Stabilising Experience Replay for Deep Multi-Agent Reinforcement Learning

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 key 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 RL relies. This paper proposes two methods that address this problem: 1) conditioning each agent's value function on a *footprint* that disambiguates the age of the data sampled from the replay memory and 2) using a multi-agent variant of importance sampling to naturally decay obsolete data. 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 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 experience 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 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.

The second 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.

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 vastly exceeds the complexity of most commonly used multi-agent testbeds. Our results confirm that, thanks to our proposed methods, experience replay can indeed be successfully combined with multi-agent RL to allow for stable training of deep multi-agent RL.

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) - which we also discuss in Section 3.2 - and AWESOME (Conitzer & Sandholm, 2007) try to tackle the nonstationarity by tracking and conditioning each agents' 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) demonstrate a variation of the coordination-free method called distributed Q-learning. 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, a end-to-end differentiable architecture that allows agents to learn to communicate and has since 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. 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 non-stationary environments, and does not take advantage of 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 a truncation. Instead 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) demonstrated some success learning a centralised policy using standard single-agent RL. This agent is able to control all the units owned by the player, and observes the full state of the game in some parametrised form. 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 $\gamma \in [0, 1)$

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

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 $Q^\pi(s,u) = \mathbb{E}[R_t|s_t = s, u_t = u]$. The Bellman optimality operator $\mathcal{T}Q(s,u) = \mathbb{E}_{s'}[r + \gamma \max_{u'} Q(s',u')]$ is a contraction operator in supremum norm with a unique fixed point, the optimal Q-function $Q^*(s,u) = \max_\pi Q^\pi(s,u)$, which in turn yields the optimal greedy policy $\pi^*(s,u) = \delta(\arg\max_{u'} Q(s,u') - u)$. 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 parametrised 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 $<s,u,r,s'>$ 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:

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

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 $G = <S,U,P,r,Z,O,n,\gamma>$. 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 RL with Fingerprints

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, of course, that the dimensionality of the Q-function has increased, 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. Note that the two factors that change the expected return for a single agent's policy are the rate of exploration and the quality of the greedy policy of the other agents. These variables change smoothly over the course of training, which implies that it should be sufficient for the fingerprint to consist simply of the episode number of the training sample. However, adding this information to the observation is not straightforward, since the collection time clearly increases monotonically over the course of training. If we simply input the episode number as a scalar, the neural network has to continuously work with inputs that are outside the range of the previous training data, leading to instability. Therfore, we must design a representation that encodes this information such that early episodes are drawn from a distribution similar to those of late episodes. In addition, to aid generalisation, the Euclidean distance between the fingerprints of similar episodes should be small.

To meet these requirements, we propose a fingerprint that consists of a binary vector that is sampled uniformly at the beginning of training. To create the temporal correlation and include temporal structure in the fingerprint, a single randomly chosen bit in the vector is flipped periodically after a fixed number of episodes, as illustrated in Figure 1. The observation function thus becomes: $O'(s) = \{O(s), F(e)\}$, where $F(e)$ is the fingerprint associated with episode e.

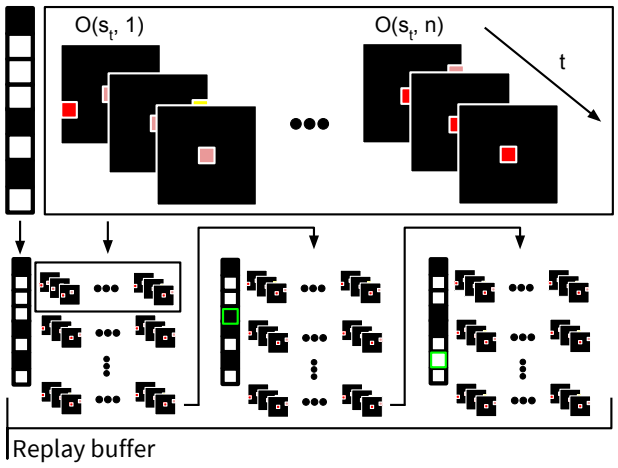

Figure 1. Multi-agent RL with fingerprints: squares highlighted in green correspond to randomly switched bits in the fingerprint, which is updated at a fixed rate across episodes. During training, the replay buffer is filled with recorded episodes and their corresponding fingerprints.

