of training.

Figure 2 also shows that there is no extra benefit from combining importance sampling with fingerprints (XP+IS+FP). This makes sense given that the two approaches both address the same problem of nonstationarity, albeit in different ways.

Figure 3, which shows the estimated value for XP+FS of a single initial state observation with different ϵ inputs, demonstrates that the network learns to smoothly vary its value estimates across different stages of training, correctly associating high values with the low ϵ seen later in training. This approach allows the model to generalise between best responses to different policies of other agents. In effect, a larger dataset is available in this case than when using importance sampling, where most experiences are strongly discounted during training. The fingerprint enables the transfer of learning between diverse historical experiences, which can significantly improve performance.

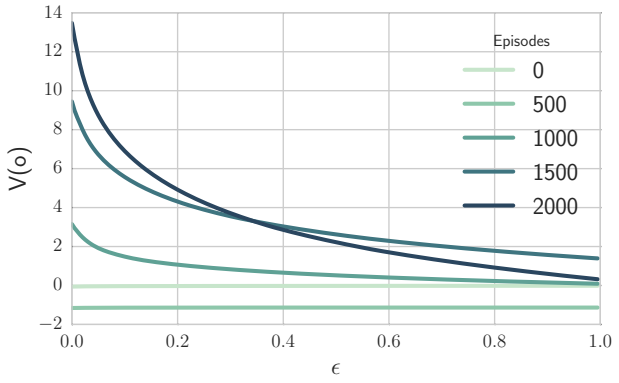


Figure 3. Estimated value of a single initial observation with different ϵ in its fingerprint input, at different stages of training. The network learns to smoothly vary its value estimates across different stages of training.

6.3. Informative Trajectories

When using recurrent networks, the performance gains of XP+IS and XP+FP are not as large; in the 3v3 task, neither method helps. The reason is that, in StarCraft, the observed trajectories are significantly informative about the state of training, as shown in Figurea 4a and 4b. For example, the agent can observe that it or its allies have taken many seemingly random actions, and infer that the sampled experience comes from early in training. This is a demonstration of the power of recurrent architectures in sequential tasks with partial observability: even without explicit additional information, the network is able to partially disambiguate experiences from different stages of training. To illustrate this, we train a linear model to predict the training ϵ from the hidden state of the recurrent model. Figure 4c shows a reasonably strong predictive accuracy even for a model trained with XP but no fingerprint, indicating that disambiguating information is indeed kept in the hidden states. However, the hidden states of a recurrent model trained with a fingerprint (Figure 4d) are even more informative.

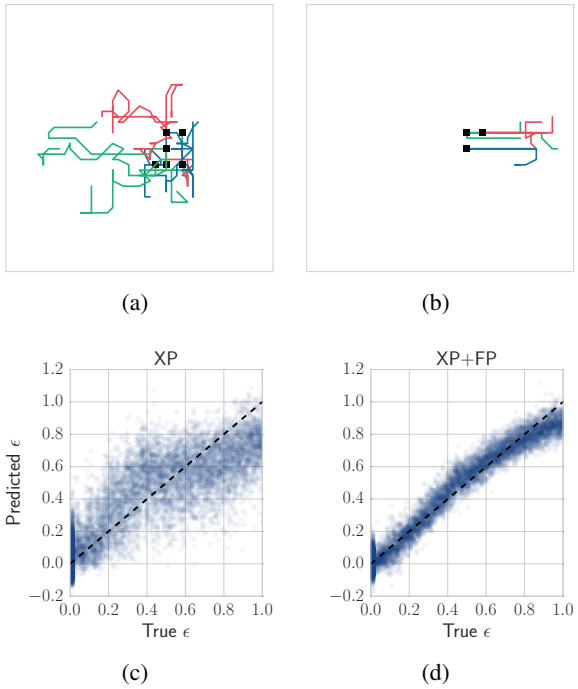


Figure 4. (upper) Sampled trajectories of agents, from the beginning (a) and end (b) of training. Each agent is one colour and the starting points are marked as black squares. (lower) Linear regression predictions of ϵ from the hidden state halfway through each episode in the replay buffer: (c) with only XP, the hidden state still contains disambiguating information drawn from the trajectories, (d) with XP+FP, the hidden state is more informative about the stage of training.

