CORRECTING EXPERIENCE REPLAY FOR MULTI-AGENT COMMUNICATION

Anonymous authors
Paper under double-blind review

ABSTRACT

We consider the problem of learning to communicate using multi-agent reinforcement learning (MARL). A common approach is to learn off-policy, using data sampled from a replay buffer. However, messages received in the past may not accurately reflect the current communication policy of each agent, and this complicates learning. We therefore introduce a 'communication correction' which accounts for the non-stationarity of observed communication induced by multi-agent learning. It works by relabelling the received message to make it likely under the communicator's current policy, and thus be a better reflection of the receiver's current environment. To account for cases in which agents are both senders and receivers, we introduce an ordered relabelling scheme. Our correction is computationally efficient and can be integrated with a range of off-policy algorithms. It substantially improves the ability of communicating MARL systems to learn across a variety of cooperative and competitive tasks.

1 INTRODUCTION

Since the introduction of deep Q-learning (Mnih et al., 2013), it has become very common to use previous online experience, for instance stored in a replay buffer, to train agents in an offline manner. An obvious difficulty with doing this is that the information concerned may be out of date, leading the agent woefully astray in cases where the environment of an agent changes over time. One obvious strategy is to discard old experiences. However, this is wasteful – it requires many more samples from the environment before adequate policies can be learned, and may prevent agents from leveraging past experience sufficiently to act in complex environments. Here, we consider an alternative, Orwellian possibility, of using present information to correct the past, showing that it can greatly improve an agent's ability to learn.

We explore a paradigm case involving multiple agents that must learn to communicate to optimise their own or task-related objectives. As with deep Q-learning, modern model-free approaches often seek to learn this communication off-policy, using experience stored in a replay buffer (Foerster et al., 2016; 2017; Lowe et al., 2017; Peng et al., 2017). However, multi-agent reinforcement learning (MARL) can be particularly challenging as the underlying game-theoretic structure is well known to lead to non-stationarity, with past experience becoming obsolete as agents come progressively to use different communication codes. It is this that our correction addresses.

Altering previously communicated messages is particularly convenient for our purposes as it has no direct effect on the actual state of the environment (Lowe et al., 2019), but a quantifiable effect on the observed message, which constitutes the receiver's 'social environment'. We can therefore determine what the received message *would be* under the communicator's current policy, rather than what it was when the experience was first generated. Once this is determined, we can simply relabel the past experience to better reflect the agent's current social environment, a form of off-environment correction (Ciosek & Whiteson, 2017).

We apply our 'communication correction' using the framework of centralised training with decentralised control (Lowe et al., 2017; Foerster et al., 2018), in which extra information – in this case the policies and observations of other agents – is used during training to learn decentralised multi-agent policies. We show how it can be combined with existing off-policy algorithms, with little computational cost, to achieve strong performance in both the cooperative and competitive cases.

2 BACKGROUND

Markov Games A partially observable Markov game (POMG) (Littman, 1994; Hu et al., 1998) for N agents is defined by a set of states $\mathcal{S}$, sets of actions $\mathcal{A}_1, ..., \mathcal{A}_N$ and observations $\mathcal{O}_1, ..., \mathcal{O}_N$ for each agent. In general, the stochastic policy of agent i may depend on the set of action-observation histories $H_i \equiv (\mathcal{O}_i \times \mathcal{A}_i)^*$ such that $\pi_i : \mathcal{H}_i \times \mathcal{A}_i \to [0, 1]$. In this work we restrict ourselves to history-independent stochastic policies $\pi_i : \mathcal{O}_i \times \mathcal{A}_i \to [0, 1]$. The next state is generated according to the state transition function $\mathcal{P} : \mathcal{S} \times \mathcal{A}_1 \times \ldots \times \mathcal{A}_n \times \mathcal{S} \to [0, 1]$. Each agent i obtains deterministic rewards defined as $r_i : \mathcal{S} \times \mathcal{A}_1 \times ... \times \mathcal{A}_n \to \mathbb{R}$ and receives a deterministic private observation $o_i : \mathcal{S} \to \mathcal{O}_i$. There is an initial state distribution $\rho_0 : \mathcal{S} \to [0, 1]$ and each agent i aims to maximise its own discounted sum of future rewards $\mathbb{E}_{s\sim\rho_{\boldsymbol{\pi}},\boldsymbol{a}\sim\boldsymbol{\pi}}[\sum_{t=0}^{\infty} \gamma^t r_i(s, \boldsymbol{a})]$ where $\boldsymbol{\pi} = \{\pi_1, \ldots, \pi_n\}$ is the set of policies for all agents, $\boldsymbol{a} = (a_1, \ldots, a_N)$ is the joint action and $\rho_{\boldsymbol{\pi}}$ is the discounted state distribution induced by these policies starting from ρ_0.

Experience Replay As an agent continually interacts with its environment it receives experiences $(s_t, a_t, r_{t+1}, s_{t+1})$ at each time step. However, rather than using those experiences immediately for learning, it is possible to store such experience in a replay buffer, $\mathcal{D}$, and sample them at a later point in time for learning (Mnih et al., 2013). This breaks the correlation between samples, reducing the variance of updates and the potential to overfit to recent experience. In the single-agent case, prioritising samples from the replay buffer according to the temporal-difference error has been shown to be effective (Schaul et al., 2015). In the multi-agent case, Foerster et al. (2017) showed that issues of non-stationarity could be partially alleviated for independent Q-learners by importance sampling and use of a low-dimensional 'fingerprint' such as the training iteration number.

MADDPG Our method can be combined with a variety of algorithms, but we commonly employ it with multi-agent deep deterministic policy gradients (MADDPG) (Lowe et al., 2017), which we describe here. MADDPG is an algorithm for centralised training and decentralised control of multi-agent systems (Lowe et al., 2017; Foerster et al., 2018), in which extra information is used to train each agent's critic in simulation, whilst keeping policies decentralised such that they can be deployed outside of simulation. It uses deterministic polices, as in DDPG (Lillicrap et al., 2015), which condition only on each agent's local observations and actions. MADDPG handles the non-stationarity associated with the simultaneous adaptation of all the agents by introducing a separate centralised critic $Q_i^{\boldsymbol{\mu}}(\boldsymbol{o}, \boldsymbol{a})$ for each agent where $\boldsymbol{\mu}$ corresponds to the set of deterministic policies $\mu_i : \mathcal{O} \to \mathcal{A}$ of all agents. Here we have denoted the vector of joint observations for all agents as $\boldsymbol{o}$.