4.2. Multi-Agent Importance Sampling

Instead of preventing IQL's nonstationarity by augmenting each agent's input with a fingerprint, we can instead correct for it 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 corrections can be used to stabilise experience replay. For simplicity, consider first a fully-observable multi-agent setting. Since Q-functions can now 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:

$$Q_a^*(s, u_a|\boldsymbol{\pi}_{-a}) = \sum_{\mathbf{u}_{-a}} \boldsymbol{\pi}_{-a}(\mathbf{u}_{-a}|s)\Big[r(u_a, \mathbf{u}_{-a}, s)+ \gamma \sum_{s'} P(s'|s, u_a, \mathbf{u}_{-a}) \max_{u'_a} Q_a^*(s', u'_a)\Big]. \quad (2)$$

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 $<s, u_a, r, \pi(\mathbf{u}_{-a}|s), s'>^{(t_c)}$.

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

$$\mathcal{L}(\theta) = \sum_{i=1}^{b} \frac{\boldsymbol{\pi}_{-a}^{t_r}(\mathbf{u}_{-a}|s)}{\boldsymbol{\pi}_{-a}^{t_i}(\mathbf{u}_{-a}|s)}[(y_i^{DQN} - Q(s, u; \theta))^2], \quad (3)$$

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

The derivation of the importance weights 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, $\hat{O}$ s.t. $\hat{O}(\hat{s}, a) = O(s, a)$. With these definitions in place we define a new reward function $\hat{r}(\hat{s}, u) = \sum_{\mathbf{u}_{-a}} \boldsymbol{\pi}_{-a}(\mathbf{u}_{-a}|\boldsymbol{\tau}_{-a})r(s, \mathbf{u})$.

Lastly we define a new transition function,

$$\hat{P}(\hat{s}'|\hat{s}, u) = P(s', \tau'|s, \tau, u) = \sum_{\mathbf{u}_{-a}} \boldsymbol{\pi}_{-a}(\mathbf{u}_{-a}|\boldsymbol{\tau}_{-a})P(s'|s, \mathbf{u})p(\boldsymbol{\tau}'_{-a}|\boldsymbol{\tau}_{-a}, \mathbf{u}_{-a}, s') \quad (4)$$

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 $\hat{G} = <\hat{S}, U, \hat{P}, \hat{r}, Z, \hat{O}, n, \gamma>$.

With these definitions we can write a Bellman equation for the above defined $\hat{G}$:

$$Q(\tau, u) = \sum_{\hat{s}} p(\hat{s}|\tau)\Big[\hat{r}(\hat{s}, u) + \gamma \sum_{\tau', \hat{s}', u'} \hat{P}(\hat{s}'|\hat{s}, u)\pi(u', \tau')p(\tau'|\tau, \hat{s}', u)Q(\tau', u')\Big] \quad (5)$$

Substituting back in the definitions of the quantities in $\hat{G}$, we arrive at a Bellman equation of a form similar to Equation2, where $\boldsymbol{\pi}_{-a}(\mathbf{u}_{-a}|\boldsymbol{\tau}_{-a})$ multiplies right hand side:

$$Q(\tau, u) = \sum_{\hat{s}} p(\hat{s}|\tau) \sum_{\mathbf{u}_{-a}} \boldsymbol{\pi}_{-a}(\mathbf{u}_{-a}|\boldsymbol{\tau}_{-a})\Big[r(s, \mathbf{u}) + \gamma \sum_{\tau', \hat{s}', u'} P(s'|s, \mathbf{u})p(\boldsymbol{\tau}'_{-a}|\boldsymbol{\tau}_{-a}, \mathbf{u}_{-a}, s') \cdot \pi(u', \tau')p(\tau'|\tau, \hat{s}', u)Q(\tau', u')\Big] \quad (6)$$

We note that unlike in the fully observable case, the right hand side contains several terms that indirectly depend on the policies of the other agents and are to the best of our knowledge intractable. However, we show empirically in the next section that the importance ratio defined above, $\frac{\boldsymbol{\pi}_{-a}^{t_r}(\mathbf{u}_{-a}|s)}{\boldsymbol{\pi}_{-a}^{t_i}(\mathbf{u}_{-a}|s)}$, which is only an approximation in the partially observable setting, nonetheless works well in practice.