7. Conclusion

This paper proposed two methods for stabilising experience replay in deep multi-agent reinforcement learning: 1) using a multi-agent variant of importance sampling to naturally decay obsolete data and 2) conditioning each agent's value function on a fingerprint that disambiguates the age of the data sampled from the replay memory. Results on a challenging decentralised variant of StarCraft unit micromanagement confirmed that these methods enable the successful combination of experience replay with multiple agents. In the future, we would like to apply these methods to a broader range of nonstationary training problems, such as classification on changing data, and extend them to multi-agent actor-critic methods.

Acknowledgements

This project has received funding from the European Research Council (ERC) under the European Unions Horizon 2020 research and innovation programme (grant agreement #637713). This work was also supported by the Oxford-Google DeepMind Graduate Scholarship, the Microsoft Research PhD Scholarship Program, EPSRC AIMS CDT grant EP/L015987/1, ERC grant ERC-2012-AdG 321162-HELIOS, EPSRC grant Seebibyte EP/M013774/1 and EPSRC/MURI grant EP/N019474/1. Cloud computing GPU resources were provided through a Microsoft Azure for Research award. We thank Nando de Freitas, Yannis Assael, and Brendan Shillingford for the helpful comments and discussion. We also thank Gabriel Synnaeve, Zeming Lin, and the rest of the TorchCraft team at FAIR for all the help with the interface.

References

Busoniu, Lucian, Babuska, Robert, and De Schutter, Bart. A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems Man and Cybernetics Part C Applications and Reviews*, 38(2):156, 2008.

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

Ciosek, Kamil and Whiteson, Shimon. Offer: Off-environment reinforcement learning. 2017.

Collobert, R., Kavukcuoglu, K., and Farabet, C. Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, 2011.

Conitzer, Vincent and Sandholm, Tuomas. Awesome: A general multiagent learning algorithm that converges in self-play and learns a best response against stationary opponents. *Machine Learning*, 67(1-2):23–43, 2007.

Da Silva, Bruno C, Basso, Eduardo W, Bazzan, Ana LC, and Engel, Paulo M. Dealing with non-stationary environments using context detection. In *Proceedings of the 23rd international conference on Machine learning*, pp. 217–224. ACM, 2006.

Foerster, Jakob, Assael, Yannis M, de Freitas, Nando, and Whiteson, Shimon. Learning to communicate with deep multi-agent reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 2137–2145, 2016.

Hausknecht, Matthew and Stone, Peter. Deep recurrent q-learning for partially observable mdps. *arXiv preprint arXiv:1507.06527*, 2015.

Hausknecht, Matthew, Mupparaju, Prannoy, Subramanian, Sandeep, Kalyanakrishnan, S, and Stone, P. Half field offense: an environment for multiagent learning and ad hoc teamwork. In *AAMAS Adaptive Learning Agents (ALA) Workshop*, 2016.

He, He, Boyd-Graber, Jordan, Kwok, Kevin, and Daumé III, Hal. Opponent modeling in deep reinforcement learning. In *Proceedings of The 33rd International Conference on Machine Learning*, pp. 1804–1813, 2016.

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

Jorge, Emilio, Kågebäck, Mikael, and Gustavsson, Emil. Learning to play guess who? and inventing a grounded language as a consequence. *arXiv preprint arXiv:1611.03218*, 2016.

Kok, Jelle R and Vlassis, Nikos. Collaborative multiagent reinforcement learning by payoff propagation. *Journal of Machine Learning Research*, 7(Sep):1789–1828, 2006.

Kuyer, Lior, Whiteson, Shimon, Bakker, Bram, and Vlassis, Nikos. Multiagent reinforcement learning for urban traffic control using coordination graphs. In *ECML 2008: Proceedings of the Nineteenth European Conference on Machine Learning*, pp. 656–671, September 2008.

Lauer, Martin and Riedmiller, Martin. An algorithm for distributed reinforcement learning in cooperative multi-agent systems. In *In Proceedings of the Seventeenth International Conference on Machine Learning*. Citeseer, 2000.