The multi-agent policy gradient for policy parameters θ of agent i is:

$$\nabla_{\theta_i} J(\theta_i) = \mathbb{E}_{\boldsymbol{o},\boldsymbol{a}\sim\mathcal{D}}[\nabla_{\theta_i}\mu_i(o_i)\nabla_{a_i} Q_i^{\boldsymbol{\mu}}(\boldsymbol{o}, \boldsymbol{a})|_{a_i=\mu_i(o_i)}]. \tag{1}$$

where $\mathcal{D}$ is the experience replay buffer which contains the tuples $(\boldsymbol{o}, \boldsymbol{a}, \boldsymbol{r}, \boldsymbol{o'})$. Like DDPG, each $Q_i^{\boldsymbol{\mu}}$ is approximated by a critic Q_i^w which is updated to minimise the error with the target.

$$\mathcal{L}(w_i) = \mathbb{E}_{\boldsymbol{o},\boldsymbol{a},\boldsymbol{r},\boldsymbol{o'}\sim\mathcal{D}}[(Q_i^w(\boldsymbol{o}, \boldsymbol{a}) - y)^2] \tag{2}$$

where $y = r_i + \gamma Q_i^w(\boldsymbol{o'}, \boldsymbol{a'})$ is evaluated for the next state and action, as stored in the replay buffer.

Communication One way to classify communication is whether it is explicit or implicit. Implicit communication involves transmitting information by changing the shared environment (e.g., scattering breadcrumbs). By contrast, explicit communication can be modelled as being separate from the environment, only affecting the observations of other agents. In this work, we focus on explicit communication with the expectation that dedicated communication channels will be frequently integrated into artificial multi-agent systems such as driverless cars.

Although explicit communication does not formally alter the environmental state, it does change the observations of the receiving agents, a change to what we call its 'social environment'[1]. For agents which act in the environment and communicate simultaneously, the set of actions for each agent $\mathcal{A}_i = \mathcal{A}_i^e \times \mathcal{A}_i^m$ is the Cartesian product of the sets of regular environment actions $\mathcal{A}_i^e$ and explicit communication actions $\mathcal{A}_i^m$. Similarly, the set of observations for each receiving agent

[1] which could also by described as a change to each agent's belief state.

$\mathcal{O}_i = \mathcal{O}_i^e \times \mathcal{O}_i^m$ is the Cartesian product of the sets of regular environmental observations $\mathcal{O}_i^e$ and explicit communication observations $\mathcal{O}_i^m$. Communication may be targeted to specific agents or broadcast to all agents and may be costly or free. The zero cost formulation is commonly used and is known as 'cheap talk' in the game theory community.

In many multi-agent simulators the explicit communication action is related to the observed communication in a simple way, for example being transmitted to the targeted agent with or without noise on the next time step. Similarly, real world systems may transmit communication in a well understood way, such that the observed message can be accurately predicted given the sent message (particularly if error-correction is used). By contrast, the effect of environment actions is generally difficult to predict, as the shared environment state will typically exhibit more complex dependencies.

3 METHODS

Our general starting point is to consider how explicit communication actions and observed messages might be relabelled using an explicit communication model. This model often takes a simple form, such as depending only on what was communicated on the previous timestep. The observed messages $\boldsymbol{o}_{t+1}^m$ given communication actions $\boldsymbol{a}_t^m$ are therefore samples from:

$$\boldsymbol{o}_{t+1}^m \sim p(\boldsymbol{o}_{t+1}^m \mid \boldsymbol{a}_t^m) \tag{3}$$

Examples of such a communication model could be an agent i receiving a noiseless message from a single agent j such that $o_{i,t+1}^m = a_{j,t}^m$, or receiving the message corrupted by Gaussian noise $o_{i,t+1}^m \sim \mathcal{N}(a_{j,t}^m, \sigma)$ where σ is a variance parameter. We consider the noise-free case in the multi-agent simulator in our experiments, although the general idea can be applied to more complex, noisy communication models.

A communication model such as this allows us to correct past actions and observations in a consistent way. To understand how this is possible, we consider a sample from a multi-agent replay buffer which is used for off-policy learning. In general, the multi-agent system at current time t' receives observations $\boldsymbol{o}_{t'}$, collectively takes actions $\boldsymbol{a}_{t'}$ using the decentralised policies $\boldsymbol{\pi}$, receives rewards $\boldsymbol{r}_{t'+1}$ and the next observations $\boldsymbol{o}_{t'+1}$. These experiences are stored as a tuple in the replay buffer for later use to update the multi-agent critic(s) and policies. For communicating agents, we can describe a sample from the replay buffer $\mathcal{D}$ as the tuple:

$$(\boldsymbol{o}_t^e, \boldsymbol{o}_t^m, \boldsymbol{a}_t^e, \boldsymbol{a}_t^m, \boldsymbol{r}_{t+1}^e, \boldsymbol{r}_{t+1}^m, \boldsymbol{o}_{t+1}^e, \boldsymbol{o}_{t+1}^m) \sim \mathcal{D} \tag{4}$$

where we separately denote environmental (e) and communication (m) terms, and t indexes a time in the past (rather than the current time t'). For convenience we can ignore the environmental tuple of observations, actions and reward as we do not alter these, and consider only the communication tuple $(\boldsymbol{o}_t^m, \boldsymbol{a}_t^m, \boldsymbol{r}_{t+1}^m, \boldsymbol{o}_{t+1}^m)$. Using the communication model at time t', we can relate a change in $\boldsymbol{a}_t^m$ to a change in $\boldsymbol{o}_{t+1}^m$. If we also keep track of $\boldsymbol{a}_{t-1}^m$ we can similarly change $\boldsymbol{o}_t^m$. In our experiments we assume for simplicity that communication is costless (the 'cheap talk' setting), which means that $\boldsymbol{r}_{t+1}^m = 0$, however in general we could also relabel rewards using a model of communication cost $p(\boldsymbol{r}_{t+1}^m \mid \boldsymbol{a}_t^m)$. Equipped with an ability to rewrite history, we next consider how to use it, to improve multi-agent learning.