5. Experiments

In this section, we describe our experiments applying experience replay with fingerprints (XP-FP) and with importance sampling (XP-IS) to the StarCraft domain. Since FP requires the network to be able to represent Q-values for a range of different policies of the other agents, we hypothesise that IS will outperform it when the network's representational capacity is sufficiently limited. We also investigate whether, in some settings and given enough recurrent capacity, the agent can infer the episode of a given training sample from its action-observation history alone, without requiring a fingerprint. To test these hypotheses, we need to be able to manipulate the informativeness of the action-observation histories and the capacity of the RNN. As a proxy for the capacity of the RNN, we vary the number of neurons in its hidden state between a 'small' and 'large' setting, of 16 and 128 units respectively. Furthermore we manipulate the informativeness of τ by changing the number of unroll steps between two settings. In 'fully unrolled' the RNN is unrolled over the entire episode during training, while in 'short unroll' hidden states are only passed for 10 steps.

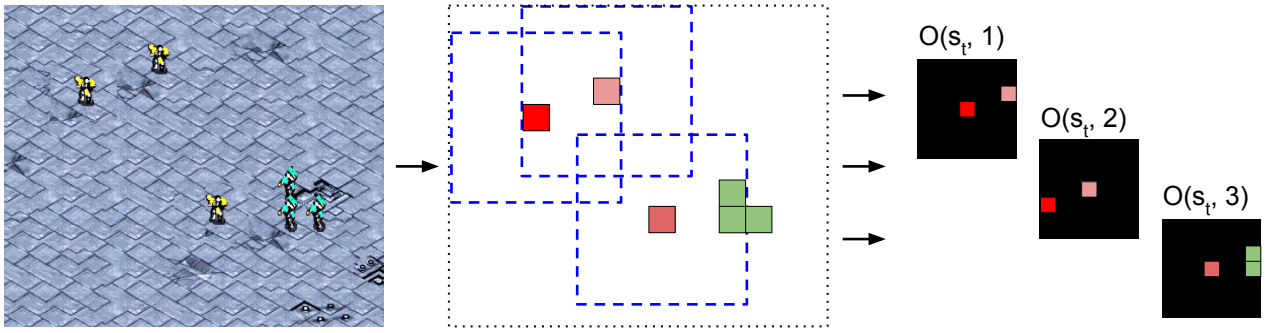

Figure 2. An example of the observations obtained by all agents at each time step t. Square fields of view are centered on each agent and not shared across the team. Each team member is assigned an id (in this figure shown as a different intensity of red).

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 square subset of the map centred on the unit it controls, as shown in Figure 2, 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 step, which the agents observe on the map, and by the number of health points it has, which is not directly observable by other units. All units are *Terran Marines*, ground units with a fixed range of fire about the length of 4 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, the agents 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. We note that skilled StarCraft players can typically solve these tasks.

5.2. Architecture

We use the recurrent DQN architecture described by Foerster et al. (2016) with a few modifications. Since we do not consider communicating agents, the number of bits in the message is set to 0. As mentioned above, we use two different sizes for the hidden state of the RNN, a 'small setting' of 16 units and a 'large setting' of 128. We begin training with $e_r = 50$ fully random exploratory episodes, after which we employ an ϵ annealing schedule of $1/(1 + \sqrt{e - e_r})$, with an exponential smoothing window of 20 episodes. We train the network for $e_{max} = 600$ training episodes. In the standard training loop we collect 5 full episodes and add them to the replay memory at each training step. During standard training we sample uniformly from the replay memory and do full unrolling of the episodes. In 'short unroll' experiments we only unroll for 10 steps, from an initial hidden state set to zeros. We also introduce a convolutional neural network to process the 2D representation of the input and a fully connected linear layer to process the fingerprint when used. In order to bound the variance of the multi-agent importance weights, we apply clipping in the range $[0.01, 2]$. All other hyperparameters are identical to Foerster et al. (2016).

6. Results

In this section, we present the results of our StarCraft experiments.

6.1. Disambiguate and Represent

Figure 3a shows that a small sized RNN (16 units) performs poorly in the 3 v 3 task using vanilla experience replay (XP) and without experience replay (NOXP). In particular, during the second half of training, the returns drop both for XP