Leibo, Joel Z, Zambaldi, Vinicius, Lanctot, Marc, Marecki, Janusz, and Graepel, Thore. Multi-agent reinforcement learning in sequential social dilemmas. *arXiv preprint arXiv:1702.03037*, 2017.

Makar, Rajbala, Mahadevan, Sridhar, and Ghavamzadeh, Mohammad. Hierarchical multi-agent reinforcement learning. In *Proceedings of the fifth international conference on Autonomous agents*, pp. 246–253. ACM, 2001.

Mataric, Maja J. Using communication to reduce locality in distributed multiagent learning. *Journal of experimental & theoretical artificial intelligence*, 10(3):357–369, 1998.

Matignon, Laetitia, Laurent, Guillaume J, and Le Fort-Piat, Nadine. Independent reinforcement learners in cooperative markov games: a survey regarding coordination problems. *The Knowledge Engineering Review*, 27(01): 1–31, 2012.

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

Robert, CP and Casella, G. Monte carlo statistical methods springer. *New York*, 2004.

Schaul, Tom, Quan, John, Antonoglou, Ioannis, and Silver, David. Prioritized experience replay. *CoRR*, abs/1511.05952, 2015.

Shoham, Y. and Leyton-Brown, K. *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press, New York, 2009.

Sukhbaatar, Sainbayar, Fergus, Rob, et al. Learning multiagent communication with backpropagation. In *Advances in Neural Information Processing Systems*, pp. 2244–2252, 2016.

Sutton, Richard S and Barto, Andrew G. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

Synnaeve, Gabriel, Nardelli, Nantas, Auvolat, Alex, Chintala, Soumith, Lacroix, Timothée, Lin, Zeming, Richoux, Florian, and Usunier, Nicolas. Torchcraft: a library for machine learning research on real-time strategy games. *arXiv preprint arXiv:1611.00625*, 2016.

Tampuu, Ardi, Matiisen, Tambet, Kodelja, Dorian, Kuzovkin, Ilya, Korjus, Kristjan, Aru, Juhan, Aru, Jaan, and Vicente, Raul. Multiagent cooperation and competition with deep reinforcement learning. *arXiv preprint arXiv:1511.08779*, 2015.

Tan, Ming. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the tenth international conference on machine learning*, pp. 330–337, 1993.

Tesauro, Gerald. Extending q-learning to general adaptive multi-agent systems. In *NIPS*, volume 4, 2003.

Usunier, Nicolas, Synnaeve, Gabriel, Lin, Zeming, and Chintala, Soumith. Episodic exploration for deep deterministic policies: An application to starcraft micromanagement tasks. *arXiv preprint arXiv:1609.02993*, 2016.

Van der Pol, Elise and Oliehoek, Frans A. Coordinated deep reinforcement learners for traffic light control. In *NIPS'16 Workshop on Learning, Inference and Control of Multi-Agent Systems*, 2016.

Wang, Ziyu, Bapst, Victor, Heess, Nicolas, Mnih, Volodymyr, Munos, Remi, Kavukcuoglu, Koray, and de Freitas, Nando. Sample efficient actor-critic with experience replay. *arXiv preprint arXiv:1611.01224*, 2016.

Watkins, Christopher John Cornish Hellaby. *Learning from delayed rewards*. PhD thesis, University of Cambridge England, 1989.

Weyns, Danny, Helleboogh, Alexander, and Holvoet, Tom. The packet-world: A test bed for investigating situated multi-agent systems. In *Software Agent-Based Applications, Platforms and Development Kits*, pp. 383–408. Springer, 2005.

Yang, Erfu and Gu, Dongbing. Multiagent reinforcement learning for multi-robot systems: A survey. Technical report, tech. rep, 2004.

Ye, Dayong, Zhang, Minjie, and Yang, Yun. A multi-agent framework for packet routing in wireless sensor networks. *sensors*, 15(5):10026–10047, 2015.

Zawadzki, E., Lipson, A., and Leyton-Brown, K. Empirically evaluating multiagent learning algorithms. *arXiv preprint 1401.8074*, 2014.