3.1 OFF-ENVIRONMENT RELABELLING

A useful perspective for determining how to relabel samples is to consider each multi-agent experience tuple separately, from the perspective of each agent, rather than as a single tuple received by all agents as is commonly assumed. For a given agent's tuple, we can examine all the observed messages which constitutes its social environment (including even messages sent to other agents, which will be seen by a centralised critic). These were generated by past policies of other agents, and since then these policies may have changed due to learning or changes in exploration. Our first approach is therefore to relabel the communication tuple $(\boldsymbol{o}_t^m, \boldsymbol{a}_t^m, \boldsymbol{r}_{t+1}^m, \boldsymbol{o}_{t+1}^m)_i$ for agent i by instead querying the current policies of other agents, replacing the communication actions accordingly and using the transition model to compute the new observed messages. For agent i this procedure is:

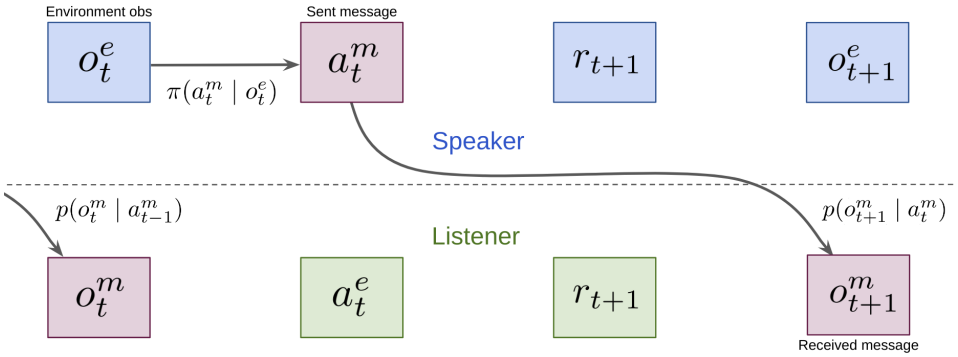

Figure 1: Consider a multi-agent experience tuple for a Listener agent receiving communication from a Speaker agent. In this simplified illustration the Speaker agent receives only environment observations, the Listener only receives communication. Our communication correction relabels the Listener's experience by generating a new message using the Speaker's current policy $\pi(a_t^m|o_t^e)$ and then generating the new Listener observation using the communication model $p(o_{t+1}^m|a_t^m)$. We shade in red the parts of the experience tuple which we relabel. Note that this relabelling only takes place for the Listener's sampled multi-agent experience, and not for the Speaker, as in this example the Speaker is not itself a Listener.

$$\begin{aligned} \hat{\boldsymbol{a}}_{\neg i,t}^m &\sim \boldsymbol{\pi}_{\neg i}(\boldsymbol{a}_{\neg i,t}^m \mid \boldsymbol{o}_{\neg i,t}) \\ \hat{o}_{i,t+1}^m &\sim p(o_{i,t+1}^m \mid \hat{\boldsymbol{a}}_{\neg i,t}^m) \end{aligned} \tag{5}$$

where $\neg i$ indicates agents other than i and we use $\hat{z}$ to indicate that z has been relabelled from its original value. Once the message has been relabelled for all agents, we can construct the overall relabelled joint observation by concatenation:

$$\hat{\boldsymbol{o}}_{t+1} = \boldsymbol{o}_{t+1}^e \oplus \hat{\boldsymbol{o}}_{t+1}^m \tag{6}$$

We illustrate our Communication Correction (CC) idea in Figure 1 for the case of two agents, one sending out communication (the Speaker) and the other receiving communication (the Listener).

We experiment with feed-forward policies which condition actions on the immediate observation, but this general idea could also be applied with recurrent networks using a history of observations $\hat{\boldsymbol{a}}_{\neg i,t}^m \sim \boldsymbol{\pi}_{\neg i}(\boldsymbol{a}_{\neg i,t}^m \mid \boldsymbol{h}_{\neg \boldsymbol{i},\boldsymbol{t}})$, for example by using a recurrent relabelling scheme to traverse the replay buffer at regular intervals. In our feedforward case, we sample an extra $\boldsymbol{o}_{t-1}^m$ in order to determine (using other agents' policies) the new $\hat{\boldsymbol{o}}_t^m$, which allows us to relabel at the point of sampling from the replay buffer. Our relabelling approach could also straightforwardly be incorporated with attention-based models which also learn to *whom* to communicate (Das et al., 2019), but for our experiments we assume this is determined by the environment rather than the model.

3.2 Ordered relabelling

One additional complexity to our approach is that the policies of the other agents may themselves by conditioned on received communication in addition to environmental observations. Our initial description ignores this effect, applying only a single correction. However, we can better account for this by sampling from the replay buffer an extra k samples into the past. Starting from the k'th sample into the past, we can set $\hat{\boldsymbol{o}}_{t-k} = \boldsymbol{o}_{t-k}$. Using Equations 5 and 6, we can then relabel according to:

$$\hat{\boldsymbol{o}}_{t-k+1} \sim p(\boldsymbol{o}_{t-k+1} \mid \hat{\boldsymbol{o}}_{t-k}, \boldsymbol{o}_{t-k+1}^e, \boldsymbol{\pi}) \tag{7}$$

We iteratively apply this correction until $\hat{\boldsymbol{o}}_{t+1}$ is generated. In general, this is an approximation if the starting joint observation $\boldsymbol{o}_{t-k}$ depends on communication, but the corrected communication

would likely be less off-environment than before. Furthermore, for episodic environments an exact correction could be found by correcting from the first time step of each episode.

In our experiments we consider a Directed Acyclic Graph (DAG) communication structure which also allows for exact corrections. In general a DAG structure may be expressed in terms of an adjacency matrix D which is nilpotent; there exists some positive integer n such that $D^m = 0, \forall m \geq n$. If s is the smallest such n, we can set $k = s - 1$ which allows information to propagate from the root nodes of the DAG to the leaves. Agents which are not root nodes will not need k updates for the influence of their messages to be propagated and so, for efficiency, for messages c steps into the past we only generate messages which will have a downstream effect c steps later (where $0 < c \leq k$). In general, we call our approach an Ordered Communication Correction (OCC), as opposed to our previous First-step Communication Correction (FCC) which only does one update.