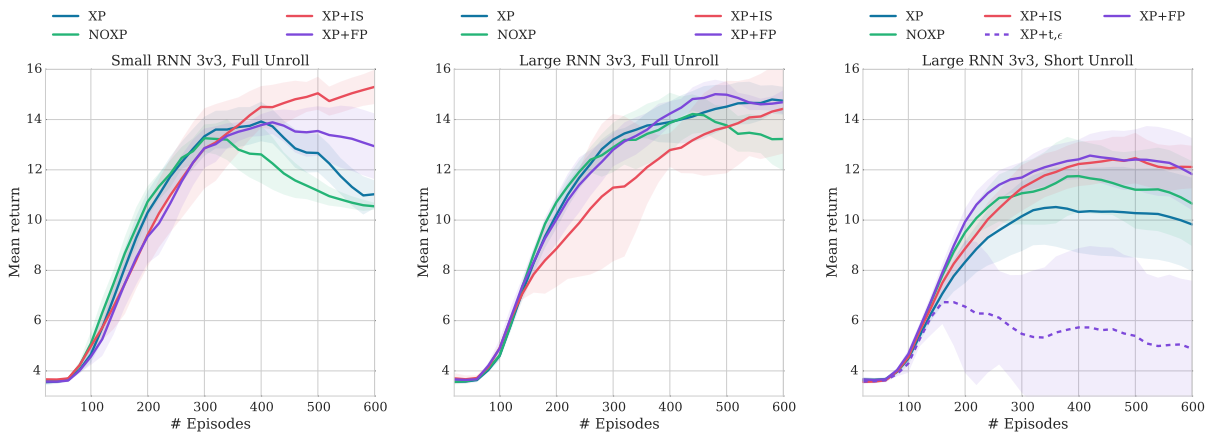

Figure 3. a) Performance of XP+FP and XP+IS on the 3 v 3 task, compared to the two baselines XP and NOXP, with a small RNN. Since the performance of the baselines is capacity limited,XP+IS is more effective than XP+FP; b) increasing the RNN size to 128 units alleviates the capacity limitation, allowing the RNN to represent Q-values for different points in history; c) reducing the unrolling to 10 time steps limits the RNN's ability to disambiguate trajectories, so XP+FP and XP+IS both improve the performance. Also shown is the naive implementation of fingerprinting, XP+t,ϵ.

and NOXP. Without experience replay, the model overfits to the greedy policy when ϵ has dropped sufficiently low. With XP, the model unsuccessfully tries to simultaneously learn a best response policy to every historical policy of the other agents. This drop for XP can be accounted for by either the limited disambiguating information, or limited *representational capacity*, available to the model. However, we also note that in our particular problem the sampled action-observation history is highly informative about the policies of the other agents, as can be seen in Figure 5. 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. Consequently, we hypothesise that the performance of the small RNN on this task is in fact capacity limited.

Figure 3b shows that by increasing the RNN size to 128 units, it is able to successfully train even with vanilla XP, resulting in returns of around 14 points. This perhaps surprising result demonstrates the power of recurrent architectures in sequential tasks with partial observability. This confirms that by observing its own action-observation history and the behaviour of the other agents, the model can distinguish between policies at different stages of training without any additional information. This is particularly the case in our StarCraft micromanagement scenarios as the units are close to each other at their starting positions, when compared to their visibility range. As a consequence they can observe a large portion of each others' trajectories.

We next test whether we can limit the ability of the RNN to disambiguate episodes by reducing the informativeness of the action-observation histories. Figure 3c shows the performance of the large RNN in the 'short unroll' setting, where the network is unrolled for only 10 time-steps during training. We observe that the performance of the RNN drops to under 10 points, similar to the performance of the XP and NOXP baselines in Figure 3a.

In order to confirm whether the performance in this 'short unroll' case is limited by the ability to disambiguate experiences at different stages in training, we must examine whether fingerprints can help recover performance. To do so, we can look at Figure 3c, which shows that by adding fingerprints, XP+FP partially recovers performance on the task compared to the XP baseline, as it helps the RNN to disambiguate experiences drawn from different environments, i.e., different policies of the other agents. By contrast, in Figure 3a, the RNN is *capacity limited* and the fingerprints are less useful, as evidenced by the low performance of XP+FP.

We also compare against a naive baseline method, XP + t, ϵ, which replaces the fingerprint, $F(e)$, with a two-component vector containing a scaled version of the episode number and the value of ϵ at the time of collection, i.e. $F_{\text{naive}}(e) = \{e/e_{max}, \epsilon(e)\}$. Figure 3c shows that this naive method leads to instability in training, rather than improving performance. Unlike XP+FP, XP+IS attempts to compensate for the nonstationarity of samples in the replay buffer. As a consequence, it should be advantageous both when the RNN is capacity limited and when it is information limited. Figure 3a shows that XP+IS improves the performance of a capacity limited RNN (16 bits), while