3.3 Implementation

We include the full algorithm in Appendix A.8. We find in our experiments that relabelling can be done rapidly with little computational cost. Although different agents require different relabelling, the majority of the relabelling is shared (more so proportionally for increasing N). For simplicity, we can therefore only relabel a single multi-agent experience for the N agents, and then correct for each agent by setting its own communication action back to its original value, as well as the downstream observations on the next time step. Once the sampled minibatch has been altered for each agent, we then use it for training an off-policy multi-agent algorithm. In our experiments we use MADDPG and Multi-Agent Actor-Critic (MAAC) (see Appendix A.4) (Iqbal & Sha, 2019) but our method could also be applied to other algorithms, such as value-decomposition networks (Sunehag et al., 2017) and QMIX (Rashid et al., 2018).

4 Results

We conduct experiments in the multi-agent particle environment[2], a world with a continuous observation and discrete action space, along with some basic simulated physics. We use feedforward networks for both policy and critic and provide precise details on implementation and hyperparameters in Appendix A.1.

4.1 Cooperative Communication with 5 targets

Our first experiment, introduced by Lowe et al. (2017), is known as Cooperative Communication (Figure 2). It involves two cooperative agents, a Speaker and a Listener, placed in an environment with landmarks of differing colours. On each episode, the Listener must navigate to a randomly selected landmark; and both agents obtain reward proportional to its negative distance from this target. However, whilst the Listener observes its relative position from each of the differently coloured landmarks, it does not know which landmark is the target. Instead, the colour of the target landmark can be seen by the Speaker, which is unable to move. The Speaker can however communicate to the Listener at every time step, and so successful performance on the task corresponds to it helping the Listener to reach the target.

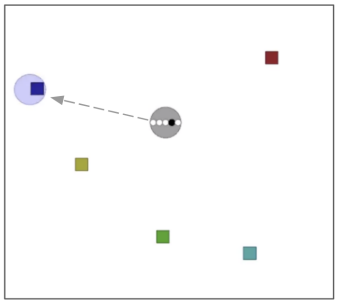

Figure 2: Cooperative Communication with 5 targets. Only the Speaker knows the target colour and must guide the Listener to the correct landmark.

Whilst Lowe et al. (2017) considered a problem involving only 3 landmarks (and showed that decentralised DDPG fails on this task), we increase this to 5. This is illustrated in Figure 2, which shows a particular episode where the dark blue Listener has correctly reached the dark blue square target, due to the helpful communication of the Speaker. We analyse performance on this task in Figure 3. Perhaps surprisingly both MADDPG and MAAC struggle to perfectly solve the problem in this case, with reward values approximately corresponding to only reliably reaching 4 of the

[2]https://github.com/openai/multiagent-particle-envs

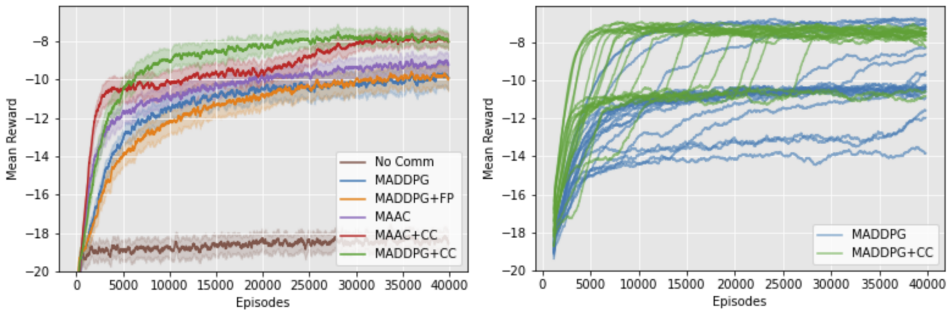

Figure 3: Cooperative Communication with 5 targets. (Left) MADDPG with communication correction (MADDPG+CC) substantially outperforms MADDPG (n=20, shaded region is standard error in the mean). (Right) Smoothed traces of individual MADDPG and MADDPG+CC runs. MADDPG+CC often has rapid improvements in its performance whereas MADDPG is slow to change.

5 targets. We also implement a multi-agent fingerprint for MADDPG (MADDPG+FP) similar to the one introduced by Foerster et al. (2017), by including the training iteration index as input to the critic (see Appendix A.5), but we do not find it to improve performance. By contrast, introducing our communication correction substantially improves both methods, enabling all 5 targets to be reached more often. By looking at smoothed individual runs for MADDPG+CC we can see that it often induces distinctive rapid transitions from the 4 target plateau to the 5 target solution, whereas MADDPG does not. We hypothesise that these rapid transitions are due to our relabelling enabling the Listener to adapt quickly to changes in the Speaker's policy to exploit cases where it has learned to select a better communication action (before it unlearns this).

4.2 Hierarchical Communication

We next consider a problem with a hierarchical communication structure, which we use to elucidate the differences between first-step and ordered communication corrections (MADDPG+FCC vs MADDPG+OCC). Our Hierarchical Communication problem (Figure 4) involves four agents. One agent is a Listener and must navigate to one of four coloured landmarks, but it cannot see what the target colour is. The remaining three Speaker agents can each see different colours which are certain not to be the target colour (indicated by their own colour in the diagram). However, only one Speaker can communicate with the Listener, with the rest forming a communication chain. To solve this task, the first Speaker must learn to communicate what colour it knows not to be correct, the middle Speaker must integrate its own knowledge to communicate the two colours which are not correct (for which there are 6 possibilities), and the final Speaker must use this to communicate the identity of the target landmark to the Listener, which must navigate to the target.

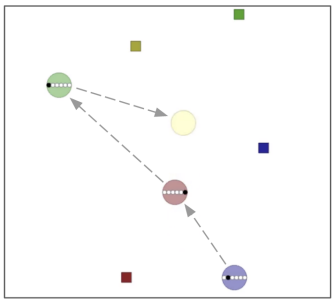

Figure 4: Hierarchical Communication. Three Speakers, with limited information and communicating in a chain, must guide the Listener to the target.

We analyse performance of MADDPG, MADDPG+FCC and MADDPG+OCC on this task in Figure 5. Whilst MADDPG+FCC applies the communication correction for each agent, it only does this over one time step, which prevents newly updated observations being used to compute the next correction. By contrast, the ordered MADDPG+OCC, with $k = 3$, starts from the root node, updates downstream observations and then uses the newly updated observations for the next update and so on (exploiting the DAG structure for more efficient updates). Our results show that MADDPG learns very slowly on this task and performs poorly, and MADDPG+FCC also performs poorly, with no evidence of a significant improvement over MADDPG. By contrast, MADDPG+OCC performs markedly better, learning at a much more rapid pace, and reaching a higher mean performance.

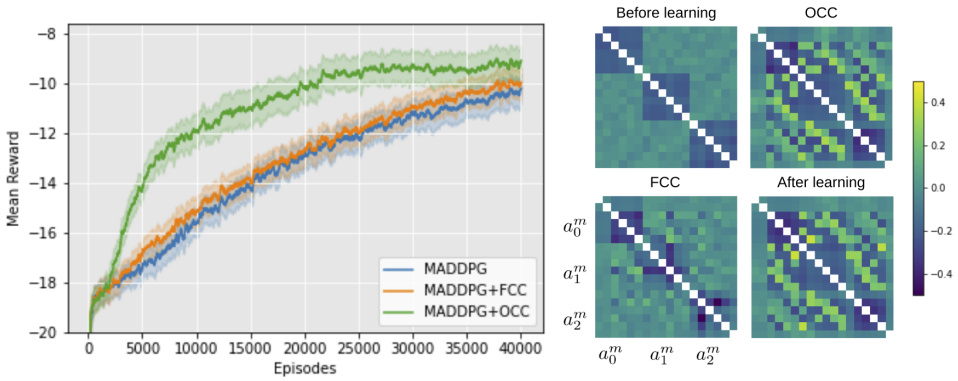

Figure 5: Hierarchical Communication. (Left) MADDPG+OCC substantially outperforms alternatives on this task (n=20). (Right) Correlation matrices for joint communication actions. Past communication from before learning is a poor reflection of communication after learning. OCC applied to past samples recovers this correlation structure whereas FCC only partially recovers this.

We would like to find out if the improved performance may be related to the ability of our method to recover correlations in communication. We therefore also examine the correlation matrices for the vector of joint communication actions. After having trained our MADDPG+OCC agents for $30,000$ episodes, we can compare samples of communication from the starting point of learning and after learning has taken place. We see that the correlation matrices are substantially different, with an intricate structure after learning reflecting the improved performance on the task. Without a communication correction, a 'before learning' sample would be unchanged, and the sample would therefore be a poor reflection of the current social environment. Using MADDPG+FCC we recover some of this structure, but there are still large differences, whereas MADDPG+OCC recovers this perfectly. This indicates that an ordered relabelling scheme is beneficial, and suggests that it may be increasingly important as the depth of the communication graph increases.

4.3 Covert Communication

Finally, we consider a competitive task first introduced by Lowe et al. (2017) called Covert Communication. In this task there are three agents; two Allies, one being a Speaker and another a Listener, and an Adversary. The Speaker sends messages which are received by both the Listener and the Adversary. However, whilst the Speaker would like the Listener to decode the message, it does not want the Adversary to decode the message. Both Speaker and Listener observe a cryptographic 'key', which varies per episode and which the Adversary does not have access to. Reward for the Allies is the difference between how well the Listener decodes the message and how well the Adversary decodes the message (with 0 reward corresponding to both agents decoding equally well).

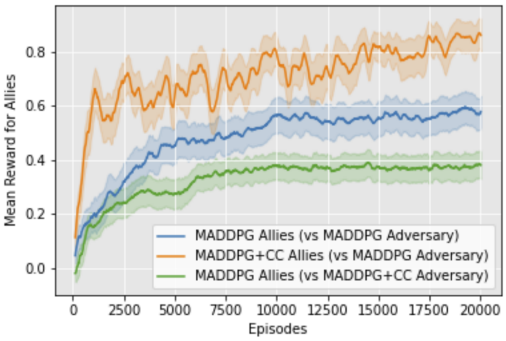

Figure 6: Covert Communication. When the Allies use the CC their performance is improved, whereas when their adversary uses it their performance is diminished (n=20).

One of the reasons this problem is interesting is because, unlike the previous Speaker-Listener problems which were ultimately cooperative, here there are competing agents. This is known to be able to induce large amounts of non-stationarity in communication policies and the environment. We therefore expect our experience relabelling to be effective in such situations, whether it be used for the Allies or the Adversary. Our results in Figure 6 demonstrate this; using the communication correction for the Allies but not the Adversary improves the Allies performance, whereas using it for

the Adversary but not the Allies degrades the Allies performance. We find that this is because the communication correction allows agents to rapidly adapt their policies when their opponents change their policy to reduce their reward (see Appendix A.7 for an analysis).

5 Related Work

Multi-agent RL has a rich history (Busoniu et al., 2008). Communication is a key concept; however, much prior work on communication relied on pre-defined communication protocols. Learning communication was however explored by Kasai et al. (2008) in the tabular case, and has been shown to resolve difficulties of coordination which can be difficult for independent learners (Mataric, 1998; Panait & Luke, 2005). Recent work in the deep RL era has also investigated learning to communicate, including how it can be learned by backpropagating through the communication channel (Foerster et al., 2016; Sukhbaatar et al., 2016; Havrylov & Titov, 2017; Peng et al., 2017; Mordatch & Abbeel, 2018). Although we do not assume such convenient differentiability in our experiments, our method is in general applicable to this case, for algorithms which use a replay buffer. Other approaches which have been used to improve multi-agent communication include attention-based methods (Jiang & Lu, 2018; Iqbal & Sha, 2019; Das et al., 2019), intrinsic objectives (Jaques et al., 2019; Eccles et al., 2019) and structured graph-based communication (Agarwal et al., 2019).

Improvements to multi-agent experience replay were also considered by Foerster et al. (2017) who used decentralised training. Importance sampling as an off-environment correction was only found to provide slight improvements, perhaps due to the classical problem that importance ratios can have large or even unbounded variance (Robert & Casella, 2013), or with bias due to truncation. Here we focus specifically on communicated messages; this allows us to relabel rather than reweight samples and avoid issues of importance sampling. Of course, our method does not alter environment actions and so importance sampling for these may still be beneficial. In addition, it may in some cases be beneficial to condition our relabelled messages on these environment actions, perhaps using autoregressive policies (Vinyals et al., 2017). Another approach that has seen some success is using the training iteration number as a simple 'fingerprint' for the critic (Foerster et al., 2017). Although this can be more effective than importance sampling, we did not find that it improved performance, perhaps because we use centralised rather than decentralised training, which can better handle issues of non-stationarity.

Our approach also bears a resemblance to Hindsight Experience Replay (HER) (Andrychowicz et al., 2017), which can be used for environments which have many possible goals. It works by replacing the goal previously set for the agent with one which better matches the observed episode trajectory, which is particularly valuable in sparse reward problems where any given episode is unlikely to be rewarded. This idea has been applied to hierarchical reinforcement learning (Levy et al., 2018), a field which can address single-agent problems by invoking a hierarchy of communicating agents (Dayan & Hinton, 1993; Vezhnevets et al., 2017). In such systems, goals are set by a learning agent, and one can also relabel its experience by replacing the goal it previously set with one which better reflects the (temporally-extended) observed transition (Nachum et al., 2018). Such ideas could naturally be combined with multi-agent HRL methods (Ahilan & Dayan, 2019; Vezhnevets et al., 2019; Ma & Wu, 2020), however they rely on communication corresponding to goals or reward functions. In contrast, our method can be applied more generally, to any communicated message.

6 Conclusions

We have shown how off-policy learning for communicating agents can be substantially improved by relabelling experiences. Our communication correction exploited the simple communication model which relates a sent message to a received message, and allowed us to relabel the received message with one more likely under the current policy. To address problems with agents who were both senders and receivers, we introduced an ordered relabelling scheme, and found overall that our method improved performance on both cooperative and competitive tasks. In the future it would be interesting to see if this general idea could be applied to other problems involving non-stationary environments, for which acquired information in the present could be used to alter past experience in order to improve future behaviour.

REFERENCES

Akshat Agarwal, Sumit Kumar, and Katia Sycara. Learning transferable cooperative behavior in multi-agent teams. *arXiv preprint arXiv:1906.01202*, 2019.

S Ahilan and P Dayan. Feudal multi-agent hierarchies for cooperative reinforcement learning. In *Workshop on Structure & Priors in Reinforcement Learning (SPiRL 2019) at ICLR 2019*, pp. 1–11, 2019.

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *Advances in Neural Information Processing Systems*, pp. 5048–5058, 2017.

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

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

Abhishek Das, Théophile Gervet, Joshua Romoff, Dhruv Batra, Devi Parikh, Mike Rabbat, and Joelle Pineau. Tarmac: Targeted multi-agent communication. In *International Conference on Machine Learning*, pp. 1538–1546, 2019.

Peter Dayan and Geoffrey E Hinton. Feudal reinforcement learning. In *Advances in neural information processing systems*, pp. 271–278, 1993.

Tom Eccles, Yoram Bachrach, Guy Lever, Angeliki Lazaridou, and Thore Graepel. Biases for emergent communication in multi-agent reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 13111–13121, 2019.

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

Jakob Foerster, Nantas Nardelli, Gregory Farquhar, Triantafyllos Afouras, Philip HS Torr, Pushmeet Kohli, and Shimon Whiteson. Stabilising experience replay for deep multi-agent reinforcement learning. *arXiv preprint arXiv:1702.08887*, 2017.

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

Emil Julius Gumbel. Statistical theory of extreme values and some practical applications. *NBS Applied Mathematics Series*, 33, 1954.

Serhii Havrylov and Ivan Titov. Emergence of language with multi-agent games: Learning to communicate with sequences of symbols. In *Advances in neural information processing systems*, pp. 2149–2159, 2017.

Junling Hu, Michael P Wellman, et al. Multiagent reinforcement learning: theoretical framework and an algorithm. In *ICML*, volume 98, pp. 242–250. Citeseer, 1998.

Shariq Iqbal and Fei Sha. Actor-attention-critic for multi-agent reinforcement learning. In *International Conference on Machine Learning*, pp. 2961–2970, 2019.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

Natasha Jaques, Angeliki Lazaridou, Edward Hughes, Caglar Gulcehre, Pedro Ortega, Dj Strouse, Joel Z Leibo, and Nando De Freitas. Social influence as intrinsic motivation for multi-agent deep reinforcement learning. In *International Conference on Machine Learning*, pp. 3040–3049, 2019.

Jiechuan Jiang and Zongqing Lu. Learning attentional communication for multi-agent cooperation. In *Advances in neural information processing systems*, pp. 7254–7264, 2018.

Tatsuya Kasai, Hiroshi Tenmoto, and Akimoto Kamiya. Learning of communication codes in multi-agent reinforcement learning problem. In *2008 IEEE Conference on Soft Computing in Industrial Applications*, pp. 1–6. IEEE, 2008.

Andrew Levy, Robert Platt, and Kate Saenko. Hierarchical reinforcement learning with hindsight. *arXiv preprint arXiv:1805.08180*, 2018.

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. *arXiv preprint arXiv:1509.02971*, 2015.

Michael L Littman. Markov games as a framework for multi-agent reinforcement learning. In *Machine Learning Proceedings 1994*, pp. 157–163. Elsevier, 1994.

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*, pp. 6379–6390, 2017.

Ryan Lowe, Jakob Foerster, Y-Lan Boureau, Joelle Pineau, and Yann Dauphin. On the pitfalls of measuring emergent communication. *arXiv preprint arXiv:1903.05168*, 2019.

Jinming Ma and Feng Wu. Feudal multi-agent deep reinforcement learning for traffic signal control. In *Proceedings of the 19th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2020.

Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, 2016.

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

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Igor Mordatch and Pieter Abbeel. Emergence of grounded compositional language in multi-agent populations. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Ofir Nachum, Shane Gu, Honglak Lee, and Sergey Levine. Data-efficient hierarchical reinforcement learning. *arXiv preprint arXiv:1805.08296*, 2018.

Liviu Panait and Sean Luke. Cooperative multi-agent learning: The state of the art. *Autonomous agents and multi-agent systems*, 11(3):387–434, 2005.

Peng Peng, Ying Wen, Yaodong Yang, Quan Yuan, Zhenkun Tang, Haitao Long, and Jun Wang. Multiagent bidirectionally-coordinated nets: Emergence of human-level coordination in learning to play starcraft combat games. *arXiv preprint arXiv:1703.10069*, 2017.

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. *arXiv preprint arXiv:1803.11485*, 2018.

Christian Robert and George Casella. *Monte Carlo statistical methods*. Springer Science & Business Media, 2013.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

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

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. *arXiv preprint arXiv:1706.05296*, 2017.

Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. Feudal networks for hierarchical reinforcement learning. In *International Conference on Machine Learning*, pp. 3540–3549, 2017.

Alexander Sasha Vezhnevets, Yuhuai Wu, Remi Leblond, and Joel Leibo. Options as responses: Grounding behavioural hierarchies in multi-agent rl. *arXiv preprint arXiv:1906.01470*, 2019.

Oriol Vinyals, Timo Ewalds, Sergey Bartunov, Petko Georgiev, Alexander Sasha Vezhnevets, Michelle Yeo, Alireza Makhzani, Heinrich Küttler, John Agapiou, Julian Schrittwieser, et al. Starcraft ii: A new challenge for reinforcement learning. *arXiv preprint arXiv:1708.04782*, 2017.

Ermo Wei, Drew Wicke, David Freelan, and Sean Luke. Multiagent soft q-learning. *arXiv preprint arXiv:1804.09817*, 2018.

A APPENDIX

A.1 HYPERPARAMETERS

For all algorithms and experiments, we used the Adam optimizer with a learning rate of 0.001 and $\tau = 0.01$ for updating the target networks. The size of the replay buffer was 10^7 and we updated the network parameters after every 100 samples added to the replay buffer. We used a batch size of 1024 episodes before making an update. We trained with 20 random seeds for all experiments and show using a shaded region the standard error in the mean.

For MADDPG we use the implementation of Iqbal & Sha (2019)[3]. For MADDPG and all MADDPG variants, hyperparameters were optimised using a line search centred on the experimental parameters used in Lowe et al. (2017) but with 64 neurons per layer in each feedforward network (each with two hidden layers). We found a value of $\gamma = 0.75$ worked best on Cooperative Communication with 6 landmarks evaluated after $50,000$ episodes. We use the Straight-Through Gumbel Softmax estimator with an inverse temperature parameter of 1 to generate discrete actions (see A.2).

For MAAC and MAAC+CC, we use the original implementation with unchanged parameters of Iqbal & Sha (2019)[4] . For our feedforward networks this corresponded to two hidden layers with 128 neurons per layer, 4 attend heads and $\gamma = 0.99$.

A.2 DISCRETE OUTPUT WITH STRAIGHT-THROUGH GUMBEL SOFTMAX

We work with a discrete action space and so for MADDPG, following prior work (Lowe et al., 2017; Iqbal & Sha, 2019), we use the Gumbel Softmax Estimator (Jang et al., 2016; Maddison et al., 2016). In particular we use the Straight-Through Gumbel Estimator, which uses the Gumbel-Max trick (Gumbel, 1954) on the forward pass to generate a discrete sample from the categorical distribution, and the continuous approximation to it, the Gumbel Softmax, on the backward pass, to compute a biased estimate of the gradients.

The Gumbel Softmax enables us to compute gradients of a sample from the categorical distribution. Given a categorical distribution with class probabilites π, we can generate k-dimensional sample vectors $y \in \Delta^{n-1}$ on the simplex, where:

$$y_i = \frac{\exp((g_i + \log \pi_i)\beta)}{\sum_{j=1}^{k} \exp((g_j + \log \pi_j)\beta)} \tag{8}$$

for $i = 1, \ldots, k$, where g_i are samples drawn from Gumbel(0,1)[5] and β is an inverse temperature parameter. Lower values of β will incur greater bias, but lower variance of the estimated gradient.

A.3 MADDPG

Although the original MADDPG proposed the multi-agent policy gradient of Equation 1, this can cause over-generalisation when $\boldsymbol{a} \sim \mathcal{D}$ is far from the current policies of other agents (Wei et al., 2018). We use a correction introduced by Iqbal & Sha (2019) (for all variants of MADDPG), by sampling actions from the feedforward policy μ rather than the replay buffer, and providing this as input to the centralised critic.

$$\nabla_{\theta_i} J(\theta_i) = \mathbb{E}_{\boldsymbol{o}\sim\mathcal{D},\boldsymbol{a}\sim\boldsymbol{\mu}}[\nabla_{\theta_i}\mu_i(o_i)\nabla_{a_i}Q_i^{\boldsymbol{\mu}}(\boldsymbol{o},\boldsymbol{a})|_{a_i=\mu_i(o_i)}]. \tag{9}$$

One minor difference from Iqbal & Sha (2019) is that they generate the discrete action of other agents using the greedy action whereas we instead take a Gumbel-Softmax sample with $\beta = 1$.

[3]https://github.com/shariqiqbal2810/maddpg-pytorch

[4]https://github.com/shariqiqbal2810/MAAC

[5]using the procedure $u \sim \mathrm{Uniform}(0,1)$ and computing $g = -\log(-\log(u)))$

A.4 MULTI-AGENT ACTOR ATTENTION CRITIC

Introduced by Iqbal & Sha (2019), MAAC learns the centralised critic for each agent by selectively paying attention to information from other agents. This allows each agent to query other agents for information about their observations and actions and incorporate that information into the estimate of its value function. In the case of MAAC+CC there is just a single shared critic and so we provide all altered samples to the same critic and do N critic updates (where N is the number of agents), and compare to MAAC which also does N critic updates but with the unaltered samples.

A.5 MULTI-AGENT FINGERPRINT

We incorporate a multi-agent fingerprint similar to the one introduced by Foerster et al. (2017). We do this by additionally storing the training iteration number. As training iteration numbers can get very large, and we find this hurts performance (we do not do use batch normalisation), we divide this number by $100,000$ before storing it (we typically train over $40,000$ episodes, which corresponds to $1,000,000$ iterations). We then train MADDPG in the conventional way, but additionally provide this number to the centralised critic. Foerster et al. (2017) also varied exploration and so provide the exploration parameter to the decentralised critics. We do not do this as we keep exploration fixed throughout training (with Gumbel $\beta = 1$).

A.6 EXPERIMENTAL ENVIRONMENTS

We provide a description of our environments in the main text but provide further information here. The episode length for all experiments was 25 time steps. For Cooperative Communication we alter the original problem by including two extra landmarks and by providing reward according to the negative distance from the target rather than the negative squared difference (as this elicits better performance in general). Our reward plots show the reward per episode.

For our novel task, Hierarchical Communication, the information as to the target landmark is distributed amongst Speakers who cannot move and communicate in a chain to the Listener. As with Cooperative Communication we display rewards per episode and use the negative distance rather than the negative square distance.

For Covert Communication we plot the average reward per time step and use the original task, with one minor change of dividing the reward by 2. This ensures that the magnitude of the Allies rewards is less than or equal to 1 (for example, when the Allies are always right and the Adversary is always wrong the Allies reward is 1).

A.7 ANALYSIS OF COVERT COMMUNICATION

When used for the Allies, our communication correction substantially improved their performance. Interestingly, we found that rather than use the key, MADDPG+CC is able to learn communication actions which mislead its adversary into making the wrong choice, whilst simultaneously helping the Listener to make the right choice. This enables MADDPG+CC to achieve more reward than could be achieved against a random opponent (on average this would be 0.5). To emphasise this, we also trained agents on the same problem but without providing the Allies with the key. We find that when both Allies and the Adversary are trained using MADDPG the reward for the Allies is zero, because the Allies cannot adapt any faster than the Adversary. By contrast, performance for the MADDPG+CC Allies against a MADDPG opponent remains very good, with the MADDPG+CC Allies continuing to out-adapt the Adversary, causing it to perform even worse than random.

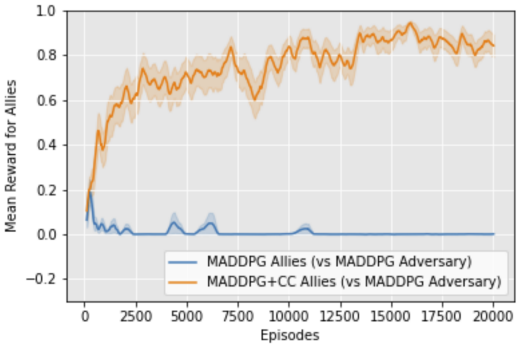

Figure 7: Covert Communication without the key (n=20). MADDPG+CC performs much better than MADDPG.

A.8 ALGORITHM

We outline our Ordered Communication Correction algorithm here (First-step CC can be recovered by setting $K = 1$). We write it for the general case, although fewer updates can be used to achieve equivalent results if the communication graph is a DAG. For notational simplicity we use $\boldsymbol{a} \sim \boldsymbol{\pi}(\boldsymbol{a} \mid \boldsymbol{o})$ to indicate that each agent's policy π_i, receiving input o_i was used to generate a_i and thus collectively the joint action $\boldsymbol{a}$. Note that our description of the set of actions $\mathcal{A}_i = \mathcal{A}_i^e \times \mathcal{A}_i^m$ allows agents to both act and communicate simultaneously; in cases where agents only do one of these we allow for a 'no op' action.

An alternative way to define actions for each agent $\mathcal{A}_i$ is to divide them into disjoint environment actions $\mathcal{A}_i^e$ and explicit communication actions $\mathcal{A}_i^m$ such that $\mathcal{A}_i^e \cup \mathcal{A}_i^m = \mathcal{A}_i$ and $\mathcal{A}_i^e \cap \mathcal{A}_i^m = \emptyset$ (Lowe et al., 2019). To deal with the former case one would require only minor alterations to the algorithm.

Algorithm 1: Ordered Communication Correction

Given:

- An off-policy multi-agent RL algorithm $\mathbb{A}$ using centralised training with decentralised policies
- One-step communication under model $p(\boldsymbol{o}_{t+1}^m \mid \boldsymbol{a}_t^m)$ which relates sent messages to received messages
- Feed-forward policies

Initialise $\mathbb{A}$;
Initialise replay buffer $\mathcal{D}$;
Initialise number of steps to correct K;
foreach *episode* **do**
  Receive initial state $\boldsymbol{x}$ and joint observations $\boldsymbol{o}$;
  foreach *step of episode* **do**
    Select joint action $\boldsymbol{a} \sim \boldsymbol{\pi}(\boldsymbol{a} \mid \boldsymbol{o})$ using current policies (with exploration);
    Execute actions $\boldsymbol{a}$ which transitions environment state to $\boldsymbol{x'}$and observe rewards $\boldsymbol{r}$, and next state $\boldsymbol{o'}$;
    Store $(\boldsymbol{o}, \boldsymbol{a}, \boldsymbol{r}, \boldsymbol{o'})$ in replay buffer $\mathcal{D}$;
    $\boldsymbol{x} \leftarrow \boldsymbol{x'}$;
    Sample from $\mathcal{D}$ at random indexes $\boldsymbol{t}$ a multi-agent minibatch
    $\boldsymbol{B_t} = (\boldsymbol{o_{t-K}}, \ldots, \boldsymbol{o_t}, \boldsymbol{a_t}, \boldsymbol{r_{t+1}}, \boldsymbol{o_{t+1}})$;
    $\boldsymbol{\hat{o}_{t-K}} = \boldsymbol{o_{t-K}}$;
    for $k = K, \ldots, 0$ **do**
      Compute new messages $\boldsymbol{\hat{a}_{t-k}^m} \sim \boldsymbol{\pi}(\boldsymbol{a_{t-k}^m} \mid \boldsymbol{\hat{o}_{t-k}})$;
      Compute new observed messages $\boldsymbol{\hat{o}_{t-k+1}^m} \sim p(\boldsymbol{o_{t-k+1}^m} \mid \boldsymbol{\hat{a}_{t-k}^m})$;
      $\boldsymbol{\hat{o}_{t-k+1}} = \boldsymbol{o_{t-k+1}^e} \oplus \boldsymbol{\hat{o}_{t-k+1}^m}$;
    end for
    $\boldsymbol{\hat{a}_t} = \boldsymbol{a_t^e} \oplus \boldsymbol{\hat{a}_t^m}$;
    for $i = 0, \ldots, N$ **do**
      Set agent i's sent and received messages back to original value (where $\boldsymbol{l}$ is index of receiving agents);
      $\boldsymbol{\hat{a}_t^{m,i}} = \boldsymbol{\hat{a}_{t,\neg i}^m} \oplus a_{t,i}^m$;
      $\boldsymbol{\hat{o}_t^{m,i}} = \boldsymbol{\hat{o}_{t,\neg l}^m} \oplus \boldsymbol{o_{t,l}^m}$;
      $\boldsymbol{\hat{o}_{t+1}^{m,t}} = \boldsymbol{\hat{o}_{t+1,\neg l}^m} \oplus \boldsymbol{o_{t+1,l}^m}$;
      $\boldsymbol{\hat{B}_t^i} = (\boldsymbol{\hat{o}_t^i}, \boldsymbol{\hat{a}_t^i}, \boldsymbol{r_{t+1}}, \boldsymbol{\hat{o}_{t+1}^i})$;
      Perform one step of optimisation for agent i using $\mathbb{A}$ with relabelled minibatch $\boldsymbol{\hat{B}_t^i}$
    end for
  end foreach
end foreach