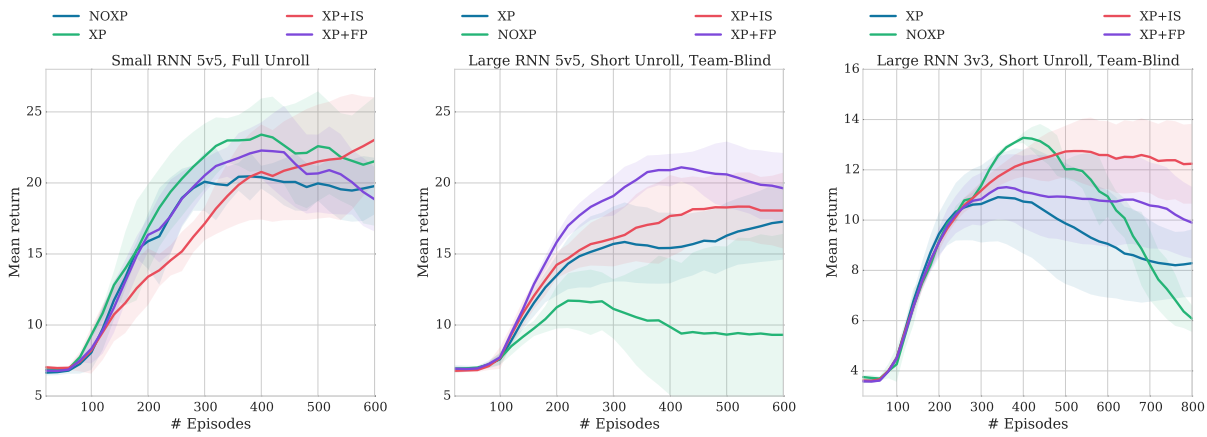

Figure 4. a) Performance of all four methods on the 5v5 problem. Due to the large amount of information contained a given observation, vanilla XP performs on par with XP+IS and XP+FP b) by doing short unrolling of the RNN and introducing 'team blindness' we can limit both the number of observations and the information in a given observation. XP now fails to disambiguate between episodes from different parts of the training process. Since it is limited by information content, including the fingerprint restores some of the performance. c) for the 3v3 task with short unrolling and 'team blindness', both the fingerprint and IS improve training.

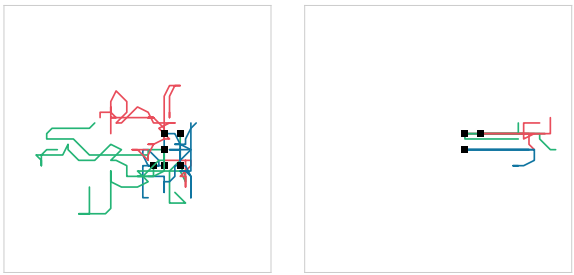

Figure 5. Shown are sampled trajectories of agents for the 3v3 task, a) from the beginning of training, b) from the end of training. Clearly the trajectories are characteristic of the stage of training. Each agent is one colour and the starting points are marked as black squares.

in Figure 3c, XP+IS improves the performance of the information limited RNN.

In figure Figure 4 we show results for a 5 vs 5 battle, increasing the number of agents. In contrast to Figure 3a, Figure 4a shows all methods perform quite well in this setting. We hypothesise that with more agents and the relatively broad field of view, it becomes possible to disambiguate between different episodes from even a single observation, allowing the CNN component of the network to compensate for the limited RNN capacity. For example, if the agents are scattered aimlessly at one point in time, they are likely to be employing poor policies - the network can use this information to accurately predict low returns. We test this hypothesis by giving the agents 'team blindness,' which removes the positions of their allies from their observations, and limiting the training to short unrolls, such that the agents can learn little from even their own τ. This case is shown in figure Figure 4b, where XP+FP improves over the other methods. By processing information correlated to the collection time, the network now manages to learn effectively using the replay memory. Finally, we test the same scenario in the 3v3 case, shown in Figure 4c. Compared to Figure 3c, the further restriction of information about the policies of other agents (team blindness), leads to a greater out-performance by our methods over XP / NOXP.

7. Conclusion

This paper presented two methods for stabilising experience replay in deep multi-agent reinforcement learning. We showed that using *fingerprints* can help agents to disambiguate between episodes from different parts of the training process and thereby partially recover performance in the face of nonstationarity. Furthermore, we showed that a variant of importance sampling applied to the multi-agent setting with replay memory recovers most of the performance which is lost due to nonstationary training samples. We also presented ablation studies on the input to and architecture of the RNN to illustrate the importance of disambiguating between episodes from different stages of training when using recurrent learning in nonstationary environments.

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.

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.

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.

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.

